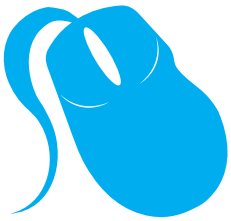


Informática

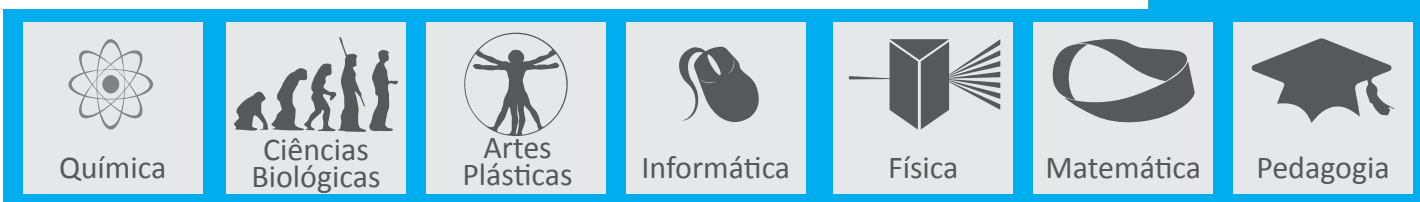
Arquitetura de Computadores

Marcial Porto Fernandez

3ª edição
Fortaleza - Ceará

Ed UECE

2015



Sumário

Apresentação

A disciplina Arquitetura de Computadores trata de explicar o funcionamento de um computador a partir de blocos funcionais básicos interligados. Um arquiteto que faz um projeto de edificação define as unidades funcionais: quartos, sala, cozinha, etc e as ligações entre elas: corredor, portas, janelas, etc. O arquiteto de computadores também define as unidades funcionais de um computador (memória, processador) e suas ligações entre as unidades funcionais: barramentos paralelos, seriais, etc.

A função básica de um computador é processar dados. Para isso ele precisa movê-los da memória até a unidade central de processamento, armazenar os resultados na memória onde eles possam ser encontrados mais tarde e controlar todas essas operações de transportar, armazenar e processar. Portanto, a Arquitetura de Computadores estabelece a forma como as unidades funcionais básicas, a unidade central de processamento, a memória e as unidades, a entrada e a saída são interligadas para realizar suas quatro funções elementares: processar, armazenar dados, mover dados ou controlar todas essas funções. A Arquitetura de Computadores não entra em detalhes de implementação dos blocos funcionais, ela trata cada módulo como uma unidade fechada com características específicas.

Um computador eletrônico realiza essas funções através de circuitos eletrônicos. A primeira decisão importante é definir como converter os dados humanos em dados entendidos pelo computador para que ele possa processá-los. De posse desses dados precisamos realizar as operações de processamento, basicamente, comparar e realizar operações matemáticas. Como um computador eletrônico entende apenas dois estados, 0 e 1, geralmente adotamos uma notação binária, e todos os dados precisarão ser convertidos para essa representação. Com os dados representados na forma binária precisamos realizar o processamento através de circuitos somadores, ou unidades lógicas e aritmética. Finalmente, os dados precisam se armazenados em um dispositivo de memória, seja ela temporária ou permanente. Para realizar o transporte de dados e, entre todas essas unidades, também precisamos projetar estruturas para conduzi-los.

Decisões de projeto de uma Arquitetura de Computadores são marcantes para se alcançar um determinado desempenho a um determinado custo. Aí está a arte do Arquiteto de Computadores: obter maiores desempenhos e capacidade com menores custos.

O autor

Parte 1

Conceitos Básicos

Capítulo 1

Introdução à Arquitetura de Computadores

Objetivos

Neste capítulo vamos apresentar os conceitos básicos de Arquitetura de Computadores. Iniciamos com a definição de alguns conceitos que serão necessários para o entendimento de uma arquitetura de computadores. Depois apresentamos uma pequena história dos computadores eletrônicos, desde a sua invenção até os dias de hoje. Finalmente, apresentamos os componentes gerais de uma arquitetura de computadores.

Introdução

O ser humano sempre precisou realizar cálculos, sejam eles para armazenar a comida, contar os animais ou construir uma casa. Os primeiros computadores do mundo eram as pessoas. O "computador" era uma profissão cujo trabalho era executar os cálculos repetitivos exigidos para computar tabelas de navegação, cartas de marés e posições planetárias. Devido aos erros decorrente do cansaço dos "computadores" humanos, durante vários séculos, os inventores têm procurado por uma maneira de mecanizar essa tarefa.

Até aproximadamente metade do século XX, havia apenas máquinas rudimentares que auxiliavam a realização de cálculos. A partir da invenção da válvula eletrônica, do transistor e do circuito integrado, foi possível construir máquinas eletrônicas com maior capacidade e velocidade. Ela é chamada mais precisamente "computador eletrônico digital de uso geral", e é dessa máquina que vamos falar a partir de agora.

Os computadores são eletrônicos porque manipulam dados usando circuitos de chaveamento eletrônico, sejam eles válvulas eletrônicas, transistores e, mais recentemente, circuitos integrados. **Digital** significa que o computador guarda e manipula todos os dados internamente sob a forma de números (todos os dados numéricos, todos os dados de texto e mesmo os sons e as figuras são armazenadas como números). O dígito significa "dedo" e, desde que os povos

começaram a contar com seus dedos, o nome **dígito** é aplicado à representação de números. O computador é de uso geral porque pode ser programado para executar uma grande variedade de aplicações (diferente de um computador de propósito específico, projetado para executar somente uma função).

1. Arquitetura e Organização de Computadores

Os termos Arquitetura de Computadores e Organização de Computadores têm significados distintos para descrever um sistema de computação. Há um consenso na comunidade científica de que o termo "Arquitetura de Computador" refere-se aos atributos que são vistos pelo programador. Por exemplo: conjunto de instruções, número de bits de representação dos dados, endereçamento de memória e mecanismos de entrada e saída, são exemplos de atributos de arquitetura de computadores.

O termo "Organização de Computadores" se refere às unidades operacionais e suas interconexões para uma determinada arquitetura e são transparentes para o programador. Por exemplo, tecnologias dos componentes eletrônicos, sinais de controle, interfaces entre os computadores e periféricos são exemplos de atributos de organização de computadores.

Apesar deste livro tratar especificamente de Arquitetura de Computadores, a Parte 2 apresenta uma introdução aos conceitos de Lógica Digital, mais apropriada para a área de Organização de Computadores, com o objetivo de facilitar o entendimento das demais unidades.

2. Hardware e Software

Um computador é um dispositivo que executa quatro funções: recebe dados de entradas (converte dados do mundo exterior para o universo eletrônico); armazena os dados (de forma eletrônica); processa dados (executa operações matemáticas e lógicas) e exibe os dados de saídas (mostra os resultados para os usuários através de uma tela).

Um computador consiste no *hardware* e no *software*. O *hardware* é o equipamento físico: o próprio computador e os periféricos conectados. Os periféricos são todos os dispositivos ligados ao computador para finalidades de entrada, saída e armazenamento dos dados (tais como um teclado, um monitor de vídeo ou um disco rígido externo).

O *software* consiste nos programas e nos dados associados (informação) armazenados no computador. Um programa é uma sequência de instruções que o computador segue com o objetivo de manipular dados. A possibilidade de incluir ou excluir programas diferentes é a fonte de versatilidade de um computador. Sem programas, um computador é apenas *hardware* de alta

tecnologia que não faz nada. Mas, com a sequência de instruções detalhadas, descrevendo cada passo do programa (escrito por seres humanos), o computador pode ser usado para muitas tarefas que variam do processamento de texto à simulação de padrões de tempo globais.

Como um usuário, você irá interagir com os programas que funcionam em seu computador através dos dispositivos de entrada conectados a ele, tal como um *mouse* e um teclado. Você usa esses dispositivos para fornecer a entrada (tal como o texto de um relatório em que você está trabalhando) e para dar comandos ao programa (tal como a definição de que uma frase do texto vai aparecer com formato negrito). O programa fornecerá a saída (os dados resultantes das manipulações dentro do computador) através de vários dispositivos de saída (tal como um monitor ou uma impressora).

Para refletir

1. Para você, qual é a categoria mais importante de um computador, o *hardware* ou *software*. Um pode ser utilizado sem o outro?
2. Pare um instante e identifique à sua volta dispositivos que dispõe de um computador. Você observou como computadores estão em todos os lugares e como são importantes para a nossa vida?

3. História do Computador Eletrônico

Apesar de haver relatos históricos de máquinas que realizassem cálculos matemáticos, como a Máquina de Pascal (1642), a máquina de Charles Babbage (1822) até o Mark I da Universidade de Harvard (1944), o primeiro computador totalmente eletrônico foi o ENIAC (1945).

Para classificar as várias etapas do desenvolvimento dos computadores eletrônicos de acordo com a tecnologia utilizada, eles foram classificados em quatro gerações, que veremos a seguir:

3.1. Primeira Geração

Os computadores de primeira geração são baseados em tecnologias de válvulas eletrônicas. Essa geração inicia em 1943 e vai até 1959. Os computadores da primeira geração normalmente paravam de funcionar após poucas horas de uso. Tinham dispositivos de entrada/saída rudimentares, calculavam com uma velocidade de milésimos de segundo e eram programados em linguagem de máquina.

Em 1943, um projeto britânico, sob a liderança do matemático Alan Turing, colocou em operação o COLOSSUS, utilizado para decodificar as mensagens criptografadas pela máquina Enigma, utilizadas pelos alemães na 2ª

Guerra Mundial. Sua característica mais inovadora era a substituição de relés eletromecânicos por válvula eletrônica[1]. Apesar de ter uma arquitetura de computador, ainda ele não pode ser chamado de "uso geral" pois realizava apenas uma função específica. Essa máquina usava 2.000 válvulas eletrônicas.

Em 1945, surgiu o ENIAC - Electronic Numerical Integrator and Computer, ou seja, "Computador e Integrador Numérico Eletrônico", projetado para fins militares, pelo Departamento de Material de Guerra do Exército dos EUA, na Universidade de Pensilvânia. Foi o primeiro computador digital eletrônico de grande escala, projetado por John W. Mauchly e J. Presper Eckert. Esse é considerado pelos pesquisadores como o primeiro computador eletrônico de uso geral, isto é, pode realizar diferentes funções a partir da troca de um programa, apesar da sua reprogramação levar semanas para ser concluída.

O ENIAC tinhas 17.468 válvulas, 500.000 conexões de solda, 30 toneladas de peso, 180 m² de área construída, 5,5 m de altura, 25 m de comprimento e realizava uma soma em 0,0002 s.

O ENIAC tinha um grande problema: por causa do grande número de válvulas eletrônicas, operando à taxa de 100.000 ciclos por segundo, havia 1,7 bilhão de chances a cada segundo de que uma válvula falhasse, além de superaquecer. As válvulas liberavam tanto calor (aproximadamente 174 KW), que, mesmo com os ventiladores à temperatura ambiente, chegava a 67°C. Então Eckert aproveitou a idéia utilizada em órgãos eletrônicos, fazendo com que as válvulas funcionassem sob uma tensão menor que a necessária, reduzindo, assim, as falhas para 1 ou 2 por semana.

Mesmo com 18.000 válvulas, o ENIAC podia armazenar somente 20 números de cada vez. Entretanto, graças à eliminação das peças móveis ele funcionava muito mais rapidamente do que os computadores eletromecânicos. No entanto, o maior problema do ENIAC era a reprogramação. A programação era realizada através da ligação de fios e interruptores em um painel. A mudança de código de programa levava semanas.

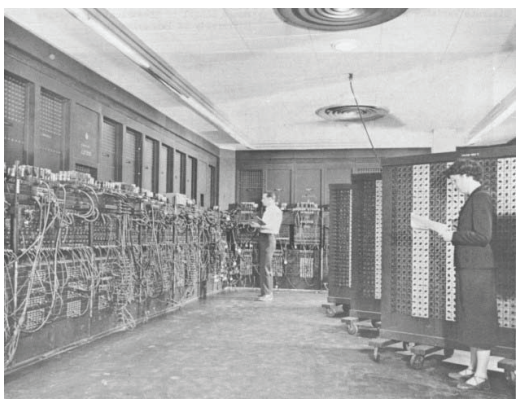


Figura 1 - ENIAC – O primeiro grande computador eletrônico [foto Exército EUA]

[1] Válvula Eletrônica é um dispositivo eletrônico formado por um invólucro de vidro de alto vácuo chamada **ampola** contendo vários elementos metálicos. A válvula serve para comutar circuito e amplificar sinal elétrico.

Eckert e Mauchly se juntaram ao matemático John Von Neumann em 1944 para projetar o EDVAC (Electronic Discrete Variable Automatic Computer), que abriu caminho para o programa armazenado. Ele foi um sucessor do ENIAC e usava numeração binária em vez de decimal e já utilizava o conceito de programa através de cartão perfurado. Isso permitia ao EDVAC mudanças de programas mais rápidas em comparação ao ENIAC.

No entanto, as principais contribuições do EDVAC foi a arquitetura de processador, e memória e unidades de entrada e saída interligadas por um barramento. Essa arquitetura, conhecida como **Arquitetura Von Neumann,** e utilizada até hoje pela maioria dos computadores. Mostramos a seguir a arquitetura do EDVAC.

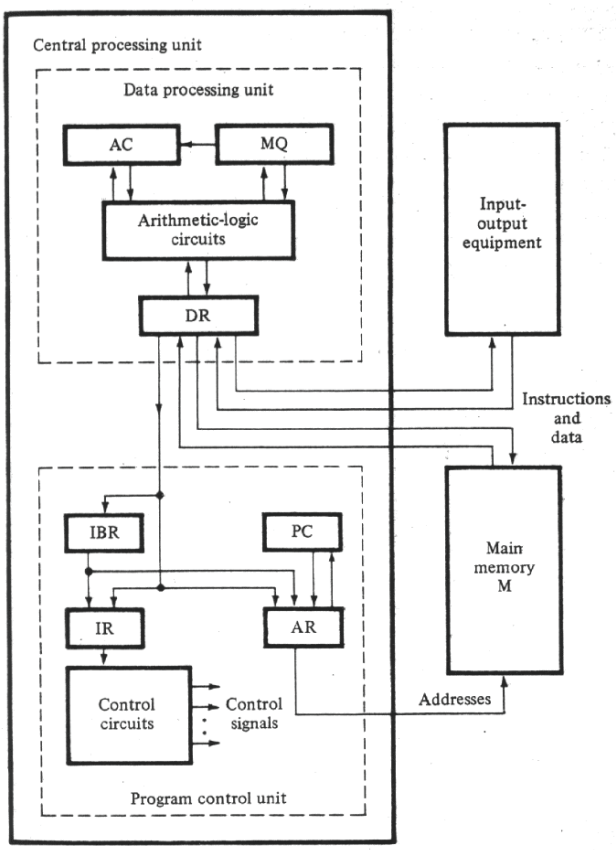


Figura 2 - Arquitetura do EDVAC (Arquitetura von Neumann)

Apenas em 1951 surgiram os primeiros computadores produzidos em escala comercial, como o UNIVAC e o IBM 650.

3.2. Segunda Geração

Nos equipamentos da segunda geração, que durou de 1959 a 1965, a válvula foi substituída pelo transistor[2]. O transistor foi criado em 1947, no Bell Laboratories, por William Shockley e J.Brattain. Seu tamanho era 100 vezes menor que o da válvula, não precisava esperar um tempo para aquecimento, consumia menos energia, era mais rápido e mais confiável.

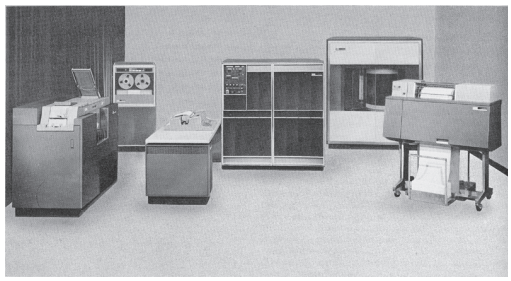


Figura 3 - IBM 1401 [foto Computer History Museum]

[2] Transistor é um dispositivo eletrônico construído com material semicondutor (germânio ou silício), que funciona como chave de circuito ou amplificador. Ele substituiu a válvula eletrônica por ser menor, por dissipar menos calor e ter por uma durabilidade maior.

O primeiro computador experimental totalmente com transistor foi construído na Universidade de Manchester e entrou em operação em 1953. Uma segunda versão do Computador de Transistor foi construída em 1955.

Os computadores da segunda geração faziam cálculos em microssegundos e eram mais confiáveis que os computadores da primeira geração. Os seus representantes comerciais clássicos foram o IBM 1401 e o IBM 7094, já totalmente transistorizado. Outra característica importante, além do uso de transistor, era a utilização de memória com núcleo de ferrite e fita magnética para armazenamento de dados. Essas máquinas já utilizavam linguagem de programação FORTRAN e COBOL.

3.3. Terceira Geração

A terceira geração, que durou de 1964 a 1970, foi marcada pela a substituição dos transistores pela tecnologia de circuitos integrados - transistores e componentes eletrônicos montados em um único substrato. O circuito integrado foi inventado por Jack St. Clair Kilby e Robert Noyce e permitiu a compactação dos computadores construídos com transistor. Nessa geração, o cálculo de operações matemáticas chegava à ordem de nanossegundos além do aumento da capacidade, redução das dimensões físicas e redução na dissipação de calor.

Figura 4 - IBM 360 [foto Computer History Museum]

O principal computador comercial da terceira geração foi o IBM 360, lançado em 1964. Ele dispunha de muitos modelos e de várias opções de expansão que realizava mais de 2 milhões de adições por segundo e cerca de 500 mil multiplicações. O System/360 foi um dos maiores sucessos de venda da IBM.

3.4. Quarta Geração

A quarta geração iniciou em 1971 e dura até hoje. Ela continuou com a utilização de circuitos integrados, mas é diferenciada da terceira geração pela utilização do microprocessador, isto é, um circuito integrado que reúne todas as funções do computador.

Em novembro de 1971, a Intel lançou o primeiro microprocessador comercial, o 4004. Ele foi desenvolvido para um fabricante de calculadoras eletrônicas, a japonesa Busicom, como uma alternativa ao circuito eletrônico que era utilizado. Juntamente com o desenvolvimento do circuito integrado de memória RAM, inventada por Robert Dennard, permitiu a construção do microcomputador, equipamento pequenos com grande capacidade de processamento. O 4004 era capaz de realizar apenas 60.000 instruções por segundo, mas seus sucessores, 8008, 8080, 8086, 8088, 80286, 80386, 80486, etc, ofereciam capacidades e velocidades cada vez maiores em velocidades cada vez maiores.

A vantagem de um circuito integrado não é apenas a redução das dimensões dos componentes de um computador, mas a possibilidade de se produzir componentes em massa reduzindo seu custo. Todos os elementos no circuito integrado são fabricados simultaneamente através de um número pequeno de máscaras óticas que definem a geometria de cada camada. Isso acelera o processo de fabricação do computador, além de reduzir o custo, da mesma forma que a invenção da impressão por Gutenberg possibilitou a difusão e barateamento dos livros no século XVI.

Um computador transistorizado de 1959 continha 150.000 transistores em uma área de 10 m^2. Esses transistores eram infinitamente menores que as válvulas eletrônicas que ele substituiu, mas eram elementos individuais que exigiam um conjunto individual para fixação. Nos anos 80, essa quantidade de transistores podia ser fabricada simultaneamente em um único circuito integrado. Um microprocessador atual *Pentium 4* contém 42.000.000 transistores fabricados simultaneamente em uma pastilha de silício do tamanho de uma unha.

Figura 5 - Microcomputador IBM PC [foto Computer History Museum]

Para refletir

1. Qual foi o maior ganho quando os computadores passaram a ser construídos com válvulas eletrônicas em vez de mecanismos eletromecânicos?
2. Qual o principal problema da primeira geração de computadores e como ele foi resolvido para permitir sua utilização de forma confiável?
3. O que a invenção do transistor trouxe para a evolução dos computadores eletrônicos?
4. Qual a semelhança entre o processo de fabricação de um circuito integrado com a imprensa? O que isso propiciou?
5. Qual a característica que diferencia a terceira da quarta geração de computadores, se ambas utilizam o circuito integrado?

4. Componentes de uma Arquitetura de Computador

Um computador é construído com vários componentes individuais. As ligações entre os diversos componentes é a essência da Arquitetura de Computadores. A forma como os componente se interligam e se comunicam vai determinar a capacidade e velocidade de processamento de uma determinada arquitetura.

A primeira seção mostra a arquitetura geral de um computador. Em seguida, apresentamos a Arquitetura Harvard e a Arquitetura Von Neumann, importantes para entender as Arquiteturas usadas atualmente. Finalmente apresentamos a Lei de Moore, um prognóstico de evolução dos microprocessadores e alguns comentários sobre os limites dela.

4.1. Componentes de um computador

A Figura 6 mostra a arquitetura simplificada de um computador. Os computadores modernos podem ter uma arquitetura bem mais complexa do que esta, mas os blocos principais estão aqui representados.

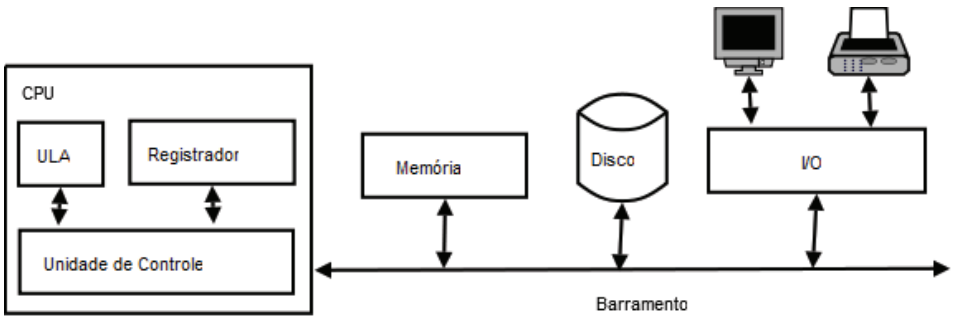


Figura 6 - Arquitetura de um Computador

O principal módulo é a Unidade Central de Processamento, usualmente conhecida como CPU (Central Processing Unit). A CPU é o "cérebro" do computador, onde todo o processamento é realizado. Ela é constituída por três submódulos: a Unidade Lógica e Aritmética (ULA), os Registradores e a Unidade de Controle. Nos computadores modernos, a CPU é construída em um único Circuito Integrado (chip).

A Unidade Lógica e Aritmética (ULA), também conhecida como *Arithmetic and Logic Unit* (ALU), é responsável por realizar as operações do computador, como soma e subtração, assim como as funções lógicas, OU, E, NÃO e OU Exclusivo. Algumas ULAs mais sofisticadas realizam também multiplicações e divisões.

Os registradores são memórias utilizadas para realizar as operações na CPU. Essas memórias são muito rápidas e de tamanho pequeno, geralmente suficiente para guardar menos de uma dezena de valores. Além dos registradores, é comum haver uma memória de rascunho dentro da CPU, chamada de **Cache**. Ela é utilizada para aumentar a velocidade de processamento, reduzindo o tempo de acesso à memória externa.

A Unidade de Controle é responsável por controlar todo o funcionamento da CPU e, também, de todo o computador. Ela controla o processamento entre ULA e Registrador e interage com o barramento externo onde ficam os periféricos. A Unidade de Controle tem uma função chave, que é a interpretação do código do programa que irá nortear os comandos do processamento.

Continuando na Figura 6, podemos observar um barramento e três módulos externos à CPU: Memória, Disco e Interface de Entrada e Saída (I/O).

O barramento serve para interligar todos os componentes internos do computador. Para se obter altas velocidades, esse barramento é, geralmente, paralelo (os dados são enviados paralelamente). Nessa figura representamos apenas um barramento, mas, geralmente, ele é dividido em barramento de dados (onde os dados trafegam), barramento de endereço (onde indicamos a localização de cada dado) e o barramento de controle (onde indicamos o comando, por exemplo, ler, escrever, copiar, etc). Em um computador real, existem vários níveis de barramento com velocidades diferentes. Quanto mais próximo da CPU mais rápido ele é, e quanto mais próximo dos periféricos mais lento ele é.

A Memória é constituída por um conjunto de memória semicondutora utilizada para armazenar os dados temporariamente. Podemos dizer que essa é a memória de trabalho do computador e geralmente fica localizada na placa mãe do computador. Essa memória é mais lenta que os registradores, mas é mais rápida que as unidades de armazenamento em disco. A memória semicondutora geralmente perde os dados quando o computador é desligado, por isso deve-se usar uma unidade de armazenamento permanente, o disco.

O disco consiste em uma unidade eletromecânica que armazena os dados em um disco magnético que mantém as informações mesmo quando o computador é desligado. Essa unidade tem grande capacidade de armazenamento, mas tem velocidade de acesso significativamente menor em relação às memórias semicondutoras.

O módulo de Entrada e Saída, ou Input/Output (I/O), estabelece a ligação do computador com o mundo externo, usando equipamentos periféricos. Essa interface permite a ligação de teclados e *mouses* (para entrada de dados), monitores ou impressoras (para exibição dos dados) e placas de comunicação (para trocar dados a longa distância). Essa interface é significativamente mais lenta que os demais componentes do computador devido à natureza dos periféricos e à incapacidade humana de processar informações na velocidade dos computadores.

As próximas seções irão detalhar cada um dos módulos que constituem um computador.

4.2. Arquitetura Harvard

A Arquitetura de Harvard, mostrada na figura abaixo, é uma arquitetura de computador onde os caminhos de comunicação e a memória de armazenamento das instruções e de dados são fisicamente separados.

O termo originou-se da característica do computador eletromecânico Mark I, desenvolvida na Universidade de Harvard, que armazenava instruções em uma fita perfurada (com 24 bits de largura), e os dados eram armazenados em contadores eletromecânicos. Essa máquina tinha espaço limitado para o armazenamento de dados, inteiramente contido na Unidade Central de Processamento e, como não havia nenhuma ligação dessa memória com o armazenamento de instrução, o programa era carregamento e modificando de forma inteiramente isolada (offline[3]).

[3] Offline é a característica de todo processo em um computador que é realizado fora do equipamento e independente de o computador estar ligado ou não.

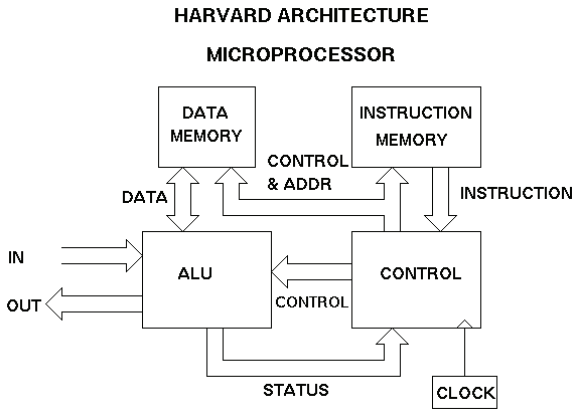


Figura 7 - Diagrama de blocos da Arquitetura Harvard

Na Arquitetura de Harvard, não há necessidade de fazer compartilhamento das memórias com dado e instrução. Em particular, a largura da palavra, o sincronismo, a tecnologia da execução e a estrutura do endereço de memória para dados e instruções podem diferir.

Em alguns sistemas, as instruções são armazenadas em memória somente de leitura (ROM) enquanto a memória de dados exige, geralmente, uma memória de leitura/escrita (RAM). Em alguns sistemas, há muito mais memória de instrução do que a memória dos dados devido ao fato de que os endereços de instrução são maiores do que endereços de dados.

Em um computador com a Arquitetura von Neumann, o processador central não pode ler uma instrução e realizar uma leitura/escrita de dados simultaneamente. Ambos não podem ocorrer ao mesmo tempo, porque as instruções e os dados usam o mesmo barramento de sistema. Em um computador com a arquitetura de Harvard, o processador central pode ler uma instrução e executar um acesso à memória de dados ao mesmo tempo porque os barramentos são distintos.

Um computador da Arquitetura de Harvard pode ser mais rápido para um determinado processador porque o acesso aos dados e instrução não usam o mesmo barramento. No entanto, uma Arquitetura de Harvard exige

um circuito eletrônico maior e mais complexo, comparado com a Arquitetura Von Neumann.

A Arquitetura de Harvard Modificada é muito semelhante à Arquitetura de Harvard, mas define um caminho alternativo entre a memória da instrução e o processador central, de forma que permita que as palavras na memória da instrução possam ser tratadas como dados somente de leitura, não apenas instrução. Isso permite que os dados constantes, particularmente tabelas de dados ou textos, sejam alcançados sem ter que, primeiramente, serem copiados para a memória dos dados, liberando, assim, mais memória de dados para variáveis de leitura/gravação. Para diferenciar leitura de dados da memória da instrução são utilizadas instruções especiais.

A vantagem principal da Arquitetura de Harvard - acesso simultâneo à memória de dados e instrução - foi compensada pelos sistemas modernos de cache, permitindo que uma máquina com Arquitetura von Neuman pudesse oferecer desempenho semelhante à Arquitetura de Harvard na maioria dos casos.

Apesar disso a Arquitetura de Harvard ainda é muito utilizada em aplicações específicas, onde a arquitetura ainda apresenta boa relação custo-desempenho, como:

- **Processadores de Sinal Digital (DSPs):** Processadores especializados em processamento de áudio, vídeo ou comunicações. Esses processadores precisam realizar uma grande quantidade de operações aritméticas em uma grande quantidade de dados, fazendo com que a Arquitetura Harvard seja economicamente viável.
- **Os microcontroladores**, pequenos processadores usados em aplicações específicas como controle de máquinas, veículos, equipamentos. Esses processadores são caracterizados pela pequena quantidade de memória de programa (geralmente memória Flash) e memória de dados (SRAM), assim, a Arquitetura de Harvard pode apressar o processamento permitindo o acesso à instrução e a dados simultaneamente. O barramento e o armazenamento separado possibilitam que as memórias de programa e de dados tenham larguras de bits diferentes. Por exemplo: alguns PICs têm uma palavra de dados de 8 bits, mas têm palavras de instrução de 12, 14, 16 ou 32 bits, conforme a complexidade do programa. Os produtos mais conhecidos que usam a Arquitetura Harvard são: processadores ARM (vários fabricantes), o PIC da Microchip Tecnologia, Inc., e o AVR da Atmel Corp.

4.3. Arquitetura Von Neumann

A Arquitetura de von Neumann é um modelo de arquitetura para computador digital de programa armazenado onde existe uma única estrutura compartilhada para armazenamento de instruções e de dados. O nome é uma homenagem ao matemático e cientista de computação John Von Neumann, que primeiro propôs essa idéia.

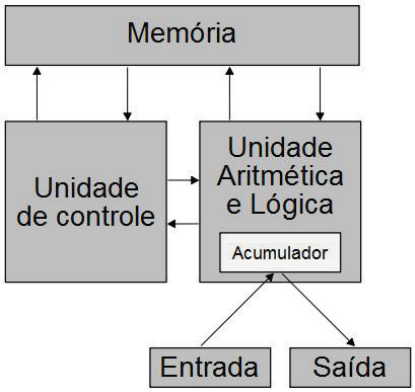


Figura 8 - Diagrama de Blocos de uma Arquitetura von Neumann

Um computador com Arquitetura von Neumann mantém suas instruções de programa e seus dados na memória de leitura/escrita (RAM). Essa arquitetura foi um avanço significativo sobre os computadores dos anos 40, tais como o Colossus e o ENIAC, que eram programados ajustando interruptores e introduzindo cabos para ligar os sinais de controle entre as várias unidades funcionais. Na maioria dos computadores modernos, usa-se a mesma memória para armazenar dados e instruções de programa.

Os termos "Arquitetura von Neumann" e "computador de Programa-Armazenado" são usados indistintamente neste texto. Ao contrário, a Arquitetura de Harvard armazena um programa em um local diferente do local utilizado para armazenar dados.

Von Neumann foi envolvido no projeto de Manhattan. Lá juntou-se ao grupo que desenvolveu o ENIAC para desenvolver o projeto do computador de programa armazenado chamado EDVAC. O termo "Arquitetura von Neumann" foi mencionada em um relatório sobre o projeto EDVAC, datado de 30 de junho de 1945, que incluiu idéias de Eckert e de Mauchly. O relatório foi lido por vários colegas de von Neumann na América e na Europa, influenciando significativamente todos os futuros projetos de computador. Apesar de ser reconhecida a contribuição de J. Presper Eckert, John Mauchly e Alan Turing para o conceito de computador de programa armazenado, essa arquitetura é mundialmente denominada Arquitetura Von Neumann.

4.4. Gargalos da Arquitetura Von Neumann

O compartilhamento de estrutura para armazenar dados e instruções entre o processador central e a memória conduz ao gargalo de Von Neumann devido à limitação da taxa de transferência dos dados do barramento entre o processador central e à memória. Na maioria de computadores modernos, a taxa de

transmissão do barramento geralmente é muito menor do que a taxa em que o processador central pode trabalhar. Isso limita seriamente a velocidade de processamento efetivo quando o processador central é exigido para executar o processamento de grandes quantidades de dados na memória.

O processador central é forçado continuamente a esperar a transferência de dados *de* ou *para* memória. A partir do aumento da velocidade dos processadores e do aumento do tamanho e da velocidade das memórias, o gargalo do barramento único se transformou em mais um problema. O termo "gargalo Von Neumann" foi criado por John Backus em artigo no ano de 1977 ACM Turing Award.

O problema de desempenho da Arquitetura Von Neumann pode ser reduzido através do uso de cache entre o processador central e a memória principal e pelo desenvolvimento de algoritmos de predição de desvio. Otimizações das linguagens orientadas a objeto modernas são menos afetadas que o código objeto linear gerado por compilador Fortran do passado.

Lei de Moore

A Lei de Moore descreve uma tendência histórica de longo prazo na evolução de desempenho de *hardware*, em que o número de transistores que podem ser colocados em um circuito integrado dobra a, aproximadamente ,cada dezoito meses. Mesmo não sendo uma "lei natural" que não possa ser controlada, a Lei de Moore é um bom prognóstico para o avanço da tecnologia microeletrônica nos últimos 40 anos.

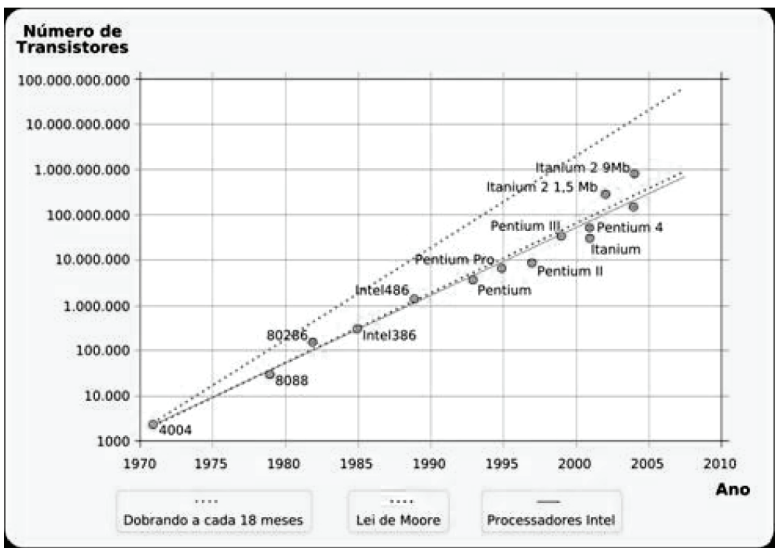


Figura 9 - Gráfico da Lei de Moore comparando com os processadores lançados [Intel]

As características de muitos dispositivos eletrônicos são fortemente associados à Lei de Moore: velocidade de processamento, capacidade de memória e o número e tamanho dos pixels nas câmeras digitais. Todos eles crescem a taxas aproximadamente exponenciais. Isso tem aumentado dramaticamente o uso da tecnologia eletrônica digital em quase todos os segmentos da economia mundial. A tendência acontece há quase meio século e não é esperado reduzir, pelo menos até 2015.

A lei recebeu o nome do co-fundador da Intel, Gordon E. Moore, que introduziu o conceito em um artigo de 1965. Ela tem sido usada desde então pela indústria de semicondutores para guiar o planejamento de longo prazo e para ajustar metas para a pesquisa e para o desenvolvimento.

Limites da Lei de Moore

Em 13 de abril de 2005, Gordon Moore anunciou, em uma entrevista, que a lei não pode se sustentar indefinidamente: "Não pode continuar para sempre. A natureza das taxas exponenciais é que você os empurra para fora e, eventualmente, disastres acontecem."

No entanto, ele previu que os transistores alcançariam os limites atômicos de miniaturização: "Nos termos de tamanho [dos transistores], você pode ver que nós estamos nos aproximando do tamanho dos átomos, que é uma barreira fundamental, mas serão duas ou três gerações antes de alcançarmos e que parece tão distante, que talvez não chegaremos a vê-lo. Nós ainda temos de 10 a 20 anos antes que alcancemos o limite fundamental. Até lá poderemos fazer chips maiores e quantidade de transistores da ordem de bilhões".

Em 1995, o microprocessador Digital Alpha 21164 tinha 9.3 milhão de transistores. Seis anos mais tarde, um microprocessador avançado tinha mais de 40 milhões de transistores. Teoricamente, com miniaturização adicional, em 2015 esses processadores devem ter mais de 15 bilhões de transistores, e, por volta de 2020, chegará na escala de produção molecular, onde cada molécula pode ser individualmente posicionada.

Em 2003 a Intel previu que o fim viria entre 2013 e 2018 com processo de manufatura de 16 nanômetros e portas com 5 nanômetros, devido ao tunelamento quântico do silício. No entanto, a melhoria da qualidade do processo de fabricação poderia aumentar o tamanho dos chips, e aumentar a quantidade de camadas pode manter o crescimento na taxa da Lei de Moore por, pelo menos, uma década a mais.

Síntese do capítulo

Neste capítulo, apresentamos os conceitos básicos de Arquiteturas de Computadores. Inicialmente apresentamos algumas definições necessárias para o entendimento de uma arquitetura de computadores que será útil em todas as unidades deste livro. Depois apresentamos uma pequena história dos computadores eletrônicos, desde a sua invenção nos anos 40 até os dias de hoje.

Apresentamos os componentes gerais de uma arquitetura de computadores que serão estudados nas próximas unidades. Finalmente, apresentamos as Arquiteturas Harvard e Von Neumann além de uma discussão sobre a Lei de Moore.

Atividades de avaliação

1. Observando a Figura 8 (arquitetura Von Neumann), avalie os componentes que podem limitar o desempenho. Proponha soluções para resolver esse gargalo
2. Qual módulo de uma arquitetura de computadores deve ser mais rápido para melhorar o desempenho de um sistema de processamento numérico (muitas operações matemáticas)?
3. Qual módulo de uma arquitetura de computadores deve ser mais rápido para melhorar o desempenho de um sistema de armazenamento de dados (muitos dados guardados e consultados)?
4. Pesquise na internet algumas tecnologias que estão sendo desenvolvidas para superar as limitações da Lei de Moore.

Leituras, filmes e sites

Leituras

VERAS, Manoel. **Virtualização:** Componente Central do Datacenter. Rio de Janeiro: Brasport, 2011.

NULL, Linda e LOBUR. **Princípios Básicos de Arquitetura e Organização de Computadores**. Porto Alegre: Bookman Editora, 2011.

HENNESSY, Jonh e PATTERSON, David. **Organização e Projeto de Computadores:** Uma abordagem quantitativa. Rio de Janeiro: Elsevier Editora LTDA, 2014.

Filmes

ENIAC O Primeiro Computador Eletrônico de propósito geral. Duração: 08min 02s. John Mauchley montou uma proposta solicitando ao exército financiamento para a construção de um computador eletrônico. A proposta foi aceita em 1943, e Mauchley e seu aluno de pós-graduação, J.Presper Eckert, passaram a construir um computador eletrônico ao qual deram o nome de ENIAC (Eletronic Numerical Integrator and Compute). O ENIAC consistia em 18 mil válvulas e 1500 relés, pesava 30 toneladas e consumia 140 quilowatts de energia. Em termos de arquitetura, a máquina tinha 20 registradores, cada um com capacidade para conter um número decimal de 10 algarismos. O ENIAC era programado com o ajuste de até 6 000 interruptores. A construção da máquina só foi concluída em 1946. Acesso em: https://www.youtube.com/watch?v=gZrMsjJPD6E

Arquitectura Harvard. Duração: 05min 20s. Este vídeo explica um pouco do que é a arquitetura de Harvard, que está relacionada com os microprocessadores e estrutura interna do circuito integrado. Acesso em: https://www.youtube.com/watch?v=eri7uBpCxL0

Arquitectura de Neumann. Duração: 14min 53s. Video em espanhol que explica os elementos da arquitetura proposta por Von Neumann. Acesso em: https://www.youtube.com/watch?v=QscWZ_rven4

Sites

Museu da História do Computador (em inglês)

http://www.computerhistory.org/

História do Computador (em inglês)

http://www.computersciencelab.com/ComputerHistory/History.htm

Discussão sobre e Lei de Moore (em inglês)

http://www.intel.com/technology/mooreslaw/

Referências

VELLOSO, F.C. **Informática:** Conceitos Básicos. 7ª Ed. Editora: Campus, 2004.

PARHAMI, B. **Arquitetura de Computadores**. 1ª Ed. Editora: McGraw-Hill, 2008.

Parte 2

Circuitos lógicos digitais

Capítulo 2

Portas Lógicas Básicas

Objetivos

Neste capítulo vamos apresentar os conceitos de Lógica Digital necessários para entendimento das Unidades seguintes. Iniciamos com a apresentação das unidades lógicas básicas, que possibilitam a construções de todos os circuitos lógicos. Em seguida apresentamos os circuitos combinacionais mais tradicionais em uma arquitetura e concluimos com a apresentação dos circuitos sequenciais, que acrescentam o conceito de armazenamento da informação.

Introdução

Os blocos básicos de uma arquitetura de computador são as *portas lógicas* ou apenas *portas*. As portas são os circuitos básicos que têm, pelo menos, uma (geralmente mais) entrada e exatamente uma *saída*. Os valores da entrada e da saída são os valores lógicos verdadeiro ou falso. Na convenção de computação é comum usar 0 para falso e 1 para verdadeiro.

As portas não têm nenhuma *memória*. O valor da saída depende somente do valor atual das entradas. Esse fato torna possível usar uma *tabela de verdade* para descrever inteiramente o comportamento de uma porta. Para representar matematicamente a lógica digital, utilizamos a Álgebra de Boole[4].

[4] Álgebra de Boole foi criada pelo matemático George Boole no século XIX com objetivo de definir uma série de operações lógicas aplicadas a sistemas binários (0s e 1s). Essa álgebra permite realizar inferências lógicas em computadores binários.

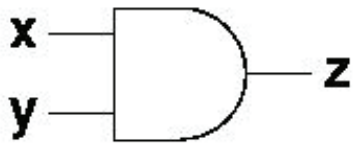
1. Portas básicas

Portas básicas são portas fundamentais que podem formar outras portas com funções especiais. Nós consideramos geralmente três tipos básicos das portas: as portas E, as portas OU e as portas NÃO (ou *inversores*). Também é usual chamar as portas pelo termo em inglês, respectivamente, AND, OR e NOT.

1.1. Porta E (AND)

Uma porta E pode ter um número arbitrário de entradas. O valor da saída é 1 se e somente se todas as entradas são 1. Se não, o valor da saída é 0. O nome foi escolhido porque a saída é 1 se e somente se a primeira entrada e a segunda entrada, e,..., e a *n-ésima* entrada são 1.

É frequentemente útil desenhar diagramas das portas e das suas interconexões. Em tais diagramas, a porta E é desenhada como:



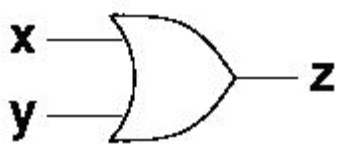
A tabela de verdade para uma porta E com duas entradas é mostrada abaixo:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

1.2. Porta OU (OR)

Como a porta E, a porta OU pode ter um número arbitrário de entradas. O valor da saída é 1 se e somente se pelo menos de um dos valores da entrada é 1. Caso contrário, a saída é 0. Ou seja, o valor da saída é 0 somente se todas as entradas são 0. O nome foi escolhido porque a saída é 1 se e somente se a primeira entrada ou a segunda entrada, *ou,..., ou a n-ésima* entrada são 1.

Em esquemas de circuito, desenha-se a porta OU como:



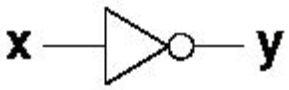
A tabela de verdade para uma porta *OU* com duas entradas é mostrada abaixo:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

1.3. Porta NÃO (NOT)

A porta NÃO também é conhecida como inversor e tem exatamente uma entrada e uma saída. O valor da saída é 1 se e somente se a entrada é 0. Se não, a saída é 0, ou seja, o valor da saída é exatamente o oposto do valor da entrada.

Em esquemas de circuito, nós desenhamos a porta NÃO como:



A tabela de verdade para um inversor é mostrado abaixo:

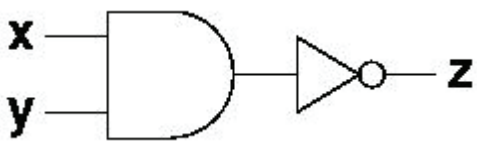
X	y
0	1
1	0

2. Portas compostas

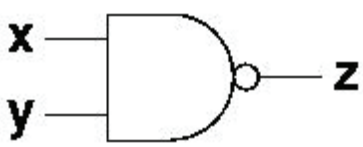
Às vezes, é prático combinar funções das portas básicas em portas mais complexas a fim reduzir o espaço em diagramas de circuito. Nesta seção, nós mostraremos algumas portas compostas junto com suas tabelas de verdade.

2.1. Porta NÃO-E (NAND)

A porta NÃO-E é uma porta E com um inversor na saída. Assim, em vez de desenhar diversas portas com esta,



desenha-se uma única porta-E com um pequeno anel na saída, como este:



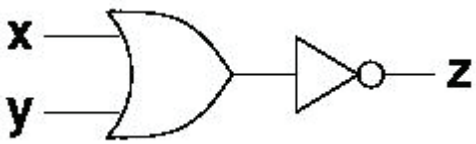
A *porta NÃO-E*, assim como a porta-E, pode ter um número arbitrário de entradas.

A tabela de verdade para a *porta NÃO-E* é semelhante à tabela para porta-E, exceto que todos os valores da saída são invertidos:

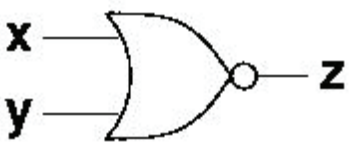
x	y	Z
0	0	1
0	1	1
1	0	1
1	1	0

2.2. Porta NÃO-OU (NOR)

A porta NÃO-OU é uma porta OU com um inversor na saída. Assim, em vez de desenhar diversas portas como esta,



desenha-se uma única porta OU com um pequeno anel na saída como este:



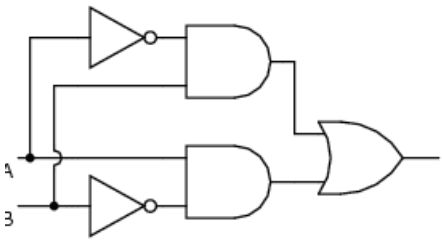
A porta NÃO-OU, assim como a porta OU pode ter um número arbitrário de entradas.

A tabela de verdade para a porta NÃO-OU é semelhante à tabela para porta OU, exceto que todos os valores da saída são invertidos:

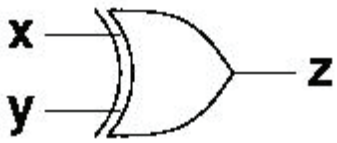
x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

2.3. Porta OU-EXCLUSIVO (XOR)

A porta OU-EXCLUSIVO é similar a uma porta OU. Apesar de poder ter um número arbitrário de entradas, normalmente representamos apenas duas. O valor da saída é 1 se somente uma das entradas é 1, e a outra é 0, por isso, se diz OU-EXCLUSIVO. Se não, a saída é 0. Assim, em vez de desenhar diversas portas como esta,



desenha-se uma porta OU-EXCLUSIVO como mostrada abaixo:



A tabela de verdade para uma porta *ou-exclusivo* com duas entradas é mostrada abaixo:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Mas para que serve uma porta OU-EXCLUSIVO? Ela é um bloco essencial para a construção de um circuito muito útil em computadores, o SOMADOR.

Para refletir

1. Analise as tabelas verdade e veja como poderá ser construído um circuito que implemente uma função OU a partir de função E (dica: De Morgan).
2. Projete um circuito com lógica digital que indique quando uma caixa d'gua está cheia, quando está vazia e quando os sensores estão com defeito (caixa cheia e não vazia).
3. Pesquise sobre Álgebra de Boole e veja como podemos representar funções lógicas de maneira textual.

3. Circuitos Combinacionais

Um **circuito combinacional** é um circuito construído com portas de lógica digital básicas que implementam funções mais complexas. A principal característica de um circuito combinacional é que a sua saída depende unicamente de suas entradas. Assim, ao se colocar um conjunto determinado de entradas, sempre obteremos a mesma saída. Os circuitos combinacionais não possuem nenhum tipo de memória.

Existe uma grande variedade de tipos de circuitos combinacionais. Apresentaremos aqui apenas o multiplexador, o decodificador, a matriz lógica programável e os somadores.

3.1. Multiplexadores

Um multiplexador, ou simplesmente **mux**, é um dispositivo que seleciona dados procedentes de várias entradas para uma única saída. São utilizados em situações onde é necessário escolher apenas uma entrada e isolar as demais.

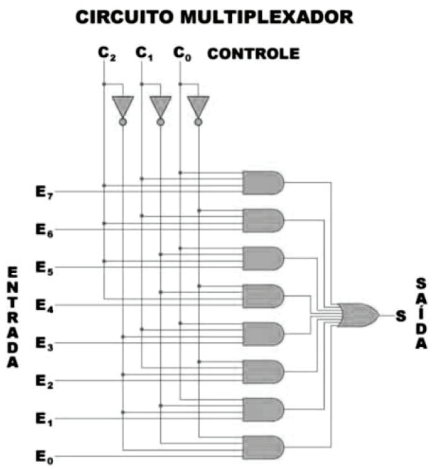


Figura 10 - Diagrama de um Multiplexador de 8 entradas e 1 saída com 3 bits de controle

Em circuitos digitais, o multiplexador é um dispositivo que possui várias entradas de dados e somente uma saída. A partir de um circuito de controle no qual a entrada é o valor da entrada desejada, é possível escolher a porta de entrada que se deseja passar para a saída. Por exemplo: um multiplexador de duas entradas é uma simples conexão de portas lógicas cuja saída S é a entrada A ou a entrada B, dependendo do valor de uma entrada C que seleciona a entrada.

O circuito a seguir mostra um multiplexador com 8 entradas (E0 a E7) e 1 saída (S). A seleção da entrada é realizada pela entrada de controle com 3 entradas (C0 a C2), que deve indicar o número binário da entrada que se quer selecionar. Se, por exemplo, colocarmos o valor 111 nas entradas C0 a C3, vamos habilitar a entrada E7 (as entradas negadas entram na porta E onde a entrada E7 está ligada). Assim, o valor da entrada E& (seja ele 0 ou 1) é copiado para a porta S.

Um demultiplexador, ou simplesmente **demux**, é um dispositivo que executa a função inversa do multiplexador. Um circuito demultiplexador é construído a partir de um decodificador, pois, a partir de um código de controle, podemos escolher uma saída.

3.2. Decodificadores

Um decodificador, ou decodificador de endereços, é um circuito que possui dois ou mais bits de controle como entrada (por exemplo, um barramento de endereço) e um ou mais dispositivos de saída que poderá ser selecionado. Quando colocamos um dado nas entradas de controle C0, C1 e C2, uma das saídas S0 a S7, e apenas uma, é selecionada. Um decodificador com N bits de controle de entrada pode selecionar até 2N dispositivos individuais de saída.

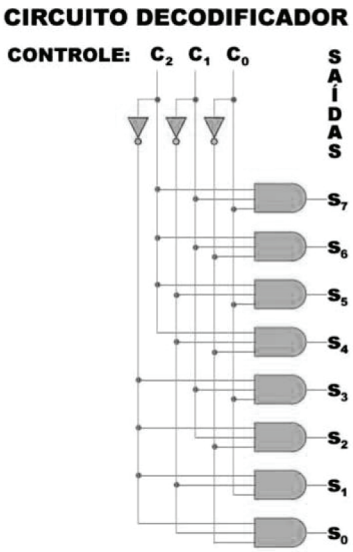


Figura 11 - Diagrama de um decodificador de 8 saídas com 3 bits de controle

Um decodificador também pode ser chamado de demultiplexador ou demux quando associamos uma entrada comum a cada porta E. Quando o circuito de controle ativar uma porta E, ela vai copiar o dado de entrada para uma determinada saída, fazendo a função inversa de um multiplexador.

Matriz Lógica Programável

Uma matriz lógica programável, também conhecido por PAL (Programable Array Lógic) ou PLD (*Programmable logic device*) é um dispositivo eletrônico de lógica digital programável que possibilita implementar uma grande variedade de circuitos combinacionais.

Entretanto, ao contrário de uma porta lógica fixa, que tem uma função determinada, um dispositivo de matrz lógica programável tem uma função indefinida no momento de sua fabricação. O circuito deve ser programado para executar a função desejada. Existe uma gama enorme de fabricantes de dispositivos programáveis e de categorias de dispositivos, cada um com uma aplicação específica. Alguns dispositivos são programáveis apenas uma vez, e outros podem ser programados várias vezes. Alguns dispositivos reprogramáveis perdem sua programação quando desligados, e outros mantêm a gravação mesmo quando a energia é desligada.

Qualquer função booleana pode ser construída a partir da combinação de portas E e portas OU. O circuito mostrado na figura a seguir mostra um exemplo de um dispositivo de matriz de lógica programável. As entradas A e B são sinais de entrada do circuito. A primeira função implementada é gerar o

sinal próprio e a negação de cada sinal. Uma matriz faz todas as combinações possíveis na matriz de portas E. Como temos duas entradas A e B, teremos 4 combinações possíveis de função E. Essa etapa gera todos os **produtos** dos sinais de entrada.

A etapa seguinte é uma matriz de portas OU, que permite a combinação de todas as entradas E. Nesse caso temos um fusível, queimado no processo de gravação, que vai definir quais funções lógicas serão implementadas para cada saída. Essa etapa gera as **somas** dos produtos desejados. Com essa organização, podemos implementar qualquer função de lógica binária.

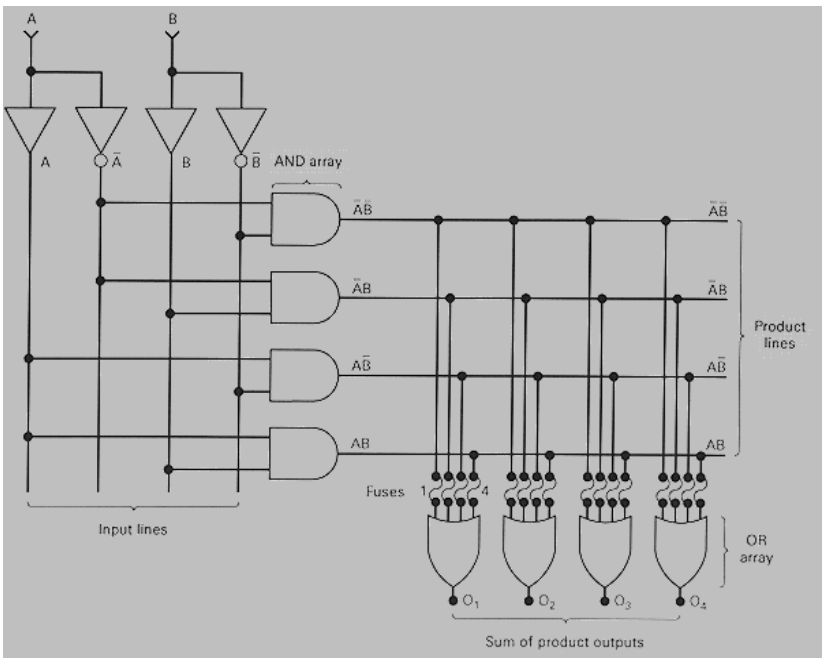


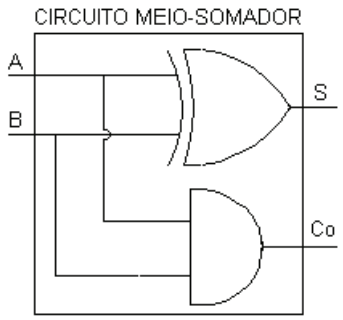
Figura 12 - Diagrama de uma Matriz Lógica Programável com 2 entradas e 4 saídas

Somador

O somador binário é um circuito que realiza a operação de soma de subtração em complemento de dois, de um bit. Essencialmente o somador é constituído por um OU-EXCLUSIVO que calcula o resultado e por um E que calcula o "vai um". O princípio é mostrado no meio somador, mas um circuito real é o somador completo, ambos mostrados nas próximas seções.

Meio Somador

O circuito somador realiza a soma de um bit, tendo como resultado a valor da soma e o bit de vai um conforme diagrama mostrado abaixo:



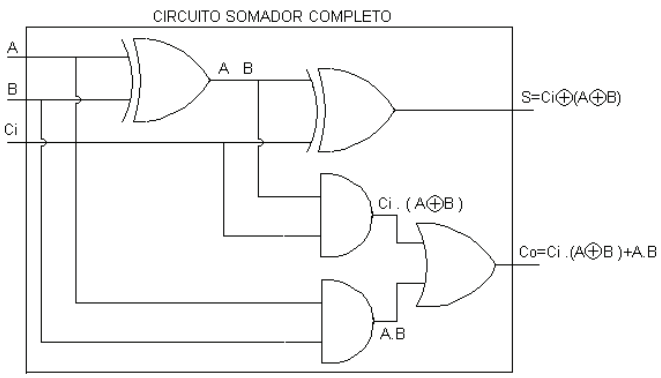
A tabela de verdade de um meio-somador é mostrado abaixo:

X	Y	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Somador Completo

Apesar do meio somador informar o bit de vai um, ele não considera o vai um do bit anterior, assim, ele só pode ser usado para somar o bit menos significativo.

Para realizar uma soma completa, precisamos utilizar o circuito somador completo que, tanto informa o vai um para o próximo bit, como considera o vai um do bit anterior da soma. O diagrama do somador completo é mostrado abaixo:



A tabela de verdade de um somador-completo é mostrado abaixo:

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Para refletir

1. Imagine como um circuito decodificador poderia exibir um dígito utilizando um display de 7 segmentos.
2. Pesquise sobre os diversos tipos de matriz lógica programável e exemplifique aplicações para cada um.
3. Monte um circuito somador de 4 bits unindo 4 modulo de somador completo. E se usar meio somadores?

4. Circuitos Sequenciais

Mostramos nas seções anteriores que a saída de um circuito combinacional depende unicamente de suas entradas. O motivo é que os circuitos combinacionais não têm nenhuma memória. Para construir circuitos de lógica digitais mais sofisticados, incluindo computadores, nós precisamos de um dispositivo mais poderoso. Esse dispositivo, chamado Flip-flop, é um circuito cuja saída depende, além das suas entradas no circuito, do seu estado precedente. Ou seja, é um circuito que tem a memória do estado passado.

Para que um dispositivo sirva como uma memória, deve ter três características:

- o dispositivo deve ter dois estados estáveis (tratamos de dados binários)
- deve haver uma maneira de ler o estado do dispositivo
- deve haver uma maneira de atribuir, pelo menos uma vez, o seu estado.

É possível produzir circuitos digitais com memória usando portas lógicas que já foram vistas. Para fazer isso, nós precisamos introduzir o conceito de realimentação (*feedback*).

Até agora, o fluxo lógico nos circuitos que estudamos foi da entrada à saída, ou um circuito "acíclico". Agora nós introduziremos um circuito em que a saída realimenta a sua entrada, permitindo realizar a manutenção de um estado, mesmo quando a informação na entrada cessa.

4.1. Flip-Flops

Flip-Flop SR

O Flip-flop SR é um tipo simples de circuito sequencial, isto é, o estado da entrada depende da saída anterior. A figura abaixo mostra o diagrama esquemático de um flip-flop SR e o símbolo usualmente utilizado. O símbolo para o flip-flop ou latch S-R é mostrado na figura abaixo.

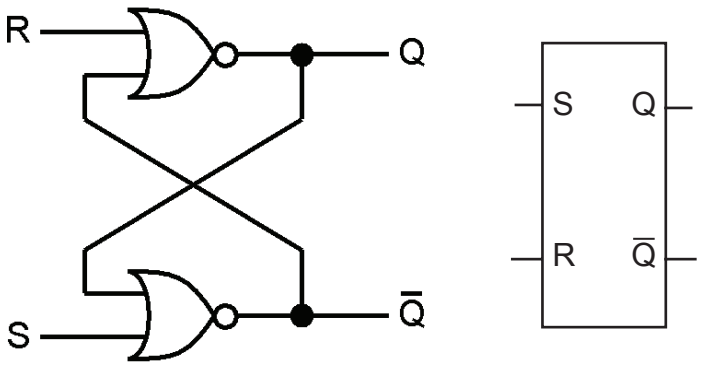


Figura 13 - Diagrama e Símbolo de um Flip-Flop SR

A saída da porta NÃO-OU é verdadeira somente quando ambas as entradas são falsas. A saída de cada uma das portas NÃO-OU é realimentada de volta à entrada da outra porta. Isso significa que, se a saída de uma porta NÃO-OU é verdadeira, a saída da outra deve ser falsa. Se a saída Q da parte superior da porta NÃO-OU é verdadeira ou um, isso significa que uma das entradas da porta $\overline{Q}$ do NÃO-OU de baixo, é verdadeira, e a saída $\overline{Q}$ do NÃO-OU de baixo deve ser falsa.

Para que a saída do NÃO-OU superior ser verdadeira, ambas as suas entradas têm que ser falsas.

Ao acionar a entrada S, a saída $\overline{Q}$ da porta NÃO-OU de baixo torna-se falso, e a saída Q da porta NÃO-OU superior, é forçada a verdadeiro. Ao acionar S outra vez e desligá-lo em seguida, a saída do circuito permanece inalterada. O que aconteceu é que nós armazenamos o valor do S no circuito que permanece S mesmo que altere a entrada. Os valores de saída Q e $\overline{Q}$ são sempre opostos.

Desligando S e acionando R e desligando-o em seguida, qualquer que seja o valor armazenado fará com que a saída Q da porta NÃO-OU superior, é forçada a falso, e saída $\overline{Q}$ da porta NÃO-OU de baixo torna-se verdadeiro. Daí podemos concluir que a entrada S verdadeira torna a saída Q verdadeira, e a entrada R verdadeira torna a saída Q falsa.

Esse circuito é um *latch* S-R (uma trava). As entradas S e R ajustam a trava, fazendo verdadeiro e falso. A saída do circuito é estável em um ou outro estado com as entradas removidas. Nós podemos remover a entrada que causou uma saída particular, e a saída permanecerá inalterada. O estado e a saída mudará somente quando a entrada complementar é aplicada. O circuito tem dois estados estáveis, por isso chamamos de circuito biestável.

A entrada S=R=1 não é permitida. Se ambas as entradas são verdadeiras, ambas as saídas devem ser falsas. Isso implica saída $Q = \overline{Q} = 0$, que é logicamente incompatível. Nessa situação, o circuito é instável e, quando uma das entradas retorna ao estado falso, a entrada restante determina o estado estável e as mudanças da saída.

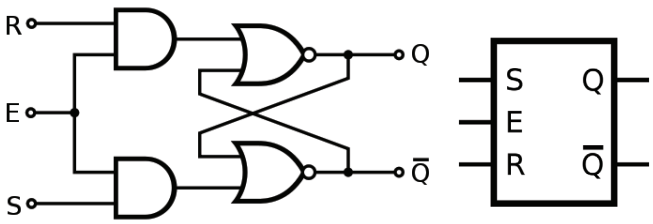


Figura 14 - Flip-flop SR com Habilitador (relógio)

A figura acima mostra um flip-flop SR com habilitador que acrescenta um par de portas lógicas E com uma entrada comum E (*Enable*). Nesse caso, quando a entrada E está falsa (zero), a saída delas e a entrada das portas NÃO-OU permanecem em falso, não importando o valor de S ou de R. Quando a entrada E fica verdadeira, o valor das portas S e R são encaminhadas para as portas NÃO-OU. Isso garante que, apenas quando o habilitador estiver ativo, é que poderemos acionar as portas R e S.

Podemos construir um circuito de flip-flop SR com portas NÃO-E como mostrado abaixo. O funcionamento é muito semelhante ao circuito com porta NÃO-OU. Sua grande vantagem é que industrialmente é mais fácil construir uma porta NÃO-E do que uma porta NÃO-OU.

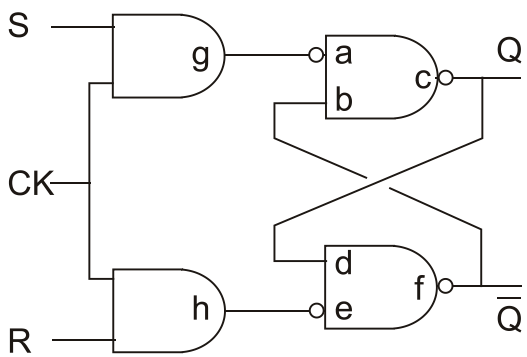


Figura 15 - Flip-flop RS NÃO-E com relógio

Podemos cascatear dois circuitos flip-flop consecutivos, possibilitando o isolamento do sinal de entrada e de saída. O flip-flop com duas seções, também conhecido como flip-flop mestre-escravo (*master-slave*), tem esse nome porque o registro de entrada é operada pela seção mestra (*master*), enquanto a seção de saída escrava (*slave*) é ativada apenas durante a segunda metade de cada ciclo de pulso de disparo do relógio.

O flip-flop mestre-escravo RS NÃO-E é mostrado abaixo:

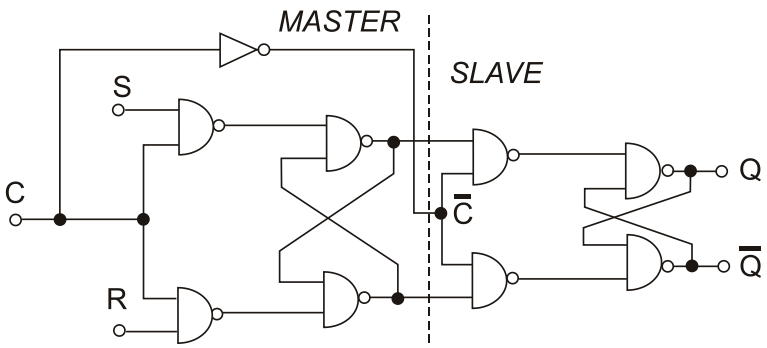


Figura 16 - Flip-flop RS mestre-escravo

O flip-flop RS mestre-escravo consiste realmente em dois circuitos idênticos de flip-flop RS, como mostrado acima. Entretanto, um inversor conectando as duas entradas de relógio (C) assegura-se de que as duas seções estejam habilitadas durante metade de ciclos com sinal oposto. Como é impossível essas duas portas de relógio sejam ativadas no mesmo instante, evitamos que o valor passe diretamente para o estágio seguinte.

Se começamos com a entrada C = 0, as entradas S e R estão desconectadas do flip-flop mestre de entrada. Consequentemente, nenhuma mudança nos sinais de entrada irá afetar o estado das saídas finais.

Quando o sinal de C vai à lógica 1, as entradas de S e de R podem controlar o estado do flip-flop de entrada, apenas como com circuito de flip-flop RS já examinado. Entretanto, ao mesmo tempo o sinal invertido de C é aplicado ou flip-flop escravo (*slave*) da saída impedindo que o estado do flip-flop de entrada tenha efeito no segundo estágio. Consequentemente, todas as mudanças de sinais de entrada nas portas R e de S são seguidas pela trava da entrada quando C estiver na lógica 1, mas não refletindo nas saídas de Q e $\overline{Q}$.

Quando C cai outra vez à lógica 0, as entradas de S e de R estão isoladas outra vez da entrada. Ao mesmo tempo, o sinal invertido de C permite agora que o estado atual da trava da entrada alcance a trava da saída. Consequentemente, as saídas de Q e $\overline{Q}$ podem somente mudar o estado quando o sinal de C cai de uma lógica 1 à lógica 0.

[5] Condição de corrida é uma falha em sistema eletrônico ou programa de computador em que o resultado do processo é inesperadamente dependente da sequência ou sincronia de outros eventos.

Há, ainda, um problema não resolvido com a organização mestre-escravo, a condição de corrida[5] possível de ocorrer quando as entradas de S e R estão na lógica 1 e C cair de 1 para 0. No exemplo acima, nós supomos automaticamente, que a raça terminará sempre com a trava mestra na lógica 1 estado, mas não será certo que está com componentes reais. Consequentemente, precisamos ter uma maneira de impedir que essas condições ocorram. Uma solução é adicionar algum *feedback* adicional do setor *escravo* para o *mestre*. O circuito resultante é chamado um flip-flop JK.

Flip-Flop tipo JK

Para impedir a possibilidade de uma condição indefinida que ocorre quando as entradas de S e de R estão em nível 1 quando a entrada de relógio habilita a porta E de um flip-flop SR, nós devemos garantir que as portas não deixarão ocorrer essa situação. Ao mesmo tempo, nós ainda queremos que o flip-flop deve poder mudar o estado em cada transição da entrada de relógio CLK, se os sinais da lógica da entrada JK estiverem em 1.

Se a saída de Q está em 1 (o flip-flop está no estado "Set"), a entrada S não pode fazê-lo ajustar-se ao que ele já é. Consequentemente, nós podemos incapacitar a entrada S sem incapacitar o flip-flop. Na mesma maneira, se a saída de Q está em 0 (o flip-flop está no estado "Reset"), a entrada de R pode ser desabilitada sem causar nenhum dano.

O circuito abaixo mostra a solução. Ao flip-flop RS nós adicionamos duas conexões novas das saídas de Q e de Q de volta às portas de entrada originais. Lembre-se que uma porta de NAND pode ter qualquer número de entradas. Para mostrar que nós fizemos uma alteração no funcionamento do circuito, nós mudamos as designações das portas de entradas e do flip-flop próprio. As entradas são designadas agora J (em vez de S) e K (em vez de R). O circuito inteiro é conhecido como um flip-flop JK.

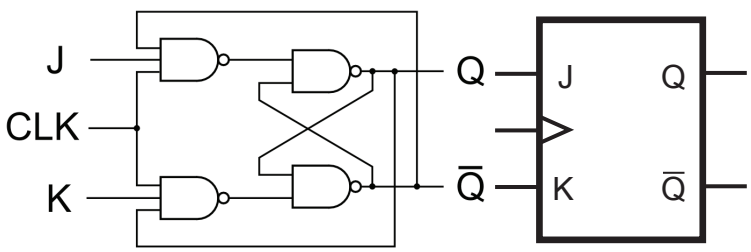


Figura 17 - Flip-flop tipo JK

Na maioria das vezes, o flip-flop JK comporta-se da mesma forma que um flip-flop RS. As saídas de Q e de Q mudarão de estado somente quando houver queda no sinal CLK, e as entradas J e K controlarão o estado das saídas.

Mas há uma diferença. Como uma das duas entradas da lógica é sempre desabilitada de acordo com o estado de saída do flip-flop, o latch mestre não pode mudar o estado enquanto a entrada de CLK estiver em 1. Em lugar disso, a entrada de habilitação pode mudar o estado do latch mestre apenas uma vez, depois da qual ele não mudará outra vez. Isso não acontecia com o flip-flop RS.

Se as entradas de J e de K estiverem ambas em nível 1, e o sinal de CLK continua a mudar, as saídas de Q e de Q mudarão simplesmente o estado com cada borda de queda do sinal de CLK (o circuito mestre do latch mudará o estado com cada transição da borda de CLK). Nós podemos usar essa característica para garantir a estabilidade do flip-flop. Para vários tipos, esta é a única maneira de assegurar-se de que o flip-flop mude o estado somente uma vez em cada impulso de relógio.

Flip-Flop tipo D

Uma vez que nós aplicamos a ideia de usar um relógio no nosso latch SR, nós podemos começar nos livrando do problema de fazer S = R = 1 e simplificar a entrada a nosso circuito.

Geralmente, o que nós queremos fazer com um dispositivo de armazenamento é guardar um bit de informação. A necessidade para explicitamente ajustar e restaurar é uma complexidade adicional. O que nós gostaríamos é um circuito que tenha uma única entrada de dados D e uma única saída de dados Q. Quando o sinal de pulso de disparo é elevado, o que aparece em D deve ser o valor armazenado em Q.

O circuito de figura abaixo é tal circuito. Tem uma entrada de dados D e uma entrada de controle C usado apara relógio. A entrada de dados é conectada através do E da porta à entrada de S de um latch SR. É conectada igualmente através de um inversor e o E da porta à entrada de R. As portas são conetadas à entrada de C do circuito. Se C é falso, nenhum sinal alcança a trava e seu estado permanece inalterado. Se C é verdadeiro, e D é verdadeiro, a entrada de S é verdadeira, e o valor armazenado tem um valor de verdade que é igual ao D. Se C é verdadeiro, e D é falso, a entrada de R da trava está conduzida através do inversor e um valor de falso, que é igual a D, é armazenado.

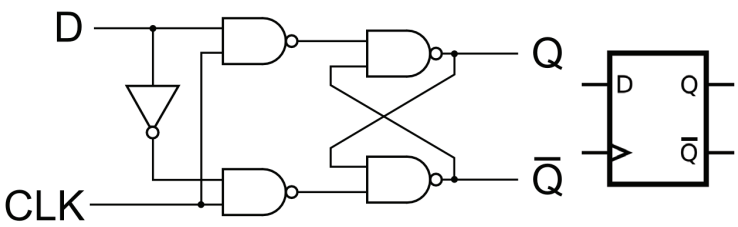


Figura 18 - Flip-flop tipo D

O conceito de um flip-flop D, onde o bit a ser armazenado está aplicado à entrada de S de um flip-flop SR, e através de um inversor à entrada de R, pode ser feito para trabalhar somente quando o CLK for habilitado.

O flip-flop D armazena o que está na porta D quando CLK é verdadeiro. Se CLK está verdadeiro, somente depois que os circuitos de entrada se estabilizem, esse circuito armazenará o valor correto do D, porque há somente uma entrada de dados D. Nesse caso uma situação indefinida como, por exemplo, S=R=1 não pode ocorrer nunca.

Enquanto a entrada CLK é verdadeira, as mudanças a D estão refletidas na saída do circuito. O valor D é então copiada para o circuito do latch, mantendo o dispositivo estável.

Um ponto essencial sobre o flip-flop de D é que, quando a entrada de pulso de relógio cai a 0 as suas, saídas podem mudar o estado. A saída Q exibe sempre o estado da entrada D no momento do pulso de disparo. Isso não acontecia nos flip-flops RS e JK. A seção mestra do RS mudaria repetidamente o estado para estabilizar os sinais de saída quando a linha do pulso de disparo do relógio for a 1, e a saída de Q refletiria qualquer entrada que recebeu recentemente. A seção mestra do JK receberia e guardaria uma entrada até a mudança do estado, e nunca mudaria esse estado até que o ciclo seguinte do pulso de relógio ocorra. Esse comportamento não é possível com o flip-flop tipo D.

4.2. Registrador de deslocamento

O Registrador de Deslocamento é um circuito para realizar a conversão de dados seriais em dados paralelos, assim como a conversão de dados paralelos para serial\is. A figura a seguir mostra um circuito de registrador de deslocamento serial-paralelo.

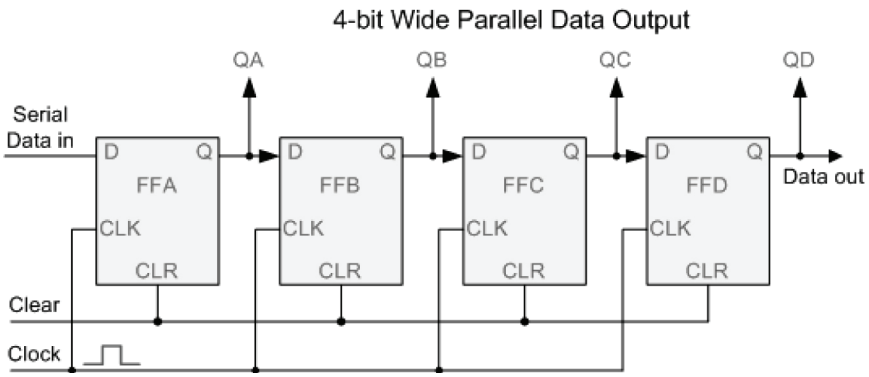


Figura 19 - Registrador de deslocamento serial-paralelo

O circuito é construído com flip-flops D ligados em série, isto é, a saída Q de um flip-flop é ligada à entrada D do flip-flop seguinte. O sinal de relógio é aplicado a todos os flip-flops de forma que se possa garantir a sincronização do circuito. O sinal serializado é introduzido no circuito através da porta D do primeiro flip-flop. A cada ciclo do relógio, o dado da porta D é transferido para a porta Q. No próximo ciclo, esse dado é copiado para o flip-flop seguinte, e assim sucessivamente. Ao final de 4 ciclos, em que 4 é o tamanho da palavra desse exemplo, podemos ler os dados paralelizados através das portas QA a QD. O circuito pode, também, dispor de um sinal Clear para iniciar o registrador (limpar o conteúdo anterior).

Outra possibilidade de circuito Registrado de Deslocamento é realizar a função inversa, isto é, transformar dados paralelos em serial, mostrado na figura abaixo.

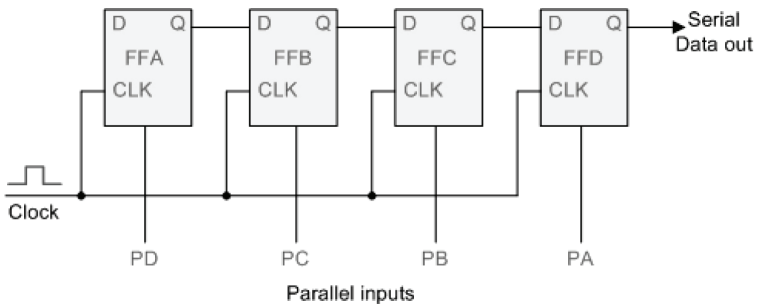


Figura 20 - Registrador de Deslocamento Paralelo-Serial

Agora nesse exemplo, o dado paralelo é aplicado as portas PA a PD. Observe que o bit mais significativo é aplicado na extremidade direita do circuito, pois este será o primeiro a ser transmitido, enquanto PD será o último. A cada ciclo de relógio, o dado da porta D é tranferido para a porta Q, que está ligada à porta D do flip-flop seguinte. Ao final de 4 ciclos, em que 4 é o tama-

nho da palavra desse exemplo, podemos ler os dados serializados através da porta Qdo último flip-flop.

4.3. Contadores

Uma exigência comum em circuitos digitais é realizar contagem, tanto para frente como para trás. Os relógios digitais pulsos estão em toda parte; os temporizadores são encontrados em uma grande variedade de equipamentos, como, fornos de microondas, máquinas de lavar, além dos vários dispositivos em automóveis.

Embora existam muitas variações de contador básico, são todos fundamentalmente muito semelhantes. Dividimos os contadores em dois grandes grupos: os contadores assíncronos e os contadores síncronos.

Contadores Assíncronos

Um contador assíncrono não precisa de um sinal de relógio comum para sincronizar o circuito. Basta aplicar um relógio na sua entrada, e, a cada pulso, será incrementada (ou decrementada) uma contagem.

A figura abaixo mostra o tipo o mais básico do circuito de contagem binário.

Esse contador é contruído com dois flip-flops JK, onde a saída Q do primeiro é ligada à porta de relógio do segundo flip-flop. O flip-flop JK apresenta um comportamento interessante: quando ambas as portas, J e K, estão em 1, na transição do relógio, a saída Q vai mudar de estado. Assim, quando ocorre uma transição na porta de relógio de 1 para 0, a saída Q vai trocar de nível. A saída Q do primeiro flip-flop está ligada na porta de relógio do segundo flip-flop. Agora, o segundo flip-flop somente vai trocar de estado quando a saída Q sofrer uma transição de 1 para 0. Com isso, o segundo flip-flop vai trocar de estado apenas a cada duas transições do primeiro flip-flop. Observe na figura do contador assíncrono o resultado da saída Q0 e Q1. Ela mostra números de 0 a 3 no formato binário, isto é, a cada transição no relógio, ele incrementa um número, exibindo uma sequência 0, 1, 2 e 3.

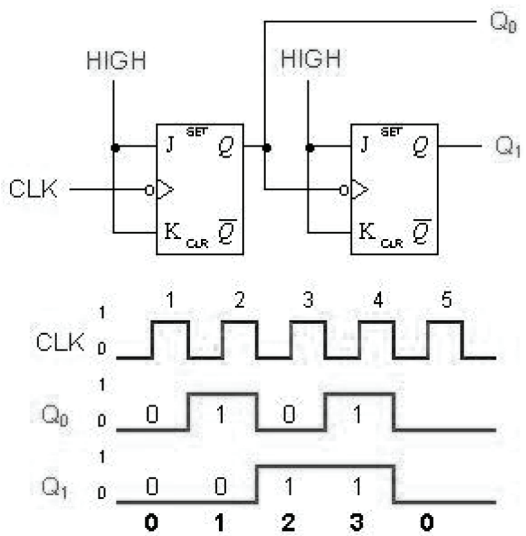


Figura 21 - Contador assíncrono de 2 bits

Ao adicionarmos mais flip-flops cascateando as portas de saída com a porta de relógio seguinte, podemos construir contadores de qualquer quantidade binária. Um circuito contador geralmente apresenta uma entrada para zerar o contador (reset) de forma que a contagem possa ser reiniciada.

Um problema grave com os contadores assíncronos é que os flip-flops individuais não mudam o estado ao mesmo tempo. Como cada flip-flop leva um tempo para mudar de estado, ao trabalharmos com um circuito de contador com vários dígitos, precisamos esperar um tempo grande para que os dados se estabilizem ao longo do circuito, mostrando valores errôneos até estabilizar. Quando o ciclo de contagem é baixo, isto é, quando os tempos de mudança de cada contagem forem grandes, e devido à incapacidade do olho humano de identificar a contagem rápida, esse fenômeno não causa grandes impactos. Porém, se a contagem for rápida, os erros tornam o circuito impossível de usar.

Para resolver esse problema, podemos usar um contador síncrono, isto é, todas as mudanças de estado ocorrem simultaneamente, cadenciadas por um relógio único.

Contadores Síncronos

Em nossa discussão inicial sobre contadores, falamos a necessidade de enviar o sinal relógio simultaneamente a todos os flip-flops de modo que os bits de contagem mudassem o estado ao mesmo tempo. Para realizar isso, precisamos aplicar o mesmo pulso de relógio em todos os flip-flops. Entretanto, não queremos que todos os flip-flops mudem de estado a cada pulso de relógio. Para isso, precisamos adicionar algumas portas de controle para determinar quando cada flip-flop deve mudar o estado e quando não deve.

Na figura a seguir, mostramos um contador síncrono de 3 dígitos construído com três flip-flops tipo JK. Podemos observar que o sinal de relógio é o mesmo para todos os flip-flops, então todos irão trocar de estado simultaneamente, evitando a Condição de Corrida, que pode ocorrer no contador assíncrono. O primeiro flip-flop JK tem ambas as entradas em 1, portanto a sua saída Q0 irá trocar de estado a cada ciclo de relógio. Outro ponto importante é a colocação de uma porta E no último flip-flop, cujo motivo será explicado a seguir.

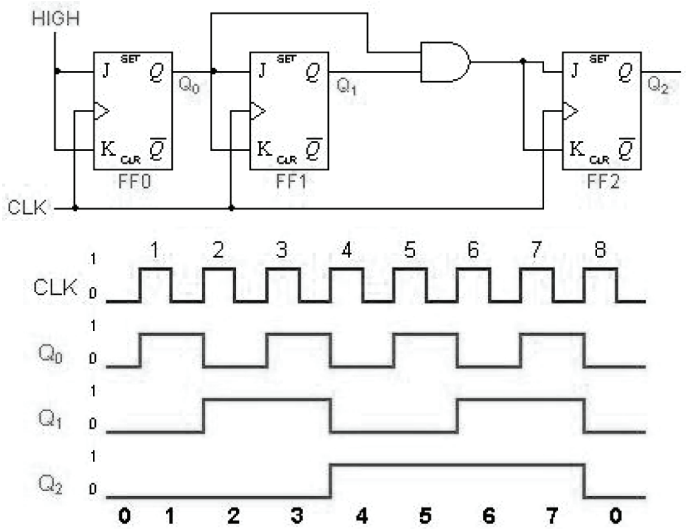


Figura 22 - Circuito e Forma de onda de um contador síncrono

Observando o diagrama de forma de onda, podemos entender o funcionamento. O ponto importante é que o flip-flop JK troca o estado de sua saída quando as entradas J e K estão em 1. Por outro lado, quando as entradas J e K estão em 0 a sua saída Q permanece no mesmo estado. Como JK do flip-flop FF0 está em 1, a saída Q0 troca em cada transição de clock, assim, o estado de Q1 somente vai trocar quando Q0 e também J e K do FF1 realizarem a transição de 1 para 0. O terceiro bit Q2 vai trocar apenas quando Q0 e Q1 realizarem a transição de 1 para 0, por isso a porta E.

Assim, conseguimos implementar um contador de três dígitos onde o sinal de relógio é aplicado simultaneamente, tornando o circuito mais confiável e seguro.

Síntese do capítulo

Neste capítulo apresentamos os conceitos básicos de Lógica Digital necessários para o entendimento das técnicas de Arquitetura de Computadores. Iniciamos com a apresentação das portas lógicas básicas, como E, OU, NOT, que possibilitam a construções de todos os circuitos lógicos. Em seguida, apresentamos os circuitos combinacionais mais tradicionais em uma arquitetura, como multiplexadores, decodificadores e somadores. Concluímos com a apresentação dos circuitos sequenciais, iniciando com os vários tipos de flip-flops e apresentando os circuitos de registrador de deslocamento e contadores.

Atividades de avaliação

1. Você observou que podemos construir flip-flops com diversas portas obtendo o mesmo resultado. Pesquise para saber as razões para escolher um ou outro tipo.
2. Analisando o circuito de um contador síncrono, imagine como você poderia construir um contador de números qua não fosse potência de dois.
3. Pesquise sobre linguagens de descrição de hardware (HDL), como VHDL e Verilog. Pense sobre a diferença no trabalho de desenvolver *software* e *hardware* nos dias atuais.

Leituras, filmes e sites

Leituras

FLOYD, Thomas. **Sistemas Digitais:** Fundamentos e Aplicações. Porto Alegre: Bookman, 2007.

ORDONEZ, Edward David et al. **Projeto, Desempenho e Aplicação de Sistemas Digitais em Circuitos Programáveis** (FPGAs). São Paulo: Bless, 2003.

TOOLEY, Mike. **Circuitos eletrônicos:** fundamentos e aplicações. Rio de Janeiro: Elsevier, 2007.

Filmes/Vídeos

Eletronica Digital - Introdução às Portas Lógicas - Porta NOT (Aula 22). Duração: 19min 19s. Vídeo aula que introduz o conceito de Portas Lógicas. Acesso em: https://www.youtube.com/watch?v=Afh8wmTUoVc

Eletrônica Digital - Projeto de Circuitos Combinacionais (Aula 44). Duração: 29min 59s. Vídeo aula que discute sobre Circuitos Combinacionais. Acesso em: https://www.youtube.com/watch?v=0G_uPyUosEc

Multiplexadores e circuito digital programável. Duração: 11min 06s. Nesta aula demonstram-se os multiplexadores (MUX) para que se possa implementar um circuito digital com a lógica combinacional programável! Acesso em: https://www.youtube.com/watch?v=lJjLq7M59lg

Sites

Apostila de Lógica Digital (em Português)

http://wwwusers.rdc.puc-rio.br/rmano/cl.html

Curso de Lógica Digital (em Português)

http://pt.wikiversity.org/wiki/Lógica_Digital

Catálogos de circuitos integrados digitais Texas Instruments (em Inglês)

http://focus.ti.com/logic/docs/logichome. tsp?sectionId=450&familyId=1

Referências

FLOYD, T. **Sistemas Digitais**. 9ª Ed. Editora: Bookman, 2007.

TOCCI, R, J. ; WIDMER, N. S. ; MOSS, G. L. **Sistemas Digitais:** Princípios e Aplicações. 10ª Ed. Editora: Pearson Prentice-Hall, 2007.

VAHID, F. **Sistemas Digitais**. 1ª Ed. Editora: Bookmark, 2008.

Parte 3

Estruturas de uma arquitetura de computadores

Capítulo 3

Unidade Central de Processamento (UCP)

Objetivos

Neste capítulo vamos apresentar detalhadamente todos os componentes que fazem parte de uma arquitetura de computador. Iniciamos apresentando a Unidade Central de Processamento, "o cérebro" do computador, responsável pelo controle do computador além das operações aritméticas.

Em seguida apresentamos as memórias, unidades para armazenamento de informações. Apresentamos, depois, os barramentos de um computador. Finalmente mostramos os dispositivos de entrada e de saída, ou seja, os dispositivos que interagem com o ser humano.

Introdução

A Unidade Central de Processamento (UCP), ou *Central Processing Unit* (CPU), é o módulo principal da arquitetura de um computador. Ela é responsável por realizar o controle do sistema de computador, além de executar funções de cálculo aritmético. O controle do sistema é realizado através da execução de um programa, isto é, uma sequência de comandos previamente definidos necessários para realizar um processamento.

Desde os primeiros computadores, sempre houve uma UCP, implementada em um gabinete ou em uma placa. Mas, atualmente, quando falamos em UCP, geralmente estamos nos referindo ao circuito integrado que realiza todas essas funções – o microprocessador.

A evolução tecnológica dos microprocessadores é surpreendentemente grande. A partir de microprocessadores que trabalhavam com relógio (clock) de alguns kHz e que podiam processar poucos milhares de instruções por segundo, atualmente atingiu-se relógios da ordem de 4GHz com poder de processamento de vários bilhões de instruções por segundo. A complexidade dos circuitos integrados também cresceu: de alguns milhares de transístores para centenas de milhões de transístores em um chip.

A UCP tem como função principal integrar todo o sistema do computador, isto é, realiza o controle de funcionamento de todas as unidades funcionais e é responsável pelo controle da execução de todos os programas do sistema.

1. História do Microprocessador

O primeiro microprocessador comercial foi lançado pela Intel em 1971, originalmente um circuito integrado especial para atender uma empresa japonesa que fabricava calculadoras eletrônicas. O processador Intel 4004 era um circuito integrado programável que trabalhava com registradores de 4 bits, clock de 740Khz, um conjunto de 46 instruções e possuía cerca de 2300 transístores. Apesar do insucesso do fabricante de calculadoras, o microprocessador programável 4004 foi utilizado por muitas outras aplicações.

Uma característica importante para avaliar a capacidade e o desempenho de um processador é o tamanho da palavra. O tamanho da palavra vai determinar o tamanho dos registradores, o tamanho do barramento e o tamanho da unidade aritmética. Quanto maior o tamanho da palavra, maior a capacidade de processamento.

A Intel prosseguiu com o desenvolvimento de novos microprocessadores: 8008, o 8080 e o 8085 (todos de 8 bits). O 8080 foi um grande sucesso e tornou-se a base para os primeiros microcomputadores no final da década de 1970 que utilizavam o sistema operacional CP/M. Anos mais tarde, a Zilog lançou o microprocessador Z80, com instruções compatíveis com o 8080 (compatível e com várias melhorias), que também fez grande sucesso. Esse foi o primeiro momento da história em que se fez um "clone" de microprocessador, dispositivo que permitia executar sem alteração programas e sistemas operacionais desenvolvidos para outros processadores (*hardware*).

Nesse instante, o desenvolvimento do *hardware* e do *software* tornaram-se independentes, pois, até então, o *software* era totalmente dependente de um hardware específico de um fabricante. A Motorola lançou a sua própria família de microprocessadores, o 68000. Todos esses microprocessadores de 8 bits foram usados em vários computadores pessoais (Sinclair, Apple, TRS-80, Commodore, etc).

A IBM decidiu entrar no mercado de computadores pessoais em 1981 lançando o seu IBM-PC, que utilizou um dos primeiros microprocessadores de 16 bits, o Intel 8088 (derivado do 8086 lançado em 1978), que viria a iniciar uma família de computadores que utilizamos até hoje. O interessante é que o processador 8088, apesar de mais recente que o 8086, utilizava um barramento externo de 8 bits (o 8086 tinha barramento externo de 16 bits), permitindo a utilização de componentes (memórias, I/O, etc) compatíveis com as famílias

anteriores, tornando o preço do IBM-PC muito próximo dos computadores de 8 bits, mas com capacidade superior. Aliado à marca IBM, um computador de maior capacidade e de preço semelhante tornou-se um enorme sucesso.

A Apple utilizava os processadores da Motorola da família 68000 (de 32 bits) nos seus computadores Macintosh. Outros fabricantes também lançaram os seus microprocessadores de 16 bits: a Zilog, o Z8000; a Texas Instruments, o TMS9900; a National Semiconductor, o 16032, mas nenhum fabricante teve tanto sucesso como a Intel.

A Intel foi lançando sucessivamente melhoramentos na sua linha 80X86, o 8086, 8088, 80186, 80188, 80286, 80386, 80486, Pentium, Pentium Pro, Pentium MMX, Pentium II, Pentium III, Pentium IV, Pentium M, Pentium D, Pentium Dual Core e Pentium Quad Core (em ordem cronológica). Algumas melhorias importantes nessa evolução foram o uso memória virtual e de multi-tarefa no 80386, o coprocessador matemático integrado no 80486 e o pipeline de processamento na linha Pentium.

Assim como a Zilog no caso do 8080, a AMD surgiu como fabricante de microprocessadores clones da família x86, mas, a partir de um certo momento, partiu para o desenvolvimento de sua própria linha de microprocessadores, ainda mantendo compatibilidade com os produtos Intel: K6, Athlon, Duron, Turion, Sempron, etc.

2. Arquitetura de uma Unidade Central de Processamento (UCP)

A função? fundamental da Unidade Central de Processamento (UCP), não obstante o formato físico e a arquitetura, é executar uma sequência de instruções armazenadas chamada **programa**. O programa é representado por uma série de números que são mantidos em algum tipo de memória no computador. Há quatro etapas que quase todos os processadores centrais executam em sua operação: a busca de instruções, a decodificação, a execução e escrita de dados. A figura ao lado mostra o diagrama em blocos de uma Unidade Central de Processamento (UCP).

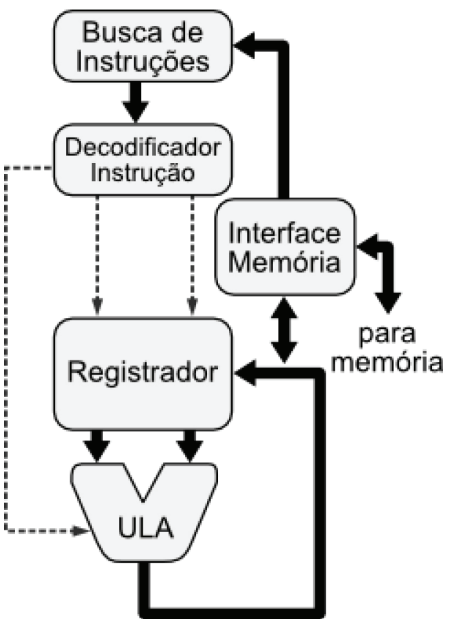


Figura 23 - Diagrama em blocos de uma Unidade Central de Processamento

A primeira etapa, a busca de instrução, envolve recuperar uma instrução (que é representada por um número ou por uma sequência dos números) na memória de programa. A posição da instrução em execução na memória de programa é determinada por um Contador de Programa (PC *Program Counter*[6]), que armazene um número que identifica o endereço de memória atual da instrução do programa. Ou seja, o Contador de Programa guarda o local do programa para a Unidade Central de Processamento.

[6] Program Counter é um registro utilizado em qualquer processador que guarda a posição de memória (endereço) onde se encontra a instrução do programa que será executada pelo processador.

Depois que uma instrução é buscada, o PC será incrementado do comprimento de instrução, para que aponte para a próxima instrução que será executada no ciclo seguinte. Geralmente, a memória de programa é um dispositivo mais lento que o processador, fazendo com que a Unidade Central de Processamento pare para esperar a instrução que será executada. Para isso, a maioria dos processadores modernos utiliza a técnica de Cache que armazena o programa em memórias mais rápidas que possibilitam um aumento na velocidade de execução do programa.

A instrução que a Unidade Central de Processamento busca na memória é usada para determinar o que irá fazer. Na etapa de decodificação, a instrução é quebrada em partes, e cada uma tem um significado para as outras partes da Unidade Central de Processamento. A maneira como um valor numérico de instrução é interpretado é definida pelo Conjunto de Instruções, conhecida também pela sigla ISA (*Instruction Set Architecture*).

Frequentemente, a primeira parte da instrução é composta por um grupo de números, chamado *opcode*, que vai indicar a operação que será executada. As partes restantes do número da instrução geralmente indicam a informação exigida para essa instrução, por exemplo, os operandos de uma operação aritmética. Tais operandos geralmente são dados de processamento, como um valor constante, ou um lugar para encontrar um valor um registrador ou um endereço de memória.

Nos projetos mais antigos de Unidades Central de Processamento, os blocos responsáveis pela decodificação da instrução eram dispositivos de *hardware* não alteráveis. Entretanto, em alguns processadores mais modernos e complexos, o Conjunto de Instrução é representado por um microprograma[7], desdobrando uma instrução mais complexa em várias instruções mais simples, por exemplo, uma multiplicação pode ser simplificada por uma sequência de somas. Um microprograma pode também ser reprogramável de modo que possa ser modificado para mudar a maneira como o processador decodifica as instruções mesmo depois que foi fabricado.

[7] Microprograma é o código básico de um processador, definindo instruções básicas para operações de controle de uma UCP.

Depois das etapas de busca de instrução e decodificação, a próxima etapa é a execução da instrução. Durante essa etapa, as várias partes do processador central são conectadas e podem executar a operação desejada. Se,

por exemplo, uma operação de adição foi solicitada, uma Unidade de Lógica e Aritmética (ULA) será conectada a um registrador de entradas e de saídas. Nas entradas são aplicados os números que serão adicionados, e a saída conterá a soma final.

A ULA contém os circuitos para executar operações aritméticas e lógicas das entradas (como a adição e operações lógicas). Se a operação da adição produz um resultado demasiado grande para que o processador consiga guardar, é marcada uma indicação de excesso aritmético em um registrador. Essa indicação, geralmente um "sim" ou "não", é conhecida como *flag*[8].

[8] Flag é um registrador simples que indica um estado de um processador, geralmente é constituído por um bit e pode assumir os valores 0 ou 1.

O passo final – a escrita de dados – simplesmente escreve os resultados da etapa da execução a algum local da memória. Os resultados podem ser escritos em algum registrador interno do processador para permitir o acesso mais rápido pelas instruções subseqüentes. Em outros casos, os resultados podem ser escritos em uma memória externa mais lenta, mas mais barata e maior. Alguns tipos de instruções manipulam o Contador de Programa, fazendo com que se mude a sequência de execução do programa.

Essas instruções são chamadas "saltos" e facilitam a criação de comportamentos como laços, execução condicional de programa (com o uso de um salto condicional) e execução de funções pré-definidas nos programas. Muitas instruções poderão mudar o estado em função de dígitos em um registro de indicação, os *flags*. Esses indicadores podem ser usados para influenciar como um programa se comporta. Por exemplo: uma instrução do tipo "comparação" considera dois valores e ajusta um número (*flag*) no registro de acordo com qual é maior. Esse flag pode então ser usado por uma instrução de salto condicional para alterar a sequência de um determinado fluxo de programa.

Após a escrita dos dados resultantes, haverá a repetição de todo o processo, com o ciclo de busca da instrução seguinte devido ao incremento do valor do Contador de Programa (PC). Se a instrução anterior era um salto, o Contador de Programa será modificado para conter o endereço da nova instrução estabelecido no salto (condicional ou não), e a execução de programa continua normalmente. Em alguns processadores mais complexos do que esse, múltiplas instruções podem ser buscadas, decodificadas e executadas simultaneamente. Essa metodologia é geralmente conhecida como *pipeline*, que é muito comum nos processadores atuais usados em computadores e em outros dispositivos.

A maioria de processadores e, certamente, a maioria de dispositivos de lógica sequencial são síncronos por natureza, isto é, são projetados para operar sobre a cadência de um sinal de sincronização. Esse sinal, conhecido como um sinal de relógio (*clock*), tem geralmente a forma de uma onda quadrada periódica. Calculando o tempo máximo que os sinais elétricos podem

ser transmitidos pelas trilhas do circuito de um microprocessador, os projetistas podem selecionar o período de relógio mais apropriado.

Esse período deve ser mais longo do que o tempo que um sinal se mova ou se propaga, nos piores casos hipotéticos.

Ao ajustar o período do relógio para um valor bem acima do pior das hipóteses de atraso de propagação, é possível projetar um processador de maneira que os dados se movam com segurança. Essa tática tem a vantagem de simplificar o circuito do processador, entretanto, temos a desvantagem de que o processador agora deve esperar pelos seus componentes mais lentos, mesmo que todos os demais sejam mais rápidos. Essa limitação pode ser compensada através de técnicas de paralelismo que possibilitam aumentar o desempenho.

Entretanto, as melhorias de arquitetura sozinhas não resolvem todos os problemas dos processadores com sincronismo global. Por exemplo: um sinal de relógio está sujeito aos atrasos de qualquer outro sinal elétrico de controle. Taxas de relógio mais elevadas em processadores cada vez mais complexos tornam difícil manter todos os sinais de relógio em todas as unidades funcionais em fase (sincronizado).

Isso obrigou, em muitos processadores modernos, a construir um circuito específico para garantir que todos os sinais de relógio sejam idênticos a fim evitar que o atraso de um sinal faça com que o processador funcione mal. Um outro problema sério que ocorre com o aumento da taxa de relógio é o aumento significativo da quantidade de calor dissipada pelo processador central. O aumento da taxa de relógio aumenta a mudança de estado das portas lógicas CMOS, que consomem energia apenas quando há mudança de estado, aumentando a dissipação de calor. Consequentemente, o aumento da taxa de relógio aumenta o calor dissipado, exigindo que o processador central necessite de dispositivos de refrigeração mais eficazes.

2.1. Unidade Lógica e Aritmética

A Unidade Lógica e Aritmética (ULA), ou *Arithmetic Logic Unit* (ALU), é a unidade da Unidade Central de Processamento (UCP, ou também CPU), responsável pela execução das operações aritméticas e lógicas. John von Neumann já propôs o conceito de ULA em 1945, quando construiu o computador EDVAC.

A ULA executa as principais operações lógicas e aritméticas que um computador precisa realizar. Ela faz as operações de soma, subtração, multiplicação e determina se um número é zero, positivo ou negativo. Além das funções aritméticas, uma ULA deve ser capaz de determinar se uma quantidade é igual, menor ou maior que outra. A ULA também realiza as funções lógicas

básicas como E, OU, OU-Exclusivo, assim como a negação, tanto com números quanto com caracteres.

A figura 24 mostra o diagrama lógico de uma ULA simples de 2 bits. Os dados de entrada, provavelmente de um registrador, são colocados nas entradas A(0), A(1) e B(0), B(1). As entradas OP(0), OP(1) e OP(2) determinam o tipo de operação a ser realizada. As saídas, onde o resultado final pode ser lido e escrito em um registrador, são OUT(0) e OUT(1). A entrada CARRY[9] IN serve para considerar o vai um de um circuito ULA cascateado, e a saída CARRY OUT serve para indicar o vai um para o circuito seguinte. Ligando-se uma série de ULAs através das entradas e saídas CARRY IN e CARRY OUT, podemos construir ULAs com tamanho de palavras maiores.

[9] Carry é um indicador de estouro, que indica quando o valor de um registrador supera o seu valor máximo admitido. Geralmente é utilizado nas operações de soma para indicar o vai um.

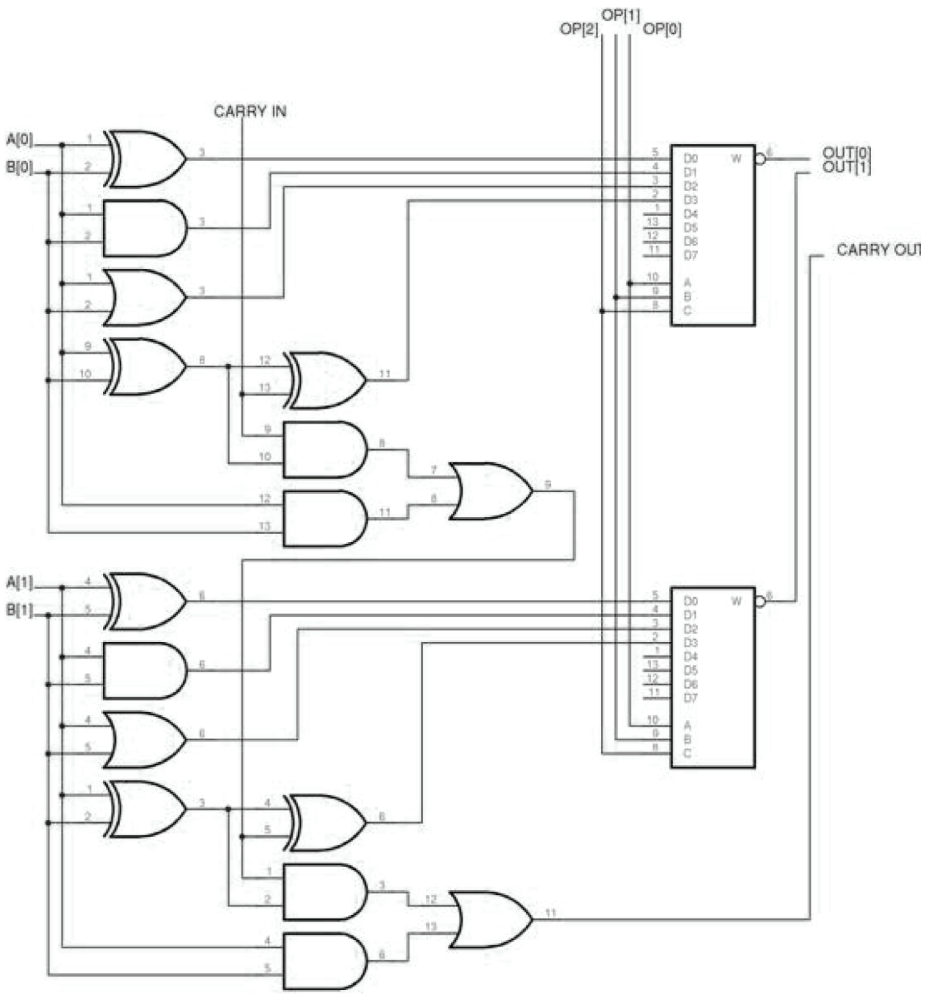


Figura 24 - Unidade Lógica e Aritmética de 2 bits (AND, OR, XOR e SOMA) [Eagle GFDL]

Para realizar corretamente as operações aritméticas, uma ULA deve operar números com a mesma base e o mesmo formato. Os primeiros computadores usavam diversos sistemas numéricos, bases numéricas, representações de sinais, complemento de um, complemento de dois, sinal-magnitude. Para cada um desses conjuntos de parâmetros é necessário realizar um projeto de ULA diferente, além da necessidade de realizar a conversão de todos os dados quando se trocava de computador. Nos computadores modernos (últimos 30 anos), quase sempre se utiliza número binário em representação de complemento de dois, representação mais simples para a ULA realizar adições e subtrações, e caracteres ASCII[10].

[10] ASCII é o acrônimo para American Standard Code for Information Interchange, é uma codificação de caracteres de oito bits que representa caracteres alfanume´ricos para permitir a representação de textos em um computador.

Uma ULA recebe geralmente dados dos registradores, que são processados, e os resultados da operação são armazenados nos registradores de saída. Teoricamente, é possível ler dados diretamente da memória e realizar a operação na ULA, porém o desempenho é impactado negativamente devido ao maior tempo de leitura de memória comparado com o tempo de cálculo.

A maioria das ULAs podem realizar as seguintes operações:

- Operações aritméticas (adição, subtração)
- Operações lógicas em bit: AND, OR, XOR e NOT
- Operações de deslocamento de bits (deslocamentos servem para realizar multiplicações e divisões por 2).

Apesar de ser possível implementar circuitos para realizar cálculos mais complexos, como logaritmos, exponenciais ou raízes quadradas, geralmente essas funções são realizadas através de algoritmos de *software*. O aumento da complexidade do circuito para funções pouco usadas pode não ser vantajoso para computadores de uso geral. Porém, máquinas de processamento numérico podem implementar essas funções em circuito.

Outro fator importante relacionado à representação de número em um computador é o tamanho e a precisão que um processador pode tratar. No caso de um processador binário, um bit se refere a uma posição significante no número representado no processador. O número de bits que um processador usa para representar os números é chamado de "tamanho da palavra", "largura da palavra" ou de "largura do barramento de dados". Ao se tratar especificamente de números inteiros, chamamos "precisão de número inteiro".

Esse número caracteriza as diversas arquiteturas de processador e, frequentemente, é um parâmetro importante para definir a capacidade do processador. Por exemplo: um processador de 8 bits trata uma faixa de números que podem ser representados por oito elementos binários (cada dígito que tem dois valores possíveis), isto é, 28 ou 256 números discretos. Assim, o tamanho do número inteiro vai definir o limite do *hardware* que vai definir a capacidade de tratamento de números inteiros que o *software* pode tratar.

A faixa de inteiros pode também definir o número de posições na memória que o processador central pode endereçar (localizar). Por exemplo: se um processador usa 32 bits para representar um endereço de memória, e cada endereço de memória representa um byte (8 bits), a quantidade máxima de memória que o processador pode endereçar é 232 bytes ou 4 GB. Essa é uma ideia muito simples sobre o espaço de endereçamento de um processador, mas muitos métodos de endereçamento mais complexos, como a paginação, possibilitam aumentar o tamanho da memória utilizando a mesma faixa de inteiro.

Convém lembrar que o tamanho da palavra é definido principalmente pela capacidade de endereçamento do processador. Um processador de alto desempenho usa 16, 32, 64 e, até mesmo, 128 bits de largura. Enquanto isso, microcontroladores com necessidades menores podem usar processadores de 4 ou de 8 bits.

2.2. Unidade de Ponto Flutuante

Da mesma maneira que uma ALU pode realizar operações entre números inteiros, uma Unidade de Ponto Flutuante (UPF) também realiza operações aritméticas entre dois números, mas eles realizam operações com número em representação de ponto flutuante, muito mais complexa que a representação de complemento para dois. Para realizar esses cálculos, uma UPF tem vários circuitos complexos, incluindo algumas ULAs internas. Usualmente engenheiros chamam uma ULA o circuito que realiza operações aritméticas com números inteiros em complemento para dois ou BCD[11].

[11] BCD é acrônimo de Binary Coded Decimal, que representa um tipo de codificação numérica para representar algarismos constituídos por 4 bits (representando números decimais de 0 à 15).

Unidade de Controle

A Unidade de Controle (UC) é responsável por gerar todos os sinais que controlam as operações no interior da UCP. Além disso, também tem o objetivo de controlar as interfaces de entrada e de saída do processador com o mundo externo. A UC realiza a decodificação de instruções e executa as ações dessas instruções.

A unidade de controle executa três ações básicas essenciais e definidas pelo próprio fabricante do processador: busca de instrução (fetch), decodificação e execução de instrução.

Assim sendo, todo processador, ao iniciar sua operação, realiza uma operação cíclica e sequencial dessas três ações. Dependendo do tipo de processador, a unidade de controle pode se ser fixa ou programável. A unidade fixa é aquela que já vem com todo o conjunto de instruções definido e programado pelo fabricante na fabricação do processador.

Na unidade programável, o microprograma, código que decodifica e executa as instruções é programado externamente no início de funcionamento do processador, permitindo a reconfiguração das instruções quando necessário.

Os microprocessadores da família Intel como 8080, 8088, 80286, 80386, 80486 e Pentium, possuem unidade de controle fixa. Um exemplo de unidade de controle programável pode ser visto nos processadores Bit Slices, arquitetura de *hardware* modular que permite ao projetista associar unidades de processamento e programar o conjunto de instruções. Outro exemplo é a utilização de Field-Programmable Gate Array (FPGA). O FPGA é uma matriz lógica programável que permite a implementação de soft processors, isto é, processadores definidos em *software* para um dispositivo FPGA, simulando o funcionamento de vários processadores de uso específico.

Inicialmente, a Unidade de Controle fornece o endereço de memória de onde deve retirar a instrução, geralmente composta por um byte ou mais. Essa instrução pode conter um código de operação (opcode[12]), ou um operando ou dado. Além de manter o controle sobre a posição de memória que contém a instrução atual que o computador está executando, a Unidade de Controle, ao decodificar o código de operação, informa à Unidade Lógica e Aritmética (ULA) qual operação a executar ou qual operação de escrita ou leitura de registrador ou memória deve executar.

[12] Opcode é um código de operação, representado em uma instrução de programa que realiza uma série de funções de comando.

Além das operações aritméticas, a Unidade de Controle também tem a capacidade de realizar operações lógicas, por exemplo: E, OU, XOR, comparação e deslocamento de bits para a direita e para a esquerda. Essas operações são realizadas pela Unidade Lógica e Aritmética (ULA), porém têm relação muito próxima da Unidade de Controle pois influencia o comportamento da UC. A grande maioria de dos fabricantes de microprocessadores implementam essas funções, mas podem atribuir um mnemônico diferente a cada uma delas, denominando-o conjunto de instruções de um determinado processador.

Outra característica muito importante em uma Unidade de Controle é que a arquitetura de um processador pode ser de dois tipos: orientada por registrador ou orientada para memória. Se for orientada para registradores, por exemplo, a arquitetura Intel, a ULA, após executar qualquer operação lógica ou aritmética, sempre vai armazenar o resultado no registrador acumulador. Se for orientada para memória, como é o caso dos microprocessadores da Motorola, nem sempre o resultado é armazenado no acumulador, podendo ser armazenado diretamente em qualquer posição de memória. Terminada a execução da instrução, a Unidade de Controle incrementa o contador de programa (PC) e inicia o processamento da próxima instrução. Essa instrução geralmente está localizada no próximo endereço de memória ou, em caso me-

nos comum, quando existe uma instrução explícita de desvio (salto), o computador vai executar a próxima instrução indicada pela instrução de desvio.

2.3. Ciclo de Instrução

O processo mais importante de uma Unidade de Controle é o Ciclo de Instrução, que determina como uma instrução vai ser executada. Um ciclo de instrução, também conhecido como ciclo buscar-descodificar-executar, é o procedimento pelo qual um computador processa uma instrução de linguagem-máquina ou um programa armazenado na sua memória

O termo buscar-e-executar é de uso geral. A instrução deve ser buscada da memória e, então, ser executada pelo processador central. Essa é a atividade fundamental de um computador, quando seu processador lê e executa uma série de instruções escritas em sua linguagem-máquina.

O processador de cada computador pode ter ciclos de instrução diferentes de acordo com a sua arquitetura e conjunto de instruções. As etapas de um Ciclo de Instrução são descritas a seguir:

1. Buscar a instrução na memória principal

O processador coloca o valor armazenado no contador de programa (PC) no barramento de endereço. O processador recebe então a instrução da memória principal através do barramento de dados para o registrador de dados (MDR – *Memory Data Register*). O valor do MDR é colocado então no registro de instrução atual (CIR Current Instruction Register), um circuito que mantêm a instrução temporariamente, de modo que possa ser decodificada e executada.

2. Decodificar a instrução

O decodificador de instrução interpreta e executa a instrução. O Registro de Instrução (IR Instruction Register) mantém a instrução atual, quando o contador de programa (PC) recalcula o endereço na memória da próxima instrução a ser executada.

3. Buscar dados da memória principal

Caso a instrução lida se refira a um dado armazenado na memória principal, neste momento, esse valor é lido. Chamamos essa instrução de Instrução de Endereçamento Indireto. Essa instrução lê os dados da memória principal e os coloca nos registradores do processador para permitir a realização das operações.

4. Executar a instrução

A partir do registrador de instrução, os valores que compõem a instrução são decodificados pela Unidade de Controle. Assim, a informação decodificada como uma sequência de sinais de controle são enviadas às unidades de função do processador para executar as ações exigidas pela instrução. Caso os parâmetros sejam operadores, os valores são encaminhados à Unidade de Lógica Aritmética (ULA) para operá-los e escrever o resultado de volta a um registro do processador. Um sinal de controle é enviado para trás de forma a confirmar a execução da instrução.

5. Armazenar o resultado

O resultado gerado pela operação executada é armazenado na memória principal ou emitido a um dispositivo de saída. Baseado no feedback do sinal de controle da ULA, o PC é incrementado para endereçar a instrução seguinte ou atualizado a um endereço diferente onde a instrução seguinte seja buscada. O ciclo é, então, repetido então.

Podemos separar o tempo de execução de uma instrução em duas fases: ciclo de busca. Estas etapas são as mesmas para cada instrução. O ciclo de busca processa a instrução da palavra de instrução que contém um opcode e um operando.

6. Executar o ciclo

As etapas 3 e 4 do ciclo de instrução são parte do ciclo da execução. Essas etapas mudarão a cada instrução.

A primeira etapa do ciclo da execução é a do Processo-Memória. Os dados são transferidos entre o processador e o módulo de I/O (memória). Então, em seguida são executadas as operações matemáticas com uso de dados e também operações lógicas para referência aos dados. As alterações principais são a etapa seguinte, na verdade, uma alteração na sequência das operações, por exemplo uma operação do salto.

3. Registradores

O registrador de uma Unidade Central de Processamento é um tipo de memória rápida e com pequena capacidade construída dentro da UCP e utilizada para o armazenamento temporário de dados no processamento. Pelo fato de estarem próximos da Unidade Lógica e Aritmética e de ser construída com circuitos de memória muito rápidos, possibilita o maior desempenho no processamento. Os registradores ficam no topo da hierarquia de memória, por isso é a forma mais rápida mas também, mais cara de se armazenar um dado.

Os registradores são utilizados na execução de programas de computadores para disponibilizar um local para armazenar dados temporários. Na maioria dos computadores modernos, o processador copia as informações (dados), guardados na memória externa, para um registrador. As instruções que realizam algum cálculo numérico ou de controle utilizam esses dados, agora lidos em maior velocidade, são executadas pelo processador e, finalmente, os resultados da operação são movidos de volta para a memória principal.

Podemos classificar os registradores em duas categorias:

- **Registradores de dados:** utilizados para armazenar valores numéricos ou caracteres, tais como números inteiros, pontos flutuantes ou caracteres ASCII. Em algumas UCPs existe um registrador de dados especial, chamado acumulador, que recebe o dado de operações aritméticas e é utilizado implicitamente em muitas operações. O acumulador é usado sempre como um operando de qualquer operação e onde as funções de comparação podem ser realizadas (ele implementa funções de comparação). As UCPs mais atuais dispõem de vários registradores com as funções de acumulador, portanto não necessita mover dados para um registrador específico.
- **Registradores de Endereço:** são registradores que recebem o endereço de um objeto. O principal registrador de endereço é o PC ou o indicador de endereço da instrução em execução. Outro tipo de informação são os "ponteiros" (variáveis contendo o endereço), que apontam para um determinado dado na memória.

4. Processador CISC/RISC

Os processadores se classificam em duas grandes famílias, conforme a característica do seu conjunto de instruções: CISC e RISC. Um computador com conjunto de instruções complexas (CISC – *Complex Instruction Set Computer*) é uma arquitetura em que cada instrução pode executar diversas operações de baixo nível, tais como uma leitura da memória, uma operação aritmética e uma escrita na memória, usando uma única instrução. O termo foi inventado em contraste com a sigla RISC, computador com conjunto de instruções reduzidas (RISC - *Reduced Instruction Set Computer*).

O exemplo mais comum de arquitetura de processador CISC é a família Intel x86. No entanto, vários outros computadores também utilizam essa arquitetura, como System/360, PDP-11, VAX, 68000.

Durante os anos 70, avaliou-se que o crescente aumento da complexidade das linguagens de programação de computadores poderia ser mais bem executada se fossem usadas instruções mais complexas (mais funções

na instrução) em menos ciclos. Algumas novas instruções foram adicionadas para sofisticar a linguagem "assembly" para serem mais bem aproveitadas pelas linguagens de alto nível. Os compiladores foram atualizados para aproveitar-se dessas instruções mais complexas.

Os benefícios de instruções semanticamente mais ricas com codificação mais compacta podem ser considerados como um grande aumento de desempenho para a compilação e para a execução de programas gerados pelos compiladores. Como as memórias são limitadas no tamanho e na velocidade, o código compacto traria muitos benefícios. Naturalmente, a razão fundamental dessa melhoria é que as memórias (isto é, RAM dinâmicas) são significativamente mais lentas quando comparadas ao processador central.

Apesar da diferença entre RISC e CISC, elas tornaram-se menos significativas com a evolução das arquiteturas CISC. O processador Intel 486 implementou o *pipeline* de instruções, sendo seguido pela AMD, Cyrix, e IBM. Esses processadores quebraram cada instrução em um subconjunto de instruções razoavelmente simples que são executadas em sequência (*pipeline*), característica muito semelhante a um conjunto de instruções típico de um processador RISC.

A geração Pentium era uma versão superescalar desse princípio, mais semelhante ao RISC. Os processadores x86 modernos decodificam e quebram instruções em sequências dinâmicas de microoperações internas, que não somente os ajuda a executar um subconjunto maior de instruções em uma forma de *pipeline* mas também facilitam uma otimização mais avançada do paralelismo de microinstrução, aumentando o seu desempenho.

O acrônimo RISC (Reduced Instruction Set Computer) indica um processador com conjunto de instruções reduzido, uma estratégia de projeto de conjunto de instruções de processador que enfatiza a simplificação das instruções, de forma que a simplificação pode aumentar o desempenho, já que as instruções executam muito mais rapidamente.

O motivo dessa estratégia é que a maioria das funções de um microprocessador são simples, simplificando, assim, a arquitetura e tornando mais eficiente esse processamento. Para as funções complexas, um processador RISC é muito ineficiente, comparado com um processador CISC, mas como a necessidade dessas funções é menor, o desempenho global acaba ficando melhor. As principais famílias de processadores RISC são: Alfa, MIPS, PA-RISC, PowerPC e SPARC.

Algumas observações que justificaram os primeiros projetos de processadores RISC em 1975 era que a compilação de programas em uma quantidade de memória limitada, nessa época na ordem de alguns Kbytes,

não permitia o aproveitamento integral do conjunto de instruções complexas, maior justificativa das arquiteturas CISC. Além disso, o uso de esquema de endereçamento complexo torna necessária a utilização de muitos ciclos de leitura de memória para executar um comando.

A proposta RISC é que funções seriam mais bem executadas através de sequências de instruções mais simples, executadas a uma maior velocidade devido à possibilidade de se utilizar relógios (clocks) maiores. Outro ponto favorável é a utilização de instruções com comprimento fixo e restringindo as operações aritméticas aos registros internos do processador (não faz operações aritméticas em memória), o que facilitou a implementação de pipelines, reduzindo o tempo de leitura e a decodificação de instruções.

É um engano achar que, em um processador com "conjunto de instruções reduzidas", as instruções são simplesmente eliminadas, tendo por resultado um conjunto menor das instruções. Na verdade ao longo dos anos, o conjunto de instruções dos processadores RISC tem crescido no tamanho e, hoje, muitos deles têm um conjunto de instruções maior do que muitos processadores CISC.

Alguns processadores do RISC, tais como o Transputer da INMOS, têm o conjunto de instrução tão grande quanto um IBM Sistema/370, tipicamente um processador CISC. Pelo outro lado, o DEC PDP-8, máquina claramente equipada com processador CISC, porque muitas de suas instruções envolvem acessos de memória múltiplos, tem somente 8 instruções básicas mais algumas instruções estendidas.

Outras características encontradas tipicamente nas arquiteturas RISC são:

- formato de instrução uniforme, usando uma única palavra com o *opcode* nas mesmas posições de bit em cada instrução, exigindo menos decodificação;
- registros de uso geral idênticos, permitindo que algum registro seja usado em algum contexto, simplificando o projeto do compilador (embora normalmente há uns registros de vírgula flutuante);
- modalidades de endereçamento simples. O endereçamento complexo executado através das sequências da aritmética e/ou operações de carga-armazena;
- no entanto, devemos ter em mente que as exceções abundam, naturalmente, tanto dentro do mundo CISC e do RISC.

Concluindo, podemos dizer que, apesar das características marcantes que diferenciam um processador CISC de um RISC, recentemente essas diferenção tem se reduzido. Podemos dizer que ambas as técnicas são utilizadas nos processadores modernos tornando-se difícil classificar um processador.

4.1. Melhorando o desempenho de processadores

Algumas técnicas podem ser utilizadas para melhorar o desempenho dos processadores:

- Processamento em *pipeline*.
- Cache de memória dentro do chip do processador.
- Previsão de desvio (*branch prediction*).
- Execução especulativa (*speculative execution*).

1. Pesquise uma arquitetura e um conjunto de instrução de um processador simples (microcontrolador).
2. Uma unidade de ponto flutuante é essencial? Você pode simular suas funções através de software?
3. Procure a especificação de microprocessadores modernos e veja quais funcionalidades para melhoria do desempenho eles implementam.

Capítulo 4

Memória

Introdução

Durante o processamento em um computador, é necessário guardar os dados em um dispositivo seguro. Além disso, o processador é muito pequeno para guardar um programa grande, sendo necessário um local externo para guardá-lo. O dispositivo que pode fazer o armazenamento de dados e a programas no computador é chamado **memória**.

Memória é um componente de computador, dispositivos ou mídia da gravação que retêm os dados digitais usados pelo computando durante um intervalo do tempo. É um dos componentes fundamentais de qualquer computador moderno, que, junto com uma Unidade Central de Processamento, constitui o esqueleto básico de um computador desde a sua invenção.

Quando falamos em memória, geralmente nos referimos a um dispositivo semicondutor conhecido como Memória de Acesso Aleatório (RAM – *Random Access Memory*), dispositivo com alta capacidade, rápido, mas provisório. No entanto, também podemos chamar de memória dispositivos de armazenamento de alta capacidade, geralmente chamada de memória de massa - discos óticos, discos rígidos magnéticos como movimentações do disco rígido, e outros.

Estes dispositivos são mais lentos do que a RAM, mas podem armazenar os dados de forma permanente. Historicamente, a memória RAM[13] e a memória de massa, respectivamente memória principal e memória secundária ou memória interna e memória externa.

[13] Memória RAM é uma memória semicondutora volátil (perde os dados quando desligada) de alta capacidade e alta velocidade.

1. Características de memórias

As memórias possuem algumas características, quais sejam:

Localização: quanto à localização, as memória podem estar dentro da UCP, interna ou externa.

Capacidade: a capacidade de uma memória é definida pelo tamanho da palavra (largura) e pela quantidade de palavras (comprimento).

Volatilidade: a memória pode ser temporária ou permanente.

- **Memória temporária:** uma memória permanente exige energia constante para manter as informações armazenadas. As tecnologias de memória temporária atualmente são as mais rápidas (apesar de não ser uma régua universal) e são usado principalmente no armazenamento primário. Como exemplo dessa tecnologia, temos registradores, cache, memória RAM.
- **Memória permanente:** reterá a informação armazenada, mesmo se não é fornecida energia elétrica constantemente. É apropriada para o armazenamento da informação por um longo prazo. Usado para a maioria dos armazenamentos secundário, terciário, etc. Como exemplos dessa tecnologia, temos disco rígido, memória flash, disco ótico, etc.

Quantos aos métodos de acesso podem ser:

- **Sequencial:** lê do início até o ponto de acesso. Ex: fita.
- **Direta:** acessa o ponto direto e depende da posição anterior. Ex: disco.
- **Randômico:** acesso individual, não depende da posição anterior. Ex: memória RAM.
- **Associativa:** acesso baseado na comparação do conteúdo. Ex: memória cache.

Quanto à hierarquia, podem ser:

- **Registradores:** localizado dentro da CPU.
- **Memória Interna ou Principal:** memória RAM e cache.
- **Memória Externa:** armazenamento de massa.

O desempenho pode ser medido por alguns parâmetros, como:

- **Tempo de acesso:** Tempo entre a aplicação do endereço no barramento de endereços e a disponibilidade do dados no barramento de dados.
- **Taxa de transferência:** Taxa na qual os dados são enviados.
- **Ciclo de memória:** Tempo requerido para que a memória possa executar um novo acesso e ler o próximo dado.

O **tempo de acesso** é o tempo que se leva para alcançar uma determinada posição na memória. A unidade dessa medida é, tipicamente, nanossegundo para a memória primária, milissegundo para a memória secundária, e segundos para a memória terciária. Também é comum diferenciar o tempo de leitura do

tempo de escrita (esse geralmente maior) e, no caso do armazenamento de acesso sequencial, o tempo mínimo, o tempo máximo e o tempo médio.

O tempo necessário para realizar uma escrita ou uma leitura em uma memória de acesso não aleatório é definido pela equação Tn = Ta + (N/R) , onde Tn é o tempo médio para ler/escrever N bytes, Ta é o tempo de acesso, N é a quantidade de bytes, e R é a taxa de transferência.

A **taxa de transferência** é a taxa em que a informação pode ser lida ou escrita na memória. No armazenamento de dados do computador, a taxa de transferência é expressada geralmente nos termos de megabytes por segundo ou MB/s, embora também se use a taxa em bits por segundo. Como no tempo de acesso, as taxas de tranferência de leitura e a escrita podem ser diferentes. Também é usual de utilizar, para acesso sequencial, a taxa mínima, a taxa máxima e a taxa média.

O **ciclo de memória** estabelece o tempo em que uma memória precisa esperar para realizar uma nova leitura e escrita. A unidade é a mesma do tempo de acesso: nanossegundo para a memória primária, milissegundo para a memória secundária, e segundos para a memória terciária.

2. Tipos de memória

As memórias podem ser dos seguintes tipos:

- **Semicondutor:** Ex: RAM, ROM, Flash
- **Magnética:** Ex: Disco rígido, fita
- **Ótica:** Ex: CD
- **Outras:** Ex: Bubble, holograma

3. Hierarquia de Memória

A memória é um dos componentes principais de um computador, pois é responsável pelo armazenamento de informações de curto e de longo prazo. As memórias são classificadas em uma hierarquia conforme a sua proximidade com o processador, sua capacidade e sua velocidade. A Figura 25 mostra um diagrama da hierarquia de memórias em um computador.

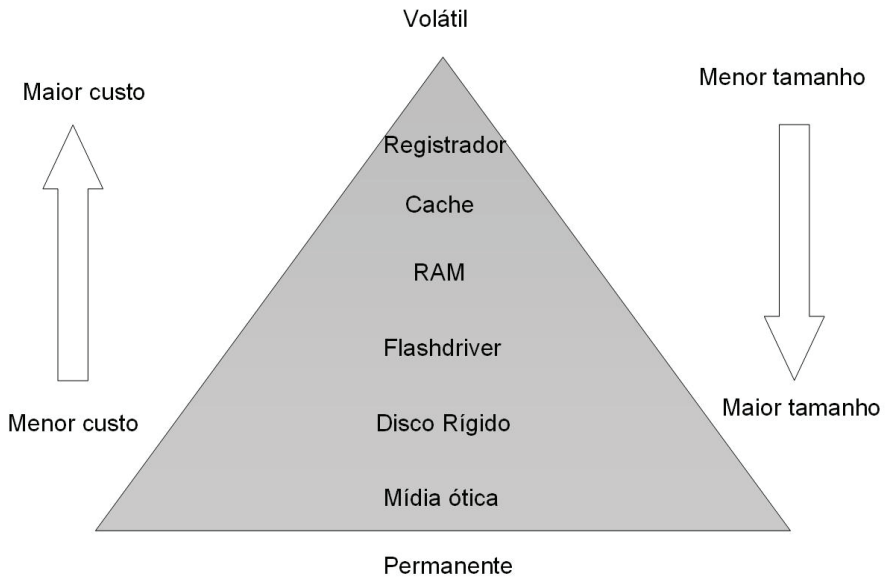


Figura 25 - Diagrama da Hioerarquia de Memória

As memórias são um dos mais importantes componentes de um computador. Historicamente, os primeiros computadores já usavam memórias tipo linha de atraso, tubos de Williams ou cilindros magnéticos. Em 1954, esses métodos de baixa confiabilidade foram substituídos pela memória de núcleo magnético de ferrite, confiável, mas de dimensão grande. A grande revolução ocorreu com a invenção do transístor, que permitiu a miniaturização da memória eletrônica, cada vez menor com uso do circuito integrado.

Isso conduziu à criação da memória de acesso aleatório moderna (RAM *Random Access Memory*), uma memória extremamente densa, barata, mas que é volátil, isto é perde a informação quando desliga a energia.

A **memória primária** (ou memória central ou memória interna) é a única acessível diretamente pelo processador. O processador central continuamente lê as instruções armazenadas lá e executa-as.

Os **registradores** do processador estão localizados dentro do processador. Cada registro armazena tipicamente uma palavra de dados (por exemplo, 32 ou 64 bits conforme a largura do processador). As instruções do processador instruem a unidade lógica e aritmética para executar cálculos ou outras operações utilizando os dados armazenados nos registradores. Os registradores são os dispositivos que usam a tecnologia rápida de todos os tipos de memória em um computador, além de ser a mais caro, por isso, seu tamanho é pequeno.

O **cache** de memória de um computador é um estágio intermediário entre registradores muito rápidos e a memória principal (RAM), mais lenta. Ele foi criado unicamente para aumentar o desempenho do computador. A informação na memória principal mais ativamente usada pelo processador é duplicada na memória cache, que é mais rápida, mas com pouca capacidade. Com os dados na memória cache, mais rápida, o processador consegue processá-las mais rapidamente comparado com a situação de estarem armazenadas na memória RAM.

Por outro lado, a memória RAM é muito mais lenta e muito maior do que os registradores do processador. Também é comum utilizar uma hierarquia de várias camadas de memória cachê, assim teremos memórias cache secundária, terciárias, etc. Conforme baixando na hierarquia, as memórias tornam-se mais lentas e com capacidades maiores.

A memória principal, também chamada memória **RAM** (RAM *Random Access Memory*), é conectada diretamente ou indiretamente ao processador através de um barramento da memória. É constituída, na verdade, por dois barramentos: um barramento de endereço e um barramento de dados. O processador envia inicialmente um número através do barramento de endereço, um número chamado **endereço de memória**, que indica a posição desejada dos dados na memória.

Aí, então, o processador lê ou escreve os dados colocados no barramento de dados. Eventualmente, pode haver uma Unidade de Gerência de Memória (MMU, *Memory Management Unit*), um dispositivo colocado entre o processador e a memória RAM, que calcula o endereço real da memória quando se usa, por exemplo, uma abstração de memória virtual.

Devido ao fato de que as memórias RAM são voláteis, isto é, o armazenamento somente perdura enquanto houver energia no computador. Um computador que contém somente memória RAM não teria um local para ler as instruções a fim de fazer o computador funcionar.

Um computador geralmente tem uma memória permanente que contém um pequeno programa de início (BIOS, *Basic Input/Output System*), que instrui o computador a ler um programa maior em uma unidade de armazenamento secundário permanente (um disco rígido , por exemplo) para a memória RAM e começar executá-lo. Essa memória permanente usada para guardar a BIOS é chamada ROM, memória apenas de leitura. Os computadores modernos permitem a gravação dessa memória ROM para atualizar a BIOS, mas, como essa atualização é realizada apenas esporadicamente, os conceitos aqui apresentados podem ser mantidos.

A **memória secundária** (ou a memória externa) difere da memória primária por que não é diretamente acessível pelo processador central. O computador geralmente usa suas interfaces de entrada/saída para ter acesso ao armazenamento secundário e transfere os dados desejados usando a área armazenamento intermediário localizadas nessas interfaces. A memória secundária não perde os dados quando o dispositivo é desligado. Uma unidade de memória secundária é, tipicamente, uma ordem de valor mais barata que a memória primária, além de armazenar uma quantidade de dados muito maior.

Nos computadores modernos, o armazenamento secundário é realizado com um disco rígido. O tempo para alcançar um byte da informação armazenado em um disco rígido é, tipicamente, alguns milésimos de segundo, ou milissegundos. Por outro lado, o tempo típico para alcançar um byte armazenado na memória de acesso aleatório (RAM) é medido em bilionésimos de segundo, ou nanossegundos. Isso ilustra a diferença significativa do tempo de acesso que distingue as memórias semicondutoras dos dispositivos de armazenamento magnético: os disco rígidos são, tipicamente, milhões de vezes mais lentos do que a memória.

Os dispositivos de armazenamento ótico, tais como CD ou DVD, têm tempos de acesso maiores ainda. Nas unidades de disco, uma vez que a cabeça de leitura/gravação no disco alcança a posição correta de onde se encontra o dado desejado, os dados subsequentes na trilha são muito rapidamente alcançados. Em consequência, a fim de esconder o alto tempo de busca inicial, os dados de discos são transferidos em grandes blocos contíguos.

Alguns outros exemplos de tecnologias de armazenamento secundário são: a memória Flash (por exemplo, drives USB), os discos flexíveis, a fita magnética, a fita de papel, os cartões perfurados, etc

A maioria dos computadores usa o conceito da **memória virtual**, permitindo o aumento da capacidade de uma memória quando necessário. Quando a memória primária se enche (por exemplo, memória RAM), o sistema move os pedaços menos usados (páginas) para um dispositivo de memória secundária (por exemplo, disco rígido), recuperando-os mais tarde quando são necessários. Como o armazenamento secundário é mais lento que o armazenamento primário, o desempenho de sistema total fica degradado. Porém, o sistema não para de funcionar devido à exaustão da memória primária.

A **memória terciária**, ou armazenamento terciário, fornece um terceiro nível de armazenamento. É uma memória com tempos de acesso muito longos, que são usados para guardar dados por longos períodos. Tipicamente, envolve um mecanismo robótico que monte e desmonte média removíveis de memória de massa, como fitas e discos. Esses dados são copiados frequentemente do armazenamento secundário. São usados para arquivamento

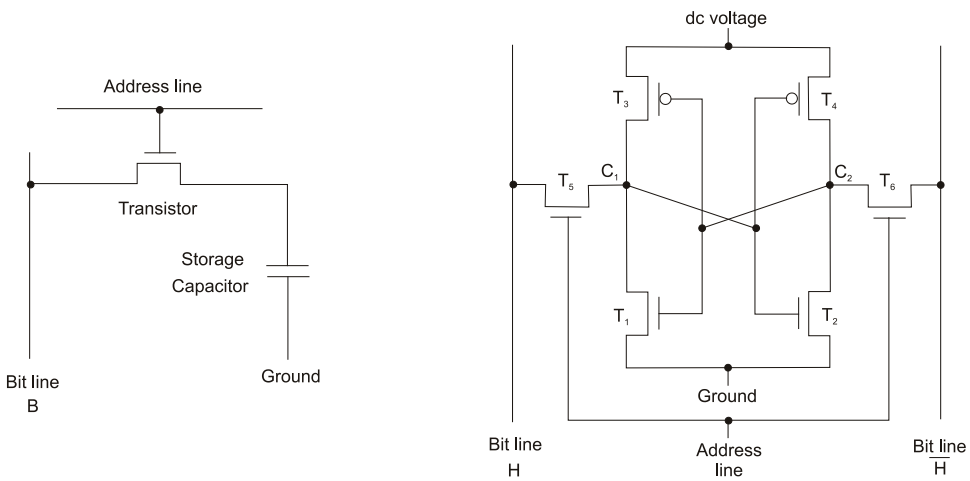
das informações, e sua velocidade é muito mais lenta que o armazenamento secundário (por exemplo, 5-60 segundos contra 1-10 milissegundos). Como exemplo, temos bibliotecas de fita e jukeboxes óticos.

Quando um computador precisa ler uma informação no armazenamento terciário, consultará primeiramente uma base de dados de catálogo para determinar qual a fita ou qual disco contêm a informação. Em seguida, o computador instruirá um braço robótico para buscar a mídia e colocá-la na unidade de leitura. Quando o computador terminar de ler a informação, o braço robótico retornará a mídia a seu lugar na biblioteca.

3.1. Memória Interna

As memórias semicondutoras podem ser do tipo RAM ou ROM. As memórias RAM podem ser lidas ou escritas e são voláteis. A memória ROM é apenas para leitura, mas é permanente.

As memórias RAM podem ser estáticas ou dinâmicas. Suas características comuns são: acesso randômico, isto é, as posições de memória podem ser lidas/escritas em qualquer ordem, e são voláteis, isto é, o armazenamento é temporário; quando se desliga a energia os dados são perdidos. A figura abaixo mostra um esquema de uma memória RAM dinâmica e estática.



(a) Dynamic RAM (DRAM) cell

(b) Static RAM (SRAM) cell

Figura 26 - Esquema de uma célula de memória RAM dinâmica e estática

3.2. Memória RAM Estática

Os bits são armazenados em chaves ON/OFF (flip-flops). Os dados permanecem até o computador ser desligado, não precisando ser refrescadas. Sua construção é complexa e ocupa uma grande área, por isso é mais cara. A vantagem é que são muito rápidas, sendo apropriadas para contrução de memória cache.

3.3. Memória RAM Dinâmica

Os bits em uma memória dinâmica são armazenados em capacitores que se decarregam com o tempo, por isso é necessário\textit{refrescar} o valor armazenado (refresh). Apesar da necessidade de um circuito especial para realizar o refresh, a construção de memória dinâmica é simples, ocupa menos espaço e barata, permitindo a construção de módulos grandes de memória. No entanto, elas são mais lentas que a memória estáticas, sendo mais adequadas para construir a memória principal.

O circuito de refresh é geralmente incluso no próprio chip e consiste em reescrever os dados para regenerar as informações. Por essa razão, há uma queda do desempenho do sistema.

3.4. Memória ROM

As memórias ROM tem armazenamento permanente e são usadas para guardar microprogramas e BIOS\footnote{Basic I/O System - Sistema básico de um computador}.

As memórias ROM podem ser:

- **Gravadas na fabricação:** cara para lotes pequenos.
- **PROM:** são programadas apenas uma vez.
- **EPROM:** são programadas eletricamente e apagadas por luz UV (ultravioleta).
- **EEPROM:** são programamdas e apagadas eletricamente (apaga um byte por vez).
- **FLASH:** são programadas e apagadas eletricamente (apaga toda memória ou um bloco).

4. Regeneração de memória dinâmica (refresh)

O circuito de regeneração geralmente é incluído no chip, sendo desnecessário qualquer circuito adicional. As etapas da regeneração são:

a) desabilita o chip.

b) contador ativa todas as linhas.

c) lê conteúdo de cada posição de memória.

d) reescreve valor na mesma posição.

O processo de regeneração consome tempo da memória, causando uma redução no desempenho. Mostramos, na figura abaixo, um diagrama de uma memória RAM com a regeneração de memória (refresh).

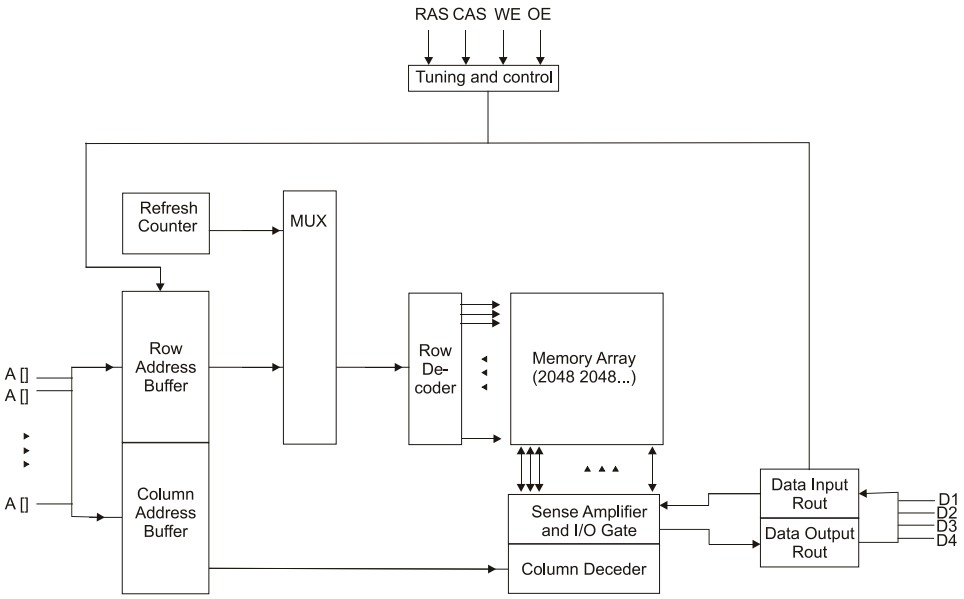


Figura 27 - Diagrama de uma memória RAM dinâmica

5. Correção de Erro

Podemos classificar os erros em memórias de dois tipos:

- **Falha grave:** defeito de *hardware* permanente.
- **Falha amena:** falha randômica, não destrutiva e produzida por ruído.

Alguns mecanismos são capazes de detectar e corrigir um erro. A figura a seguir mostra o funcionamento de um mecanismo de correção de erros.

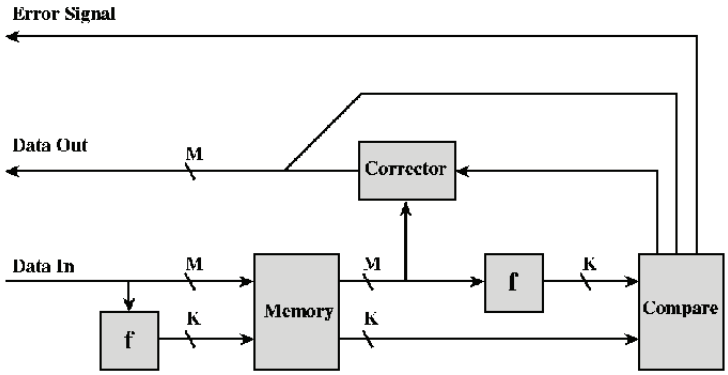


Figura 28 - Mecanismo de correção de erro

Um exemplo do algoritmo do Código de Hamming é mostrado na figura abaixo. O dado original é colocado nos 4 compartimentos internos conforme figura (a). Os três circulos A, B e C se interceptam e, em cada compartimento vazio, colocamos um bit de paridade, isto é, em cada círculo, a quantidade de bits "1" é sempre par (b). Supondo que haja um erro em um bit de dado (figura (c)) podemos facilmente descobri-lo.

Ao calcular a paridade verificamos que há uma divergência nos círculos A e C, mas não no B. O único compartimento pertencente a A e C, mas não a B é o local da ocorrência do erro, mostrado na figura (d). O código de Hamming não é eficiente em termos de tamanho de palavra pois, para corrigir uma palavra de 4 bits, precisamos adicionar mais 3 bits de paridade.

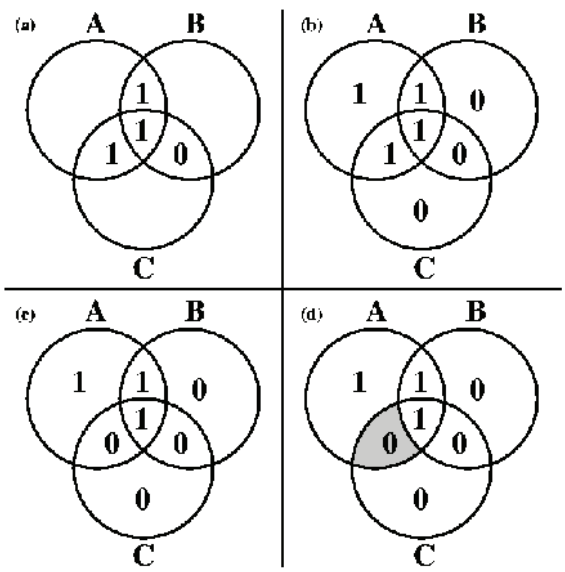


Figura 29 - Código de correção de erro de Hamming

6. Memória Cache

Memória cache é uma pequena quantidade de memória rápida colocada entre a CPU e a memória principal. Eventualmente, ela pode ser colocada dentro do chip da CPU. A idéia é que a memória cache leia um bloco grande da memória principal e sirva à CPU palavras isoladas para otimizar o desempenho. A figura a seguir mostra um diagrama da arquitetura de um computador com memória cache.

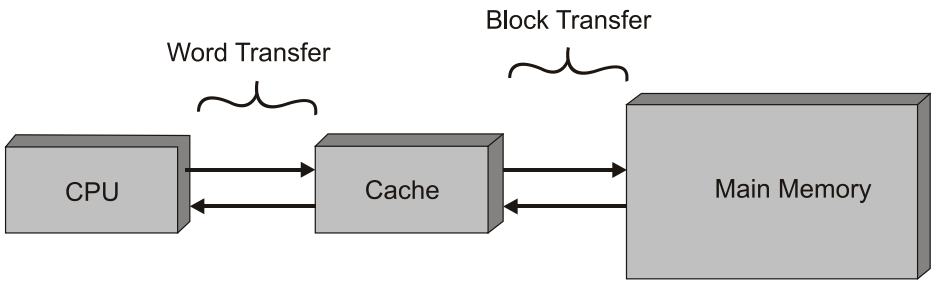


Figura 30 - Diagrama de um computador com memória cache

O funcionamento de uma memória cache é apresentado a seguir:

a) A CPU solicita uma posição de memória.

b) Verifica se essa posição está no cache.

c) Se está, pega no cache.

d) Se não, requisita bloco da memória principal.

e) Inclui rótulo para identificar bloco de memória

f) CPU lê a memória desejada.

g) Muito provavelmente, a próxima posição solicitada pela CPU deverá estar no cache.

A escolha do tamanho de uma memória cache não é trivial. Inicialmente, quanto maior a quantidade de memória rápida melhor será o desempenho do computador. Mas as memórias cache são memórias rápidas que têm um custo elevado, então quanto menos memória, menor o custo do computador.

Se a memória cache for pequena, é necessário muitas buscas à memória principal, degradando o desempenho. Como verificar a existência de uma posição de memória no cache toma tempo, o sistema pode ficar até mais lento do que sem o uso de cache. Apesar de o desempenho aumentar com o aumento da memória cache, ele não aumenta significativamente a partir de um certo ponto. A escolha do tamanho de cache é empírica.

7. Mapeamento

Um detalhe importante na construção de memória cache é a forma como se mapeia a memória principal na memória cache. Na figura abaixo, um bloco de tamanho K e palavras de tamanho W da memória principal mapeado em um bloco de K palavras na memória cache. O campo *tag* identifica a localização do bloco N na memória principal.

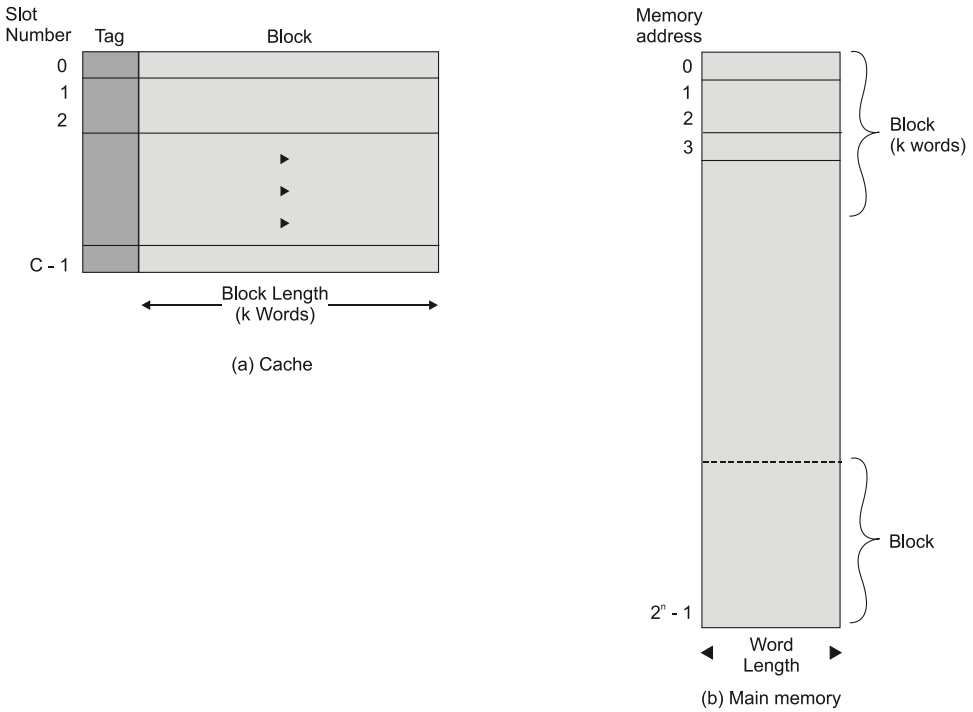


Figura 31 - Mapeamento de memória cache

7.1. Mapeamento Direto

O mapeamento direto mapeia cada bloco de memória principal em uma linha da memória cache exclusiva. O tag apenas indica o bloco da memória. A figura a seguir mostra um diagrama de memória cache com mapeamento direto.

O mapeamento direto é o mais simples e barato porque a localização é fixa para cada bloco. O problema é quando o programa acessa dois blocos que mapeiam na mesma linha, exigindo a troca frequente do contexto.

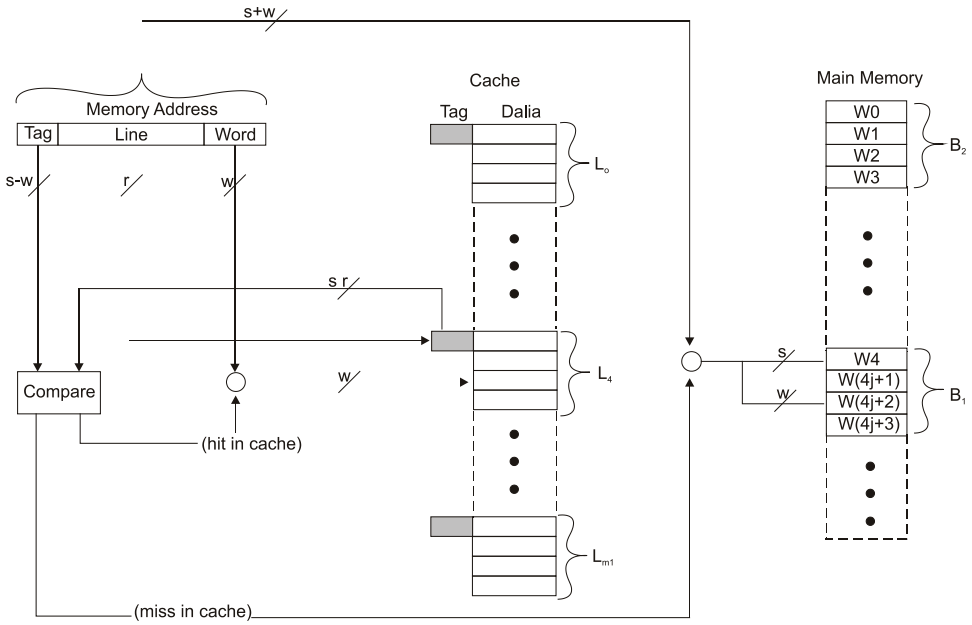


Figura 32 - Diagrama de uma memória cache com mapeamento direto

7.2. Mapeamento Associativo

No mapeamento associativo, um bloco de memória principal pode ser mapeado em qualquer linha da memória cache. A figura abaixo mostra um diagrama de memória cache com mapeamento associativo.

A identificação da memória é realizada com o tag e a palavra. O tag identifica o bloco de memória e é verificado a cada busca de memória. O mecanismo é muito eficiente, mas o circuito é mais complexo e caro.

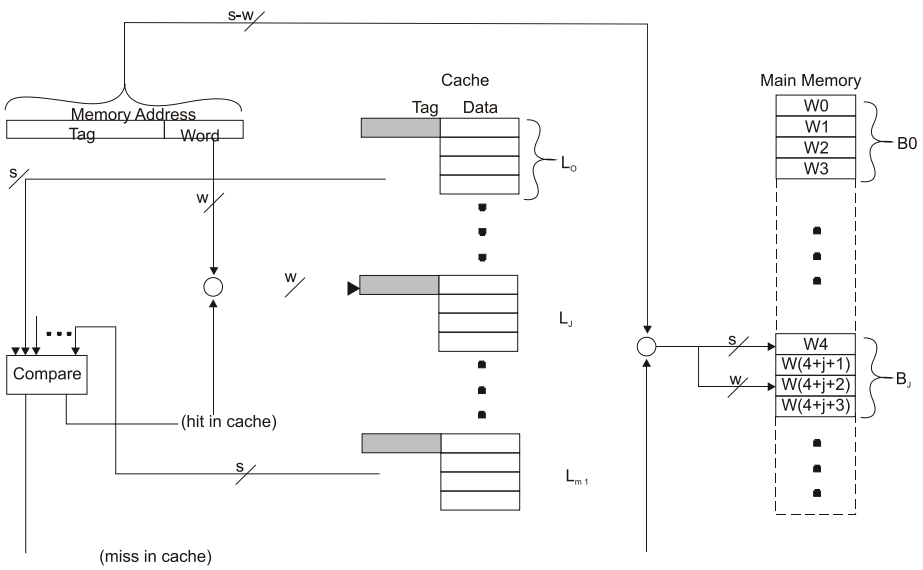


Figura 33 - Diagrama de uma memória cache com mapeamento associativo

7.3. Mapeamento Associativo por Conjunto

O mapeamento associativo por conjunto é uma solução intermediária entre memória associativa e memória direta. A figura abaixo mostra um diagrama de memória cache com mapeamento associativo.

A memória cache é identificada por um tag livre e um conjunto (set) fixo para cada endereço e cada conjunto contém uma certa quantidade de linhas. Esse mecanismo oferece um bom compromisso de custo e flexibilidade, alem de ser o mais utilizado.

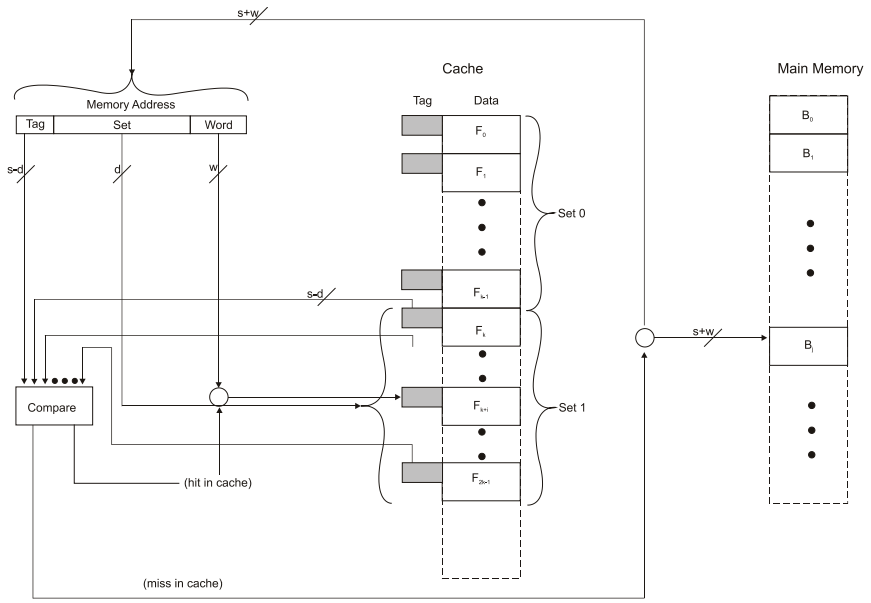


Figura 34 - Diagrama de uma memória cache com mapeamento associativo por conjunto

8. Algoritmos de substituição

No mapeamento direto, o algoritmo de substituição é apenas a ligação bloco e linha.

No mapeamento associativo, podemos atribuir várias técnicas para substituir o conteúdo da memória. Geralmente são implementados em *hardware* para obter mair velocidade e tentam identificar os blocos menos usados. Os algoritmos de substituição são:

- **FIFO (First In First Out):** substitui o bloco mais antigo.
- **LRU (Least Recently Used:** substitui pelo recentemente usado.
- **LFU (Least Frequntly Used):** substitui pelo menos usado (menos hits).
- **Randômico:** escolhe qualquer bloco.

O método randômico é o mais simples e é apenas um pouco menos eficiente do que os demais.

9. Políticas de atualização

Antes de substituir um bloco na memória cache, é necessário verificar se ele não foi alterado e se há a necessidade de atualizar a memória principal. Estatisticamente, 15% dos acessos à memória alteram o conteúdo e precisam ser atualizados.

Existem duas políticas:

- **Escrita Direta (Write-through):** todos os blocos do cache são copiados para a memória, o que gera muito tráfego e torna a escrita lenta.
- **Escrita de volta (Write-back):** bloco no cache tem uma indicação se ele sofreu alteração, e apenas esses blocos são alterados. O problema é que não há confiabilidade nos dados armazenados na memória (será que já foi alterado?).

Exemplo: Características do cache no processador Pentium 2

Como exemplo apresentamos o mecanismos de cache utilizados no processador Pentium II.

a) Mapeamento associativo por conjunto.
   - 128 conjuntos.
   - Tag de 24 bits.

b) Substituição LRU com 1 bit.

c) Política Escrita e volta (Write-back) mas pode ser configurado para Write-through.

d) MESI (Modified/Exclusive/Shared/Invalid) indica o estado da memória cache e evitar incoerências.

9.1. Memória Externa

A memória interna apresenta alta velocidade, mas tem limitação de tamanho e não mantém os dados quando o computador é desligado. Assim, torna-se necessário utilizar um sistema de armazenamento secundário ou uma memória externa.

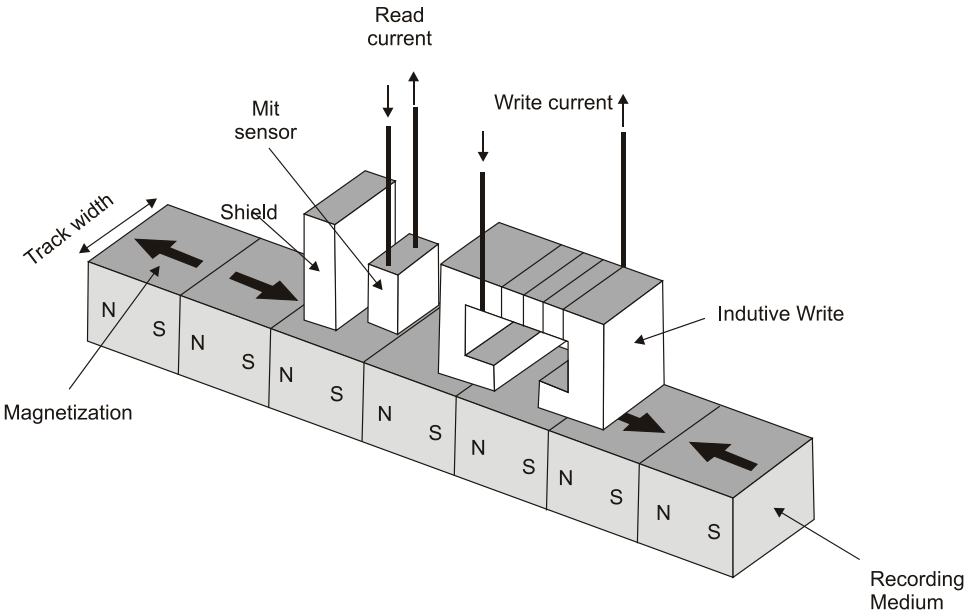


Figura 35 - Funcionamento de uma memória magnética

O principal dispositivo de memória externa é o disco magnético. O armazenamento magnético usa padrões diferentes de magnetização sobre uma superfície magnética para armazenar a informação. O armazenamento magnético é permanente, isto é, é mantido mesmo se a unidade tiver sua energia desligada. A informação é gravada usando uma ou várias cabeças de leitura/gravação, que aplicam um campo magnético sobre uma superfície magnetizável.

Uma cabeça de leitura/gravação cobre somente uma parte da superfície de modo que a cabeça ou a mídia (ou ambos) devam ser movidos um relativo ao outro, a fim alcançar os dados desejados. Em computadores modernos, o armazenamento magnético pode ter os seguintes formatos:

a) Disco magnético para o armazenamento secundário.

- Disco Rígido
- Disco flexível

b) Fita magnética, usada para o armazenamento terciário.

O armazenamento ótico, ou disco óptico típico, armazena a informação nas deformidades na superfície de um disco circular e lê essa informação iluminando a superfície com um diodo laser e observando a reflexão. O armazenamento do disco óptico é permanente. As deformidades podem ser permanentes (gravados na fabricação), dados gravados uma única vez (escreve apenas uma vez) ou regravável (mídia regravável; pode escrever várias vezes). Os formatos mais comuns são os seguintes:

- **CD, CD-ROM, DVD, BD-ROM:** gravado apenas uma vez durante a fabricação; usado para a distribuição maciça de informação digital (música, vídeo, programas)
- **CD-R, DVD-R, DVD+R BD-R:** escreve uma vez; usado para o armazenamento terciário.
- **CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE:** pode escrever várias vezes; escrita lenta, leitura rápida; usado para o armazenamento terciário

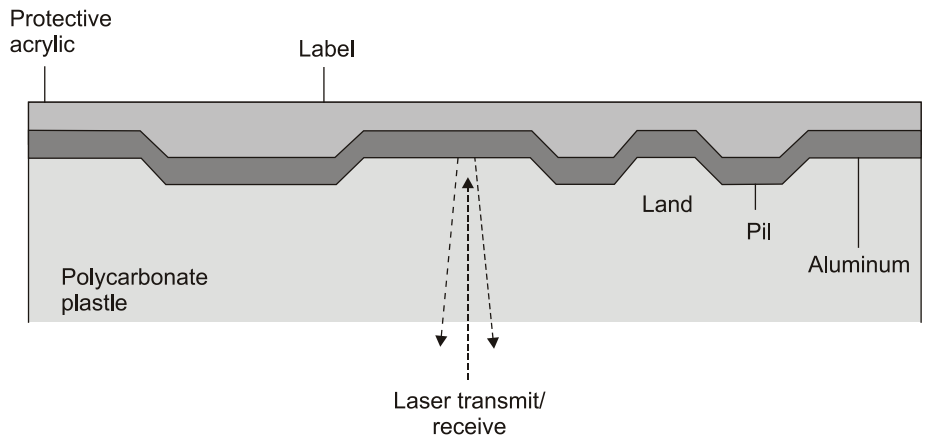


Figura 36 - Funcionamento de uma memória ótica

O disco magnetoótico é um disco ótico especial de material no qual é usado um campo magnético em sua superfície para fazer a gravação das informações. A informação é lida de forma ótica e reescrita combinando métodos magnéticos e óticos. O armazenamento de disco magnetoótico é permanente e tem acesso sequencial. Tem tempos de leitura e de escrita rápidos. É utilizado para armazenamento terciário.

9.2. Melhorando o desempenho de memórias

Algumas técnicas podem ser utilizadas para melhorar o desempenho das memórias:

- aumentar a largura de bits no lugar de profundidade.
- utilizar cache com memórias mais rápidas.
- reduzir a frequência de acesso à memória (cache).
- aumentar a capacidade de conexão com barramentos mais rápidos e barramentos hierárquicos.

Atividades de avaliação

1. Leia a especificação de um computador pessoal e relacione todos os tipos de memória que ele usa.
2. Estude o diagrama de um chip de memória DRAM e veja como ele consegue localizar um registro. Você já viu esse circuito?
3. Pesquise sobre os vários tipos de memória de massa e relacione as técnicas usadas para melhorar o desempenho para compatibilizar com as velocidades do processador.

Capítulo 5

Barramentos

Os diversos componentes de um computador precisam se comunicar. O barramento é um subsistema que transfere dados entre componentes dentro de um computador ou entre computadores.

Os barramentos do computador são basicamente linhas de comunicação elétricas em forma paralela ou serial com conexões múltiplas, isto é, permite ligar vários dispositivos. Com o aumento do desempenho, os sistemas de interconexão têm papel importante.

O barramento interno tem como função conectar a CPU, memória e interfaces. O barramento externo tem como função conectar computadores aos seus periféricos, impressoras, vídeo e outros computadores.

1. Características de um Barramento

O barramento tem como principais características:

1. estrutura de comunicação entre componentes do computador.
2. geralmente tem filosofia de difusão (*broadcast*).
3. geralmente é paralelo (vários canais).
4. pode ou não conter linhas de energia para alimentação dos perféricos.

Podemos classificar os barramentos em dois tipos:

Dedicado: as linhas de endereço e dados são separadas, isto é, têm linhas físicas exclusivas. É necessário mais linhas de barramento, mas a transferência é mais rápida.

Compartilhado: as linhas de endereço e de dados compartilham as mesmas linhas físicas. É necessário uma linha de controle para indicar o momento em que há um endereço válido ou um dado válido no barramento. A vantagem é que é necessário menos linhas de barramento, porém perde-se em desempenho, e o controle é mais complexo.

Um barramento pode ser construído de diversas formas. As mais comuns são:

- trilhas paralelas de circuito impresso.
- cabo flexível.
- conectores paralelos.

1.1. Barramentos Paralelos e Seriais

Os barramentos podem ter as linhas de informação transmitidas em paralelo ou em série. Os barramentos paralelos transportam as palavras de dados paralelamente em múltiplos fios ou trilhas, e os barramentos seriais transportam os dados em forma de sequência de bits. A capacidade de transmissão em barramentos paralelos geralmente é maior que os seriais, capazes de transmitir apenas um bit por vez.

Entretanto, quando as taxas de transmissão aumentam, os problemas de sincronismo, de consumo de energia, de interferência eletromagnética entre os fios ou trilhas nos barramentos paralelos tornam-se cada vez mais difíceis de se contornar. A solução usada para esse problema é aumentar a largura do barramento para manter a taxo do relógio mais baixa.

Um barramento serial pode realmente operar em taxas de dados mais elevadas do que um barramento paralelo. Apesar de ter poucas linhas de transmissão, um barramento em série não tem o problema de interferência entre as vias e menos problemas de sincronismo. O USB, Firewire, e Serial ATA são exemplos de barramentos seriais de alta velocidade.

1.2. Exemplos de Barramentos Paralelos

- ATA (Advanced Technology Attachment) e também ou seus variantes PATA, IDE, EIDE, ATAPI, etc., barramento paralelo para disco rígidos, óticos ou fita.
- ISA Industry Standard Architecture, barramento de placa-mãe
- ISA Estendido ou EISA, evolução do ISA
- PCI (Peripheral Component Interconection), barramento atual para placa-mãe.
- MicroChannel ou MCA, usado pela IBM
- Computer Automated Measurement and Control (CAMAC), para sistemas de instrumentação.
- Multibus para sistemas industriais.

1.3. Exemplos de Barramentos Seriais

- USB (Universal Serial Bus), usado para uma variedade de dispositivos externos
- HyperTransport, barramento serial ou paralelo para conectar componentes em uma placa-mãe, como memórias, circuitos de vídeo e processadores matemáticos.
- I^2C (Inter-Integrated Circuit), para interconectar periféricos externos a baixa velocidade como termômetros, displays, sensores, etc.
- SATA Serial ATA, versão serial do ATA para conectar discos rígidos e discos óticos.
- PCI Express ou PCIe, evolução de barramento PCI para ligar interfaces em computadores pessoais.
- SCSI (Small Computer System Interface), barramento serial para ligar discos rígidos, discos óticos e fitas magnéticas.
- EIA-485 ou RS-485, interface confiável de baixa velocidade, geralmente usada para ligar dispositivos periféricos em automação predia e ou industrial.
- Firewire ou IEEE 1394, interface serial de alta velocidade para ligar periféricos (mais rápido que o USB).

1.4. Categorias de Barramento em um Computador

Os barramento são divididos em barramentos de dados, de endereços e de controle:

Barramento de dados

O barramento de Dados tem como objetivo transferir dados e referências de endereço de memória. A largura do barramento de dados vai determinar o desempenho do computador: quanto mais largo mais dados simultâneos são transmitidos ao mesmo tempo. Geralmente, o tamanho dos barramento de dados é múltiplo de 8 bits, por exemplo: 8, 16, 32, 64.

Barramento de endereços

O barramento de Endereços tem como objetivo identificar a origem e o destino do dado. Por exemplo: a CPU lê uma instrução na memória e identifica um dispositivo. A largura do barramento de endereços vai determinar a capacidade máxima de memória. Por exemplo: a CPU 8080 tem um barramento de endereços de 16 bits de largura, possibilitando o endereçamento de 64 Kbytes de memória.

Barramento de controle

O barramento de Controle tem como objetivo enviar dados de controle entre os subsistemas. Por exemplo: quando a CPU vai escrever um dado na memória, ela coloca o endereço da memória no barramento de endereço, coloca o dado que vai escrever no barramento de dados e envia um sinal de escrita na memória via barramento de controle. O tamanho do barramento de controle irá depender da arquitetura do computador e da CPU. Exemplos de sinais de controle são: leitura e escrita de memória, leitura e escrita de E/S, interrupção, relógio etc.

1.5. Barramento Interno e Externo

A maioria dos computadores tem barramentos internos e externos. Um barramento interno conecta todos os componentes internos de um computador na placa-mãe (onde se encontra o processador e a memória interna). Esses tipos de barramentos são também chamados de *barramento local*, porque pretendem conectar os dispositivos locais, e não outras máquinas externas ao computador. Um barramento externo conecta periféricos externos à placa-mãe do computador. Um barramento interno é sempre mais rápido que um barramento externo.

Conexões de rede tais como o Ethernet não são consideradas geralmente como barramentos devido ao fato de ligar computadores a longa distância, embora a diferença tecnológica não seja perceptível. As definições de barramento interno e externo são, às vezes, distorcidos. Por exemplo: o I^2C pode ser usado como um barramento interno ou externo; o InfiniBand foi criado para substituir os barramentos internos PCI, mas é utilizado para ligar equipamentos em rede.

1.6. Funções dos barramentos

A funções fornecidas pelo barramento para cada categoria de componentes são:

Memória

- Recebe e envia dados;
- Recebe endereços;
- Recebe sinais de controle (relógio, leitura, escrita).

Entrada e Saída (E/S)

- Recebe e envia dados CPU/periféricos;
- Recebe sinais de controle da CPU;

- Envia sinais de controle para periféricos;
- Recebe endereço dispositivo (endereço da porta que identifica o periférico);
- Envia sinal de interrupção.

Unidade Central de Processamento (UCP)

- Recebe instrução e dados;
- Escreve dados;
- Envia sinais de controle;
- Recebe e trata interrupções.

2. Hierarquia de Barramentos

Para melhorar o desempenho de um computador, podemos construir vários barramentos com velocidades diferentes.

2.1. Barramento único

É a estrutura mais simples, onde todos os dispositivos compartilham o mesmo barramento. Um exemplo de barramento único é mostrado na figura abaixo:

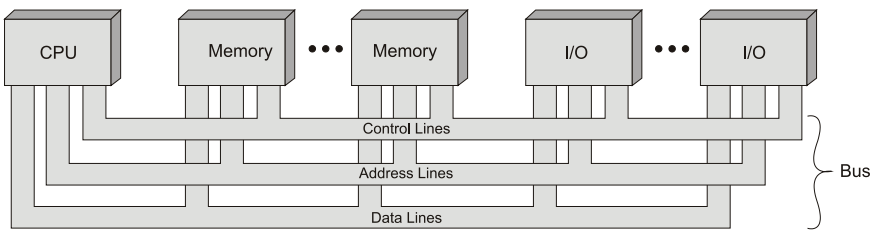


Figura 37 - Diagrama de um barramento único

O barramento único apresenta alguns problemas que impactam no desempenho do sistema. Como diversos dispositivos com velocidades diferentes compartilham o mesmo barramento os dispositivos mais rápidos (por exemplo, memória) precisam esperar a transferência dos dispositivos mais lentos (por exemplo, dispositivos de E/S).

2.2. Barramentos múltiplos

A solução para compatibilizar componentes com velocidades diferentes é construir vários barramentos agrupando os componentes semelhantes. Mostramos, na figura a seguir, um barramento múltiplo onde podemos ver os seguintes barramentos:

- **Barramento Local** Interliga a CPU com a memória cache.
- **Barramento do Sistema** Interliga a memória cache com a memória principal.
- **Barramento de Alta Velocidade** Interliga dispositivos de E/S de alta velocidade.
- **Barramento de Expansão** Interliga dispositivos de E/S de baixa velocidade.

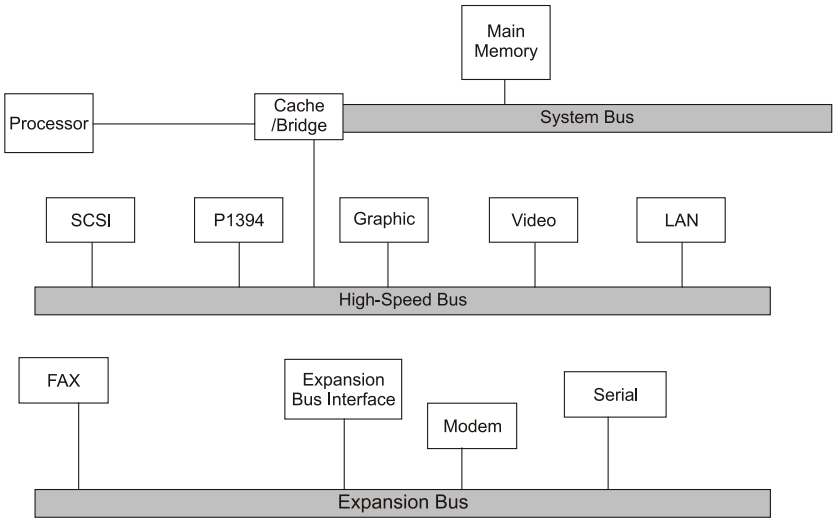


Figura 38 - Diagrama de um barramento múltiplo

A conexão entre os diversos barramentos é realizado pelas interfaces que armazenam a informação enquanto aguardam o barramento mais lento, liberando rapidamente o barramento de maior velocidade.

3. Arbitragem de barramento

A arbitragem é necessária quando vários módulos podem requisitar o barramento para seu uso, por exemplo, quando usamos DMA. É necessário garantir que apenas um módulo utilize o barramento.

3.1. Arbitragem centralizada

Nesse caso apenas um dispositivo controla o acesso ao barramento. Todos os módulos solicitam o acesso ao barramento a ele. Esse dispositivo pode estar dentro da CPU ou fora dela.

3.2. Arbitragem distribuída

Cada módulo tem seu próprio dispositivo de arbitragem que interage com os dispositivos dos demais módulos. Existe uma lógica de controle em cada módulo.

4. Temporização de Barramento

A temporização coordena os eventos no barramento.

4.1. Síncrona

Todos os eventos são sincronizados pelo sinal de relógio. O barramento de controle inclui uma linha de relógio para garantir a sincronia entre os diversos componentes.

Todos os dispositivos precisam ler esse relógio e, geralmente, temos um evento para cada ciclo de relógio. Esse circuito é mais simples, menos flexível e que apresenta um desempenho pior pois os ciclos são definidos pelo ciclo dos dispositivos mais lentos.

4.2. Assíncrona

Cada evento tem o seu próprio sinal de controle. Por exemplo: a CPU envia sinal para a memória indicando a disponibilidade de um endereço no barramento, e a memória envia sinal para a CPU indicando a disponibilidade do dado no barramento. O circuito é mais complexo, porém há um aproveitamento melhor dos tempos variáveis dos dispositivos.

5. Barramento PCI

O barramento PCI (Peripheral Component Interconection) foi criado em 1990 e é o padrão mais utilizado nos computadores PC. Ele foi desenvolvido pela Intel, mas foi liberado para domínio público, o que propiciou sua grande difusão. O padrão definiu 49 linhas de sinal para barramento de 32 bits. Existe também a versão para barramento de 64 bits.

Linhas obrigatórias:

- Sistema Relógio e reset.
- Endereço e Dado 32 ou 64 bits multiplexados.
- Controle de Interface Sinal do mestre indicando o início e fim da transação e seleção de dispositivo.
- Arbitragem Linhas não compartilhadas (cada dispositivo tem um par de controle REQ/GNT ligado ao árbitro.
- Erros Sinais de erros de paridade e de sistema.

Linhas opcionais:

- **Linhas de Interrupção:** Linhas não compartilhadas e limitadas a 4.
- **Controle de cache:** Utilizado para controlar o cache de memória.

- **Extensão do barramento:** Aumenta a largura do barramento de dados para 64 bits e as respectivas linhas de arbitragem.
- **Linhas de teste:** Pinos utilizados para testar as interfaces (JTAG).

5.1. Temporização

A figura abaixo mostra uma operação de leitura em um barramento PCI. Uma vez que o mestre tenha obtido o controle do barramento, inicia a transferência ativando o sinal FRAME, que permanece assim até o final da transferência. O iniciador coloca o endereço no barramento AD, que é identificado no início do segundo ciclo de relógio. O mestre envia o sinal IRDY para selecionar o dispositivo alvo que confirma a seleção através do sinal DEVSEL.

Após alguns ciclos de espera, o sinal C/BE muda de estado indicando a presença de dado no barramento AD. Através do sinal TRDY, o mestre indica a presença de um dado válido no barramento. A mudança do TRDY indica a disponibilidade dos dados seguintes no barramento. O final da tranferência é indicado quando os sinais FRAME, IRDY, TRDY são desativados e, em seguida, o sinal DEVSEL.

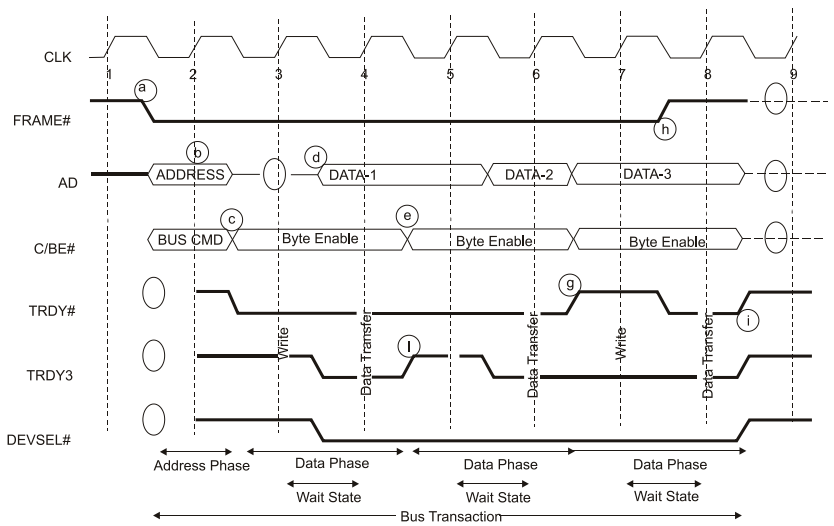


Figura 39 - Temporização do barramento PCI

5.2. Arbitragem

O barramento PCI utiliza arbitragem centralizada, na qual cada dispositivo tem seu par de sinais de controle REQ/GNT, conforme mostrado na figura a seguir. O dispositivo que deseja o controle do barramento ativa o sinal REQ e o árbitro PCI autoriza o acesso com o sinal GNT.

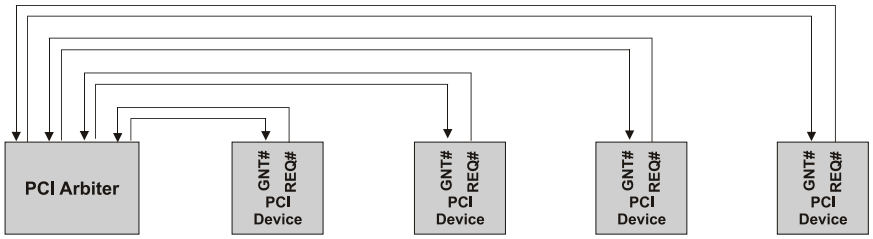


Figura 40 - Arbitragem do barramento PCI

Atividades de avaliação

1. Discuta sobre vantagens e desvantagens dos barramentos seriais e paralelos.
2. Pesquise sobre a evolução da velocidade e da largura de barramentos internos do microcomputador PC.
3. Pesquise como pode ser implementado um árbitro de um barramento.

Capítulo 6

Entrada e saída

Um computador precisa se comunicar com o mundo exterior. Ele é constituído por vários dispositivos que apresentam quantidades de dados, velocidades e formatos diferentes. Os dispositivos de entrada e de saída são mais lentos do que a CPU e a memória.

Na computação, a entrada/saída, ou E/S, ou ainda o termo em inglês I/O (Input/Output), referem-se à comunicação entre um sistema de processamento de dados (tal como um computador) e o mundo exterior - possivelmente um ser humano ou outro sistema de processamento de dados. As entradas são os sinais ou os dados recebidos pelo sistema, e as saídas são os sinais ou os dados emitidos a partir dele.

Podemos também usar termo "executar E/S" significando executar uma operação da entrada ou de saída. Os dispositivos de E/S são usados por uma pessoa (ou por outro sistema) para comunicar-se com um computador. Por exemplo: um teclado ou um *mouse* pode ser um dispositivo de entrada para o computador, enquanto monitores e as impressoras são considerados dispositivos de saída de um computador. Os dispositivos para comunicação entre computadores, tais como modem e cartões de rede, são, tipicamente, de entrada e saída.

Podemos relacionar os tipos de dispositivos externos:

- **Comunicação humana:** teclado, display, impressora
- **Comunicação com máquina:** disco, fita
- **Comunicação remota:** interface de rede, modem

1. Módulo de Entrada e Saída

O componente que estabelece a ligação entre o barramento de alta velocidade (onde estão CPU e memória) com os dispositivos de entrada e saída (impressora, teclado, modem, etc) é o módulo de entrada e saída.

1.1. Funções do Módulo de E/S

- Controle de temporização
- Comunicação com a CPU
- Comunicação com o dispositivo
- Armazenamento de dados
- Detecção de erros

1.2. Etapas de E/S

a) CPU verifica estado do dispositivo de E/S

b) Módulo de E/S retorna estado

c) Se OK, CPU requisita transferência de dados

d) Módulo de E/S pega dados do dispositivo

e) Módulo E/S transfere dados para CPU

1.3. Módulo de E/S

Apresentamos, na figura abaixo, o diagrama de um módulo de entrada e saída. O módulo de E/S deixa o controle do dispositivo transparente para a CPU e pode suportar mais de um dispositivo.

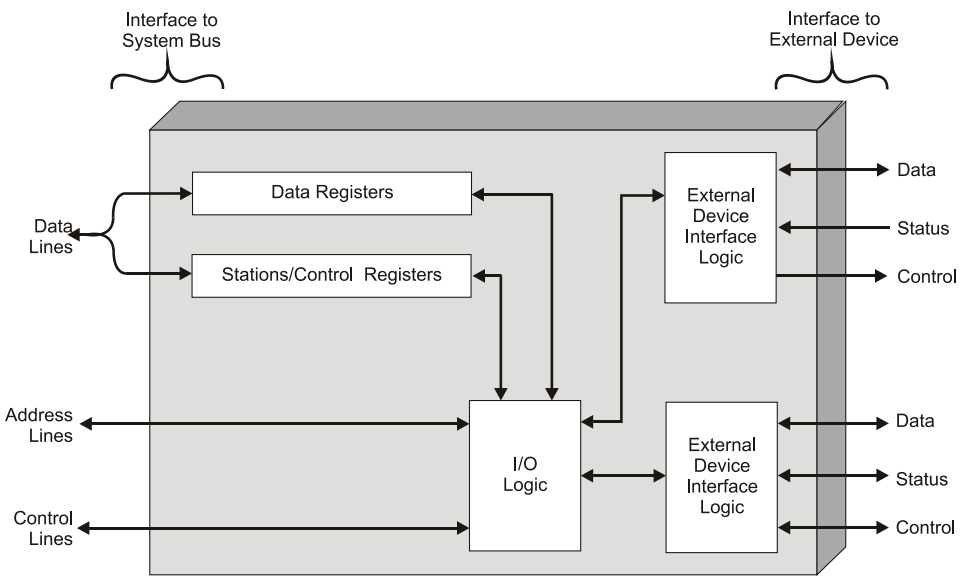


Figura 41 - Diagrama de blocos de um módulo de E/S

1.4. Técnicas de Entrada e Saída

As técnicas de entrada e de saída são:

a) entrada e saída programada

b) entrada e saída por interrupção

c) entrada e saída por DMA

1.5. Entrada e Saída Programada

A CPU tem controle direto sobre o módulo de E/S: determina o status, escreve/lê comandos e transfere dados. A CPU aguarda o módulo terminar a operação e, como os dispositivos são mais lentos que a CPU, há um desperdício de tempo. Mostramos a seguir o seu funcionamento:

a) CPU requisita operação de E/S

b) Módulo E/S executa operação

c) Módulo E/S marca indicador de estado (flag)

d) CPU verifica o indicador de estado periodicamente

e) Módulo de E/S não informa CPU diretamente

f) Módulo E/S não interrompe a CPU

g) CPU deve esperar pela conclusão da requisição

O endereçamento de E/S é semelhante ao endereçamento de memória, e cada dispositivo tem um endereço único. O comando da CPU contém o identificador (endereço).

Podemos realizar dois tipos de mapeamento de E/S:

- **Memória mapeada:** os dispositivos de E/S e de memória compartilham o mesmo espaço de endereços. Operações de E/S são semelhantes à leitura/escrita em memória. Não há comando especializado para operações de E/S.
- **E/S isolado:** o espaço de endereçamento é separado da memória. É necessário comandos especilizados para E/S. Geralmente, o tamanho dos endereços de E/S são menores do que os endereços de memória.

1.6. Entrada e Saída por Interrupção

A vantagem do controle por interrupção é que a CPU não espera pela resposta do dispositivo. A CPU também não precisa testar o estado do dispositivo, pois o módulo de E/S interrompe a CPU quando o dado estiver pronto. Mostramos, a seguir, o seu funcionamento:

a) CPU executa comando de leitura no módulo de E/S

b) CPU retorna ao processamento normal

c) Módulo E/S lê dado do dispositivo

d) Módulo E/S interrompe CPU

e) CPU requisita dados

f) Módulo E/S transfere dados

A CPU verifica a ocorrência de interrupção após cada instrução. Se houver interrupção, salva contexto (registradores) e processa interrupção.

Se em um computador existem vários módulos de E/S (o que é normal), é necessário um procedimento para identificar o módulo que interrompeu a CPU.

Linha diferente para cada módulo O limite é a capacidade de linhas de interrupção da CPU.

a) **Pool de software:** CPU recebe apenas uma interrupção. Ao recebê-la, pesquisa todos os módulos para descobrir. Método lento.

b) **Interrupção vetorado (Daisy-chain):** A CPU tem apenas uma linha de interrupção. Ao receber a interrupção, a CPU recebe um reconhecimento com uma cadeia de caracteres indicando o dispositivo solicitante.

c) **Arbitração de barramento (Bus master):** Módulo que deseja interromper obtem o controle do barramento, a CPU reconhece o módulo e coloca o vetor na linha de dados.

Cada interrupção tem sua linha e, quando estiver tratando uma interrupção e receber outra solicitação, o computador pode ter duas atitudes:

a) nova interrupção fica aguardando até terminar o tratamento da primeira interrupção.

b) ao receber uma nova interrupção que tenha maior prioridade, suspende a execução do programa e trata a interrupção. Se não (prioridade menor), aguarda a conclusão da interrupção corrente.

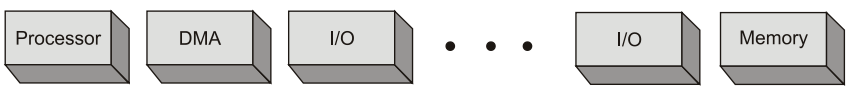
1.7. Exemplo: Processador x86

Os processadores da linha x86 têm apenas uma linha de interrupção. Para aumentar essa capacidade, é utilizado o controlador de interrupção 8259, que oferece 8 linhas de interrupção com prioridade. Ao receber uma solicitação de interrupção, o 8259 envia o sinal de interrupção para o processador (INTR) e, após receber as confirmação (INTA), envia o vetor da interrupção recebida. O processador, então, passa a tratar a interrupção diretamente com o módulo de E/S.

1.8. Entrada e Saída por DMA (Direct Memory Access)

Acesso direto à memória (DMA - *Direct Memory Access*) é uma técnica utilizada nos computadores e nos microprocessadores modernos que permite que certos subsistemas de *hardware* do computador possam transferir dados diretamente para a memória RAM, independentemente da unidade central de processamento. Muitos sistemas de hardware utilizam DMA, como controladores de unidade de disco, placas gráficas, placas de rede e placas de som.

O DMA também é usado para transferência de dados entre processadores multi-core, onde seu elemento de processamento é equipado com uma memória local (memória cache) e para transferir dados entre a memória local e a memória principal desses sistemas. Computadores equipados com DMA podem transferir dados de e para dispositivos muito mais rapidamente e com muito menos sobrecarga de CPU. Do mesmo modo, um elemento de processamento dentro de um processador multi-core podem transferir dados de e para a sua memória local sem ocupar o seu tempo de processamento e permitindo, simultaneamente, a computação e a transferência de dados.



(a) Single-bus, detached DMA

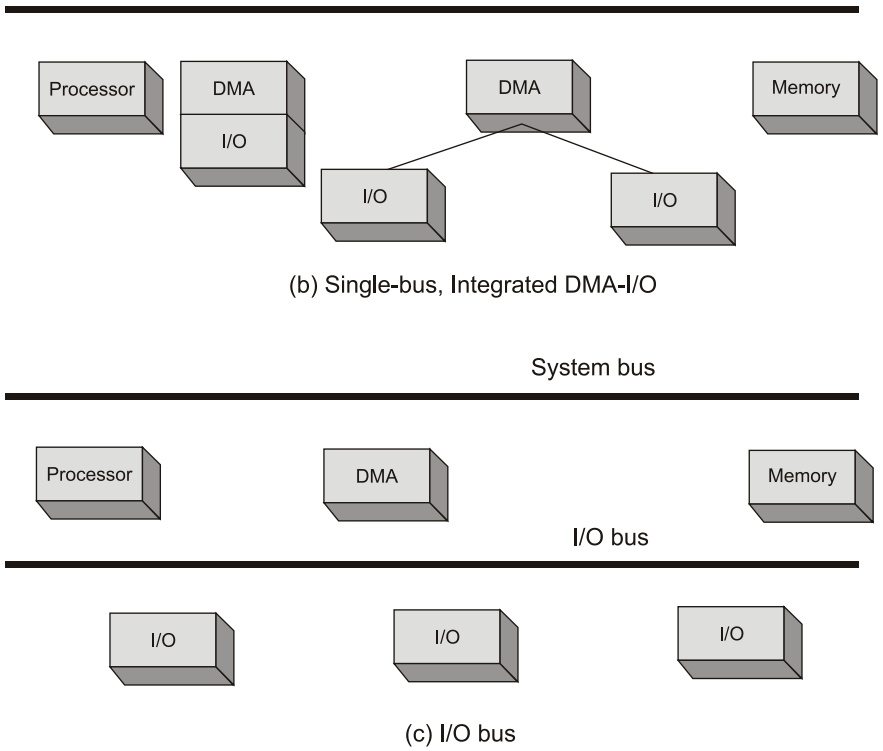


Figura 42 - Configurações de DMA

Síntese do capítulo

Neste capítulo apresentamos detalhadamente todos os componentes que fazem parte de uma arquitetura de computador. Iniciamos apresentando a Unidade Central de Processamento, "o cérebro" do computador, responsável pelo controle do computador além de realizar as operações aritméticas. Em seguida, apresentamos as memórias, tanto internas semicondutoras quanto as externas de alta capacidade.

Apresentamos, depois, os diversos barramentos de um computador. Finalmente, mostramos as interfaces que interagem com os dispositivos de entrada e de saída, isto é, interfaces que interagem com os dispositivos que o ser humano utiliza.

Atividades de avaliação

1. Relacione os principais problemas de uma interface de entrada e de saída e discutas as soluções para melhorar o desempenho.
2. Pesquise sobre a evolução da velocidade das interfaces de entrada e saída de um microcomputador PC.
3. Projete um circuito simples de controle de entrada e saída de um computador (dica: transfira dados da memória para a interface e vice-versa).

Leituras, filmes e sites

Leituras

VELLOSO, Fernando. **Informática:** Conceitos básicos. 9. Ed. Rio de Janeiro: Elsevier, 2014.

FORMICE, Cesar Renato, GERALDI, Luciana Maura Aquaroni e GALASSI, Carla Renata. **História da Era da Informática.** São Paulo: AgBook, 2013.

ROLF, Enderlein. **Microeletrônica:** uma introdução ao universo dos microchips, seu funcionamento, fabricação e aplicações. São Paulo: Editora da Universidade de São Paulo, 1994.

Filmes

Rapida história da evolução dos microprocessadores. Duração: 05min 29s. O vídeo apresenta a evolução dos microprocessadores desde o 4004 aos sonhados processadores quantico e de dna. Acesso em: https://www.youtube.com/watch?v=8MY5gSrCr4E

Memória Cache. Duração: 31min 47s. O vídeo aborda definições e funções de Memória Cache. Acesso em: https://www.youtube.com/watch?v=Xt9KjdOdR2w

Barramentos. Duração: 31min 34s. O vídeo discute sobre barramentos. Acesso em: https://www.youtube.com/watch?v=oUlH9t_8kaE

Sites

Apostila Arquitetura de Computadores B. Piropo (em Português)

http://www.bpiropo.com.br/arqcom1.htm

Informações gerais sobre Arquiteturas de Computadores (em Português)

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

Tutorial sobre Memória Cache (em Inglês)

http://www.cs.iastate.edu/~prabhu/Tutorial/CACHE/mem_title.html

Referências

STALLINGS, W. **Arquitetura e Organização de Computadores** 5ª Ed. Editora: Prentice Hall, 2002.

PATTERSON, D. A.& HENNESSY, J. L. **Organização e Projeto de Computadores** 3ª Ed. Editora: Campus, 2005.

HENNESSY, J. L. & PATTERSON, D. A. **Arquitetura de Computadores:** Uma Abordagem Quantitativa 3ª Ed. Editora: Campus, 2003.

MURDOCCA, M. J. & HEURING, V. P. **Introdução à Arquitetura de Computadores** 1ª Ed. Editora: Campus, 2000.

WEBER, R. F. **Arquitetura de Computadores Pessoais,** Vol 6. 2ª Ed. Editora: Bookman, 2008.

WEBER, R. F. **Fundamentos de Arquitetura de Computadores,** Vol 8. 3ª Ed. Editora: Bookman, 2008.

Parte 4

Software em um computador

Capítulo 7

Conjunto de Instruções

Objetivos

Neste capítulo vamos apresentar a interação do *software* com uma arquitetura de computador. Inicialmente, vamos apresentar os conceitos de conjunto de instruções e como as instruções executam o processamento em um computador. Em seguida, apresentamos os conceitos básicos de um sistema operacional e sua interação com a arquitetura do computador.

Introdução

Definimos Conjunto de Instruções (ou em inglês, *instruction set*) são as instruções de um processador que realizam operações em um computador, microprocessador ou microcontrolador. O conjunto de instrução oferece ao programador uma representação em mnemônicos do código de máquina para facilitar o entendimento e o uso pelo programador. O programador irá processar o programa escrito em nessa linguagem, que é específico apenas para um determinado processador.

Cada processador possui o seu próprio conjunto de instruções, fornecido pelo fabricante, que também costuma disponibilizar um compilador *assembler,* que transforma o conjunto de instruções em código de máquina para ser utilizado por esse processador ou família de processadores.

Assim como exixtem dois tipos de processadores, se o conjunto de instruções for reduzido, chamamos de RISC e, se o conjunto de instruções for complexo chamamos de CISC.

1. Ciclo de Instrução

Definimos ciclo de instrução o período de tempo no qual um computador lê uma instrução em linguagem de máquina da sua memória e processa a sequência de ações que o processador deve realiza para executar essa instrução em código de máquina num programa. Por essa razão, o ciclo de instrução também é chamado de **ciclo de busca e execução** ou **ciclo busca-execução**.

A expressão "ciclo de busca e execução" é a melhor opção de nomenclatura, pois descreve em essência do funcionamento do computador. A instrução deve ser buscada na memória principal e, depois, executada pelo processador. A figura abaixo mostra um diagrama do funcionamento do ciclo de instrução ou ciclo de busca e execução.

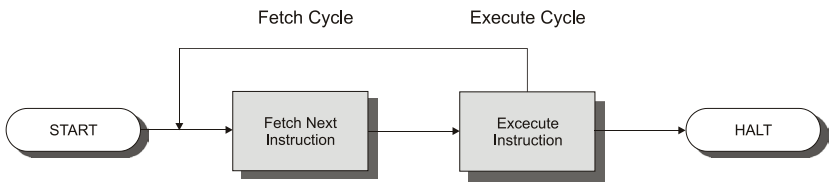


Figura 43 - Diagrama do ciclo de instruções

1.1. Busca de instrução

a) Contador de Programa (PC) guarda endereço da próxima instrução.

b) Processador busca instrução na posição de memória apontada pelo PC.

c) Incrementa PC.

d) Instrução é carregada no Registrador de Instrução (IR).

e) Processador interpreta instrução e executa ação.

1.2. Executa instrução

- Processador Memória Tranfere dado da CPU para memória.
- Processaror-E/S Tranfere dado da CPU para E/S.
- Processamento de dado-Aritmética e lógica.
- Controle altera sequência de execução (jump).
- Vários combinações dos comandos anteriores.

Apresentamos na figura abaixo um diagrama de estados do cliclo de instruções.

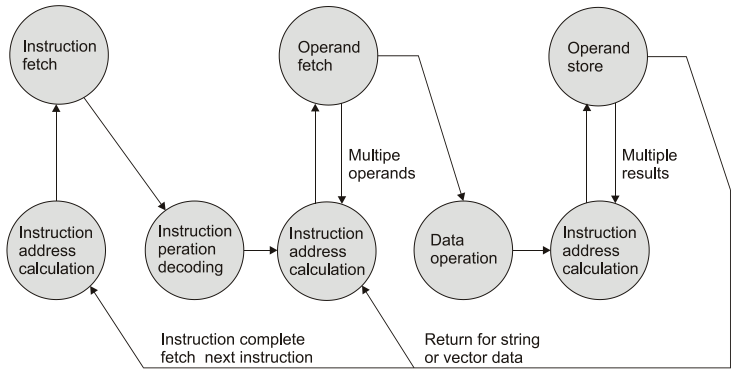


Figura 44 - Diagrama de estados do ciclo de instruções

1.3. Exemplo de execução de instrução

Mostramos na figura abaixo um exemplo de execução de programa. Este trecho de programa realiza a soma do conteúdo do endereço de memória 940 ao conteúdo do endereço de memória 941 e armazena o resultado nesse último endereço. São executadas três instruções em três ciclos de busca e execução.

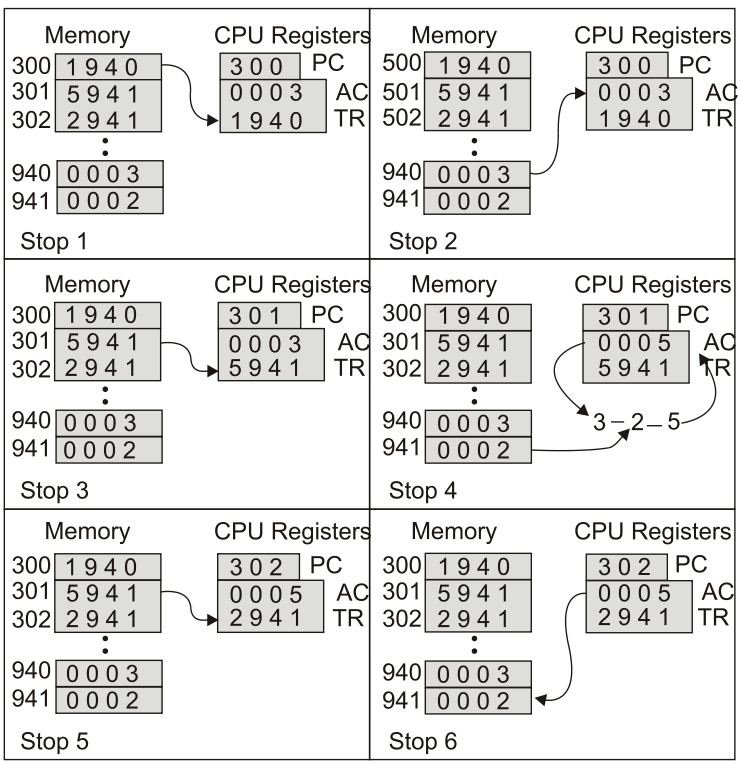


Figura 45 - Exemplo de execução de um programa

a) O conteúdo do PC é 300, endereço da primeira instrução. A instrução é carregada no Registrador de Instrução (IR).

b) Os quatro primeiros bits do IR indicam a operação da instrução. O restante da instrução (12 bits) contém o endereço 940, local onde está o operando da operação (0003) é compilado para o acumulador (AC).

c) O PC é incrementado e a próxima instrução é buscada.

d) O conteúdo do AC é somado com o conteúdo do endereço de memória 941 (0002). O resultado é armazenado no AC.

e) O PC é incrementado, e a próxima instrução é buscada.

f) O conteúdo do AC é armazenado na posição de memória 941.

2. Linguagem de Máquina

Uma linguagem de máquina é o conjunto de instruções específicas para um determinado processador. Para uma arquitetura de processador, uma instrução dada pode ser especificada com os seguintes parâmetros:

- registradores para cálculo aritmético, endereçamento ou funções de controle;
- posições ou deslocamento (*offset*) de memória;
- modalidades de endereçamento usadas para interpretar os operandos.

As operações mais complexas podem ser especificadas combinando várias instruções simples, que, em uma arquitetura von Neumann, são executados sequencialmente ou desviadas seguindo o fluxo de controle das instruções.

Algumas operações disponíveis na maioria dos conjuntos de instrução:

- **Movimentação**

a) Atribui a um registrador (uma posição de memória de rascunho para o processador) um valor constante fixo.

b) Move um dado de uma posição de memória para um registrador ou vice-versa. Isto é feito para guardar os dados para executar uma operação mais tarde ou para armazenar o resultado de uma computação.

c) Ler ou escrever dados nos dispositivos de entrada e saída.

- **Calculando**

a) Soma, subtrai, multiplica, ou divide os valores de dois registradores, colocando o resultado em um registrador.

b) Executa operações lógicas nos bits, executando as funções E, OU de bits correspondentes em um par de registradores e a negação (NÃO) de cada bit em um registrador.

c) Compara dois valores nos registradores (por exemplo, para considerar se um é maior, menor ou se são iguais).

- **Definindo o Fluxo do programa**

a) Faz um salto na execução de um programa de uma outra posição e executa as instruções nessa posição.

b) Faz um salto na execução de um programa para outra posição se uma determinada condição acontecer (por exemplo, um valor de registrador).

c) Faz um salto na execução de um programa para outra posição, mas guarda a posição da instrução seguinte como um ponto para retornar posteriormente (para executar uma sub-rotina).

Alguns computadores incluem instruções complexas em seu conjunto de instruções. Uma única instrução complexa desempenha funções que podem tomar muitas instruções em outros computadores. Tais instruções, classificadas como instruções que executam múltiplas etapas, são constituídas por várias instruções simples. Alguns exemplos de instruções complexas são:

- Salvar muitos registradores simultaneamente.
- Mover grandes blocos de memória
- Realizar aritmética complexa ou de ponto flutuante (por exemplo, seno, cosseno, raiz quadrada, etc.).
- Executar instruções que combinam operações lógicas e aritméticas com um operando na memória e outro no registrador ou entre operandos na memória.

Um tipo especial de instrução complexa é o SIMD (*Single-Instruction Stream Multiple-Data Stream*), que realiza operações em vetores ou uma operação que execute a mesma operação aritmética em múltiplas partes de dados ao mesmo tempo. SIMD têm a habilidade de manipular grandes vetores e matrizes no tempo mínimo, permitindo a paralelização de algoritmos que tratam de sons, imagens e processamento de vídeo. Tecnologias comerciais como MMX, 3DNow! e AltiVec são exemplos de instruções SIMD.

O projeto de um conjunto de instruções é uma tarefa complexa. Na história do desenvolvimento dos computadores, duas filosofias de conjunto de instrução predominaram: o CISC e o RISC.

O primeiro foi o CISC (*Complex Instruction Set Computer*), ou Computador com Conjunto de Instruções Complexas, que implementa instruções complexas que realizam muitas funções, reduzindo o tamanho dos programas em código de máquina.

Nos anos 70, entretanto, foram realizadas algumas pesquisas que decidiram que muitas instruções do conjunto de instruções poderiam ser eliminadas. O resultado dessa pesquisa foi o computador RISC (*Reduced Instruction*

Set Computer), ou Computador com Conjunto de Instruções Reduzido, uma arquitetura que usa um conjunto menor de instruções.

Um conjunto de instrução mais simples pode oferecer velocidades de processamento mais elevadas, o tamanho do processador reduzido e o consumo de potência menor. Mesmo considerando que, para realizar um mesmo processamento, um computador RISC precisa de mais instruções, o aumento da velocidade devido à sua simplicidade compensa. Entretanto, um conjunto de instruções mais complexo pode melhorar o desempenho de operações matemáticas, melhora a eficiência do cache de memória e simplificar a programação.

3. Implementação

Um conjunto de instrução pode ser implementado de várias maneiras, isto é, as formas de se implementar um mnemônico pode ser diferente para cada arquitetura. Apesar de haver várias maneiras de implementar um conjunto de instrução, todas elas podem executar os mesmos códigos binários. As várias maneiras de implementar um conjunto de instrução dão diferentes relações custo-benefício, desempenho, consumo de energia, tamanho, etc.

Ao projetar microarquiteturas, os engenheiro usam blocos prontos de circuitos eletrônicos (projetados, geralmente, de forma isolada), como somadores, multiplexadores, contadores, registradores, ULAs, etc. Algum tipo da linguagem de transferência dos registradores é usada para descrever a decodificação e definir a sequência de microinstrução para uma determinada arquitetura física. Há duas maneiras básicas de construir uma unidade de controle para executar uma instrução (embora muitos projetos usem formas intermediárias desse dois modos):

a) projetos dos primeiros computadores e de alguns computadores RISC, em que o conjunto instrução completo é decodificado e sequenciado conforme a arquitetura.

b) projetos de computadores que empregam rotinas e/ou tabelas do microcódigo escrito em ROM e/ou PLAs. Cada instrução é transformada em um conjunto de microinstruções que são, então, executadas.

Recentemente, existem alguns projetos de processador que compilam o conjunto de instrução em uma RAM ou em umamemória FLASH dentro do processador central. Como exemplo, temos os processadores Rekursiv, Imsys e os *soft processors* em FPGA.

Um conjunto de instruções pode ser emulado em *software* através de um interpretador. Naturalmente, devido à interpretação dinâmica, a execução de programas em processador emulado é mais lenta do que a de programas executados diretamente no próprio processador. Atualmente, é comum um fabricante de processador disponibilizar emuladores em *software* aos programadores de forma que eles possam desenvolver e testar programas antes que o *hardware* esteja pronto.

Frequentemente, os detalhes de implementação têm forte influência na seleção de instruções para um determinado conjunto de instrução. Por exemplo: muitas implementações de pipeline de instrução permitem apenas uma única leitura de memória ou escrita de memória para cada instrução, conduzindo à redefinição do conjunto de instruções. Para outros casos, é necessário incluir um atraso para que o ciclo de instrução possa ser compatível com as demais instruções.

As demandas de desempenho dos Processadores de Sinal Digital (DSP – *Digital Signal Processor*) direcionam o desenvolvimento de novas arquiteturas e dos novos conjunto de instruções. Por exemplo: a fim aumentar o desempenho de filtros digitais, a instrução MAC (*Multiplicar e Acumular*) em um processador DSP típico deve ser executada usando uma arquitetura Harvard que possa buscar uma instrução e duas palavras de dados, simultaneamente, em apenas um ciclo.

4. Número de Operandos

Os conjuntos de instrução podem ser categorizados pelo número máximo de operandos (registradores, posições de memória ou valores constantes) para uma determinada instrução. Alguns dos operandos podem ser dados implícitos, armazenados em uma pilha ou em um registrador implícito. Esse número é, consequentemente, diferente do que o arity das operações.

Apresentamos a seguir as diversas formas de organizar operandos em um conjunto de instruções. Nos exemplos, ma, mb e mc referem-se endereços de memória, e reg1, reg2 referem-se a registradores do processador.

- 0 operandos: são instruções que se referem a posições fixas (pilha) ou operações pré-definidas. Por exemplo: push; pop; add.
- 1 operando: cada instrução executa sua operação usando um único operando especificado. O registrador de acumulador é implícito. Por exemplo: load ma; add mb; add reg1; store ma.
- 2 operandos: caso da maioria das instruções envolvendo dois operandos, geralmente um de origem e outro de destino. Por exemplo: load ma, reg1; load mb, reg2; add reg1,reg2;

- 3 operandos: instruções mais complexas, geralmente em máquinas CISC. Podemos combinar em apenas uma instrução a leitura e operação. Por exemplo: add reg1, mb, mc (soma reg1, mb e mc e coloca resultado em reg1).
- mais de 3 operandos: instruções usadas em algumas máquinas CISC que permitem operações de matrizes e vetores. Por exemplo: mvector ma, mb, mc, 6 (multiplica vetor na posição mb e mc com 6 valores e escreve na memória ma).

Atividades de avaliação

1. Pesquise sobre o conjunto de instruções de um processador comercial. Leia o manual e faça um resumo das principais categorias de instruções.
2. Suponha um processador com 3 registradores (A, B e C) e que realize operação de soma e subtração. Observando o conjunto de instruções de outros processadores, crie a seu próprio conjunto de instrução considerando que este processador só faz desvio com acumulador (registrador A) igual ou diferente de zero. Defina instruções de leitura/escrita em registrador, memória e constante, operações matemáticas, desvio de fluxo.

Capítulo 8

Sistema Operacional

Um sistema operacional (SO) é um programa básico que estabelece a relação entre o *hardware* e o usuário, e é responsável pela gerência e pela coordenação das atividades de compartilhamento dos recursos do computador que atua como um hospedeiro para aplicações para o usuário. Uma das finalidades de um sistema operacional é assegurar o correto funcionamento dos aplicativos em relação ao *hardware*. Isso alivia programas aplicativos de ter que controlar detalhes de uso do *hardware* e compartilhamento de recurso entre múltiplas aplicações, facilitando a escrita de aplicações.

Quase todos os computadores (desde supercomputadores, computadores de mesa, telefones celulares, videogames), eletrodoméstico (máquinas de lavar louça, máquinas de lavar roupa) e reprodutores de mídia (DVD, VCR) usam um sistema operacional de algum tipo. As funções são implementadas nos diversos sistemas operacionais de acordo com as aplicações específicas de cada equipamento.

Apesar de Sistema Operacional ser um assunto extenso e que não é objetivo desse curso, mostraremos aqui apenas a relação entre um sistema operacional com a arquitetura do computador.

1. Serviços de um Sistema Operacional

Os sistemas operacionais oferecem um número de serviços aos programas aplicativos e aos usuários. As aplicações acessam esses serviços com as Interface de Programação de Aplicativos (API - *Application Programming Interface*) ou chamadas de sistemas (system call). Invocando essas interfaces, a aplicação pode solicitar um serviço ao sistema operacional, passar parâmetros e receber os resultados da operação.

Os usuários podem também interagir com o sistema operacional com algum tipo da interface do *software* (SUI *Software User Interface*) como comandos usando a linha de comando (CLI *Command Line Interface*) ou usando uma interface gráfica (GUI - *Graphical User Interface*). Para computadores

pessoais, a interface é geralmente parte do sistema operacional (por exemplo, MS-Windows), mas, em sistemas operacionais multiusuários de grande porte, a interface é, geralmente, um programa aplicativo fora do sistema operacional, permitindo várias interfaces de interação (por exemplo, Unix e suas diversas interfaces, sh, bash, csh, etc).

2. Interação do Sistema Operacional com o Hardware

Mostramos a seguir as principais funções de um Sistema Operacional que fazem a interação com o *hardware*.

2.1. Execução de programa

O Sistema Operacional (SO) faz a mediação entre uma aplicação e o *hardware* do computador. Ele também permite ao usuário executar funções de gerenciamento e de manutenção do sistema computacional. Executar um programa envolve a criação de um processo pelo Sistema Operacional que o inicia atribuindo um espaço de memória e de outros recursos. O SO também estabelece uma prioridade para o processo (quando o SO é multitarefa), determina o código do programa de carregamento na memória e executa o programa propriamente dito. O programa, então, interage com o usuário ou com outros dispositivos além de executar a função pretendida.

2.2. Interrupções

As interrupções são ferramentas importantes para os sistemas operacionais, porque fornecem uma maneira eficaz para que o sistema operacional interaja com o *hardware*. A interrupção consegue informar ao sistema operacional as várias fontes de entrada dos eventos que exigem uma ação. A programação baseada em interrupção é suportada pela maioria de processadores atuais. As interrupções fornecem um computador uma maneira automática de executar um código específico em resposta a um determinado evento.

Quando uma interrupção é recebida, o *hardware* do computador suspende automaticamente o que programa está executando no momento, guarda seu status atual e desvia a execução do código para o programa associado previamente a essa interrupção. Em sistemas operacionais modernos, as interrupções são gerenciadas pelo núcleo do sistema operacional. As interrupções podem vir do *hardware* de computador ou de algum programa em execução.

Quando um dispositivo de *hardware* provoca uma interrupção, o núcleo do sistema operacional decide como tratar esse evento, geralmente executando algum código de processamento. Quando ocorrem várias interrupções, o funcionamento depende da prioridade da interrupção (por exemplo: uma pessoa

geralmente atende primeiramente a um alarme de incêndio antes de atender um telefone). O processamento de interrupções de *hardware* é uma tarefa que, geralmente, é delegada a dispositivo de *software*, que podem ser parte do núcleo do sistema operacional, parte de um outro programa, ou ambos.

Um programa pode também provocar uma interrupção[14] no sistema operacional. Se um programa deseja acessar um dispositivo de *hardware* por exemplo, pode interromper o núcleo do sistema operacional, que faz com que o controle seja passado de volta ao núcleo após o seu acesso. O núcleo processará, então, o pedido.

[14] **Interrupção** é uma técnica que permite a interrupção de um programa de computador e desvia o controle da UCP para tratar de um evento esporádico que necessita de tratamento imediato. Como exemplo de ações que precisam de tratamento de interrupção, citamos a digitação em um teclado.

2.3. Modo protegido e Modo supervisor

A maioria dos processadores modernos dispõem de operação dupla da modalidade: modo protegido e modo supervisor. Esses modos permitem que determinadas funções do processador central sejam controladas e alteradas apenas pelo núcleo do sistema operacional.

Entretanto, esse termo é usado geralmente, na teoria de sistema operacional, para referir todas as modalidades que limitam as capacidades dos programas que funcionam dessa forma, fornecendo funcionalidades como o endereçamento de memória virtual e limitando o acesso ao *hardware* de uma maneira definida por um programa que funciona no modo supervisor. Essas modalidades existem em processadores Intel x86, nos supercomputadores, minicomputadores e na maioria dos sistemas multiusuários.

Quando um computador começa a funcionar, primeiramente está funcionando automaticamente no modo supervisor. Os primeiros programas a funcionar no computador, como a BIOS, *bootloader* e o sistema operacional têm acesso ilimitado ao *hardware*

Na modalidade protegida, os programas podem ter o acesso a um conjunto maior de instruções do processador. Um programa de usuário pode deixar a modo protegido apenas chamando uma interrupção, fazendo com que o controle seja passado de volta ao núcleo. Dessa maneira, o sistema operacional pode manter o controle exclusivo sobre todos os programas que têm acesso ao *hardware* e à memória.

2.4. Gerência de memória

Dentre outras coisas, o núcleo do sistema operacional multitarefa deve ser responsável por controlar toda a memória de sistema que é usada pelos programas. O sistema operacional deve assegurar que um programa não interfira na memória já usada por um outro programa. Como os programas compartilham o mesmo conjunto de memória, cada um deve ter o acesso apenas à sua memória.

A gerência de memória cooperativa, usada pelos sistemas operacionais mais antigos, supõe que todos os programas fazem o uso voluntário da memória, e cada um tem a responsabilidade de não exceder a memória alocada para ele. No entanto, os programas contêm, frequentemente, erros que podem escrever fora da área inicialmente alocada, causando erro de execução nos demais programas.

Se um programa falhar, pode causar a sobrescrita de memória usada por um ou por vários programas. Outro problema mais grave é a execução de programas maliciosos, ou vírus, que podem alterar a memória de outros programas ou afetar a operação do próprio sistema operacional.

A proteção de memória permite o núcleo do sistema operacional delimitar o acesso dos processos à memória do computador. Existem vários métodos de proteção de memória, e os principais são: a segmentação da memória e a paginação. Todos os métodos exigem algum nível de suporte do *hardware* que não existe em todos os computadores. Esse dispositivo é conhecido como MMU (*Memory Management Unit*)

Na segmentação e na paginação, há determinados registros de modo protegido do processador central, que faz com que um programa apenas alcance um determinado endereço de memória. As tentativas de alcançar outros endereços provocarão uma interrupção que indica que o processador central entre no modo supervisor, fazendo com que o núcleo do sistema operacional tome uma medida de correção. Essa ocorrência é chamada **violação de segmentação**, e, como geralmente é difícil resolver o problema de programas diferentes, o resultado final consiste no encerramento do programa mal comportado e em relatório de erro.

2.5. Memória Virtual

O uso do endereçamento de memória virtual (tal como a paginação ou a segmentação) significa que o núcleo do sistema operacional pode definir a memória que cada programa pode usar em um determinado momento, permitindo que o sistema operacional gerencie o uso de memória para múltiplas tarefas.

Se um programa tentar acessar uma memória que não está em sua escala de memória permitida, mas essa memória não foi alocada para outro programa, o núcleo do sistema operacional será interrompido na mesma maneira quando um programa exceder sua memória alocada. Esse tipo de falha é chamado **falha de página** (*page fault*).

Quando o núcleo detectar uma falha que de página ajustará o tamanho da memória virtual usada por esse programa, ele concedendo-lhe o acesso à memória solicitada. Isso dá ao núcleo a capacidade de aproveitar ao máximo a memória, disponível Em sistemas operacionais modernos, as páginas de memória que são menos usadas, podem temporariamente ser armazenadas no disco ou em outros meios de armazenamento para liberar espaço na memória principal e para ser utilizado por outros programas. Isso é chamado *swap*, porque uma área da memória pode ser usada pelos vários programas, além de essa área de memória poder ser salva ou recuperada do disco sob demanda.

3. Multitarefa

A multitarefa refere-se à característica de executar vários programas independentes simultaneamente no mesmo computador, dando a impressão que se está executando várias tarefas ao mesmo tempo. A técnica mais usada é a partilha de tempo, isto é, cada programa usa uma parte do tempo do computador para executar a tarefa.

O núcleo do sistema operacional contém uma função de *software* chamada de escalonador, que determina quanto tempo cada programa gastará na execução do programa e quais controles de execução devem ser passados aos programas. O controle é passado a um processo pelo núcleo, que permite o acesso do programa ao processador central e à memória. Em um controle de tempo, nos computadores iniciais, é retornado ao núcleo através de algum mecanismo, de modo que um outro programa possa ser permitido usar o processador central. Essa passagem assim chamada do controle entre o núcleo e as aplicações é chamada **comutador de contexto**.

Um modelo de gerenciamento da alocação de tempo aos programas dos sistemas operacionais mais antigos é chamado **multitarefa cooperativa**. Nesse modelo, quando o controle é passado a um programa pelo núcleo do sistema operacional, ele pode executar o tempo que precisar antes de retornar o controle ao núcleo. Isso significa que um programa malicioso ou que esteja funcionando mal pode não somente impedir que outros programas usem o processador central mas também bloquear o sistema inteiro em um *loop* infinito.

A filosofia do sistema multitarefa preemptivo é aquela de assegurar que todos os programas usem um tempo do processador central limitado pelo

núcleo do sistema operacional. Isso implica que todos os programas devem ser limitados em quanto tempo são permitidos usar os recursos do processador central sem serem interrompidos. Os sistemas operacionais modernos empregam uma interrupção programada. Um temporizador, funcionando no modo protegido, é ajustado pelo núcleo do sistema operacional, que provoca um retorno ao modo supervisor depois que o tempo especificado decorrer, garantindo que o sistema não seja bloqueado.

4. Driver de dispositivo

Um driver de dispositivo é um tipo específico de *software* de computador desenvolvido para permitir a interação do Sistema Operacional (e dos aplicativos) com dispositivos de *hardware*. Tipicamente, ele faz a ligação com o subsistema específico do barramento ou de comunicações do computador no qual o dispositivo de *hardware* está conectado, fornecendo comandos, transmitindo e recebendo dados do dispositivo, e, na outra extremidade, comunicando com o sistema operacional e as aplicações de *software*.

O objetivo-chave de um driver de dispositivo é abstração. Cada modelo de *hardware* (mesmo dentro da mesma classe de dispositivo) é diferente, possui comando e modelos de dados diferentes. A cada nova versão de *hardware* liberado pelos fabricantes, por motivo de desempenho ou de funcionalidade, exigem novos e, talvez, diferentes comandos. Os computadores e seus sistemas operacionais não podem estar preparados para saber controlar todos os dispositivos atuais e futuros.

Para resolver esse problema, os desenvolvedores de sistemas operacionais estabelecem uma forma padronizada para acessar as funções dos dispositivos de *hardware*. Os fabricantes de um determinado *hardware*, então, fazem um programa específico que traduz os comandos e funções padronizadas pelo desenvolvedor do sistema operacional para os comandos específicos do seu *hardware*. Assim, qualquer novo *hardware* com o seu respectivo driver de dispositivo (para um determinado sistema operacional) pode funcionar plenamente em um sistema operacional.

Síntese do capítulo

Neste capítulo apresentamos os diversos *softwares* usados em um computador e sua interação com a arquitetura de computadores. Inicialmente, mostramos os conceitos de conjunto de instruções e como as instruções executam o processamento em um computador. Em seguida, apresenta-

mos os conceitos básicos de um sistema operacional e sua interação com a arquitetura do computador.

Atividades de avaliação

1. Lendo as funções do sistema operacional, relacione as funções relativas a um sistema operacional monousuário e monotarefa (executa uma tarefa por vez).
2. Pesquise sobre a história dos sistemas operacionais e relacione as principais melhorias que são utilizadas até hoje.
3. Um sistema operacional tem total acesso ao *hardware* de computador. Como você poderia melhorar a segurança para evitar incidentes?

Leituras, filmes e sites

Leituras

DA SILVA, Vanderlei Alves Santos. **Hardware e Software**. Joinville/SC: Clube de Autores, 2008.

REISSWITZ, Flavia. AnÁlise de **Sistemas V. 4: Processos de desenvolvimento de software**. Joinville/SC: Clube de Autores, 2009.

TANENBAUM, Andrew S. e WOODHULL, Albert S. **Sistemas Operacionais:** Projetjos e Implementação. Porto Alegre/RS: Bookman, 2008.

Filmes

Curso de Algoritmos e Programação: O que é Linguagem de Programação e Maquina. Duração: 32min 43s. Nessa video aula se explica o que é Linguagem de Programação e Linguagem de Maquina de forma bem simples e detalhada. Acesso em: https://www.youtube.com/watch?v=u1HG0dAka5M

Quantas linguagens de programação você deveria saber? Duração: 14min 21s. O video é destinado a perguntas e respostas com Rodrigo Souza. Neste video o questionamento é: Rodrigo em quantas linguagens você programa, e quantas linguagens é humanamente possível de se aprender na sua opinião? Vale a pena aprender muitas linguagens ou é melhor se concentrar só em

uma? Digo isso por que sempre que passo para uma linguagem nova eu acabo perdendo a prática ou até mesmo me esquecendo como se programa na anterior. Acesso em: https://www.youtube.com/watch?v=zZFP8UVD4A4

Sistemas operacionais Windows e Linux. Duração: 37min 53s. O video conceitua os sistemas operacionais Windows e Linux e discute especificidades e funcionalidades. Acesso em: https://www.youtube.com/watch?v=TJvVdwsj1Qw

Sites

Conjunto de Instruções (em inglês)

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

Conjunto de Intruções do microcontrolador 8051 (em Inglês)

http://www.win.tue.nl/~aeb/comp/8051/instruction-set.pdf

Apostila sobre Sistemas Operacionais publicada na revista RITA (em Português)

http://www2.unitins.br/BibliotecaMidia/Files/Documento/BM_6335 36494556679022sistemasoperacionais.pdf

Artigo sobre Sistemas Operacionais (em Português)

http://informatica.hsw.uol.com.br/sistemas-operacionais.htm

Sistemas Operacionais (em Inglês)

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

Referências

STALLINGS, W. **Arquitetura e Organização de Computadores** 5ª Ed. Editora: Prentice Hall, 2002.

HENNESSY, J. L. & PATTERSON, D. A. **Arquitetura de Computadores: Uma Abordagem Quantitativa** 3ª Ed. Editora: Campus, 2003.

WOODHULL, A. S. & TANENBAUM, A. S. **Sistemas Operacionais:** Projeto e Implementação 3ª Ed. Editora: Artmed, 2008.

TANENBAUM, A. S. **Sistemas Operacionais Modernos** 2ª Ed. Editora: Prentice-Hall, 2009.

Parte 5

Tópicos avançados em arquitetura de computadores

Capítulo 9

Pipeline

Objetivos

Neste capítulo vamos apresentar alguns tópicos avançados em Arquitetura de Computadores. São técnicas modernas que têm como objetivo aumentar o desempenho dos computadores construindo novas alternativas arquiteturais. A primeira técnica é o Pipeline, na qual unidades de processamento diferentes realizam operações em série da mesma forma que uma linha de produção. A outra técnica trata de processamento paralelo, envolvendo todas as formas de paralelizar o processamento de instruções e de dados para aumentar o desempenho dos sistemas.

Introdução

O termo em inglês pipeline significa encanamento, tubulação. Na computação, um pipeline é um conjunto de elementos que processam dados conectados em série, de modo que a saída de um elemento seja a entrada do seguinte. A ligação dos módulos se assemelha a uma ligação de canos para formar uma tubulação, por isso o nome pipeline. Por falta de uma tradução para Português que tenha consenso na comunidade científica, usaremos o termo original pipeline.

O processamento nos elementos de um pipeline são executados paralelamente, de forma que todos terminem no mesmo tempo. O pipeline se assemelha a uma linha de produção onde cada unidade executa a sua função e passa para a unidade seguinte. Para corrigir as diferenças no tempo de processamento em cada unidade, geralmente usamos algum dispositivo de armazenamento entre os elementos.

1. Tipos de Pipeline

Os pipelines em um computador podem ser:

- **Pipeline de instrução**, tais como o pipeline clássico RISC, que são usados nos processadores para sobrepor a execução de instruções múltiplas com os mesmos circuitos. Os circuitos são divididos geralmente acima em estágios, incluindo: a busca de instrução, a decodificação de instrução, a execução de operação, e o salvamento em registrador, onde cada estágio processa uma instrução diferente e sequencial de cada vez.
- **Pipelines gráficos**, encontrados na maioria de placas gráficas, que consistem em unidades aritméticas múltiplas ou em processadores específicos, que executam os vários estágios das operações de renderização gráfica comuns (projeção de perspectiva, corte de janela, cálculo de cor e de luz, renderização, etc.).
- **Pipelines de *software***, que consiste nos múltiplos processos de *software,* de modo que o fluxo de saída de um processo seja encaminhado automaticamente para o fluxo de entrada do seguinte. Os pipelines de Unix são a aplicação clássica deste conceito.

2. Conceitos

O pipeline é um conceito natural na vida quotidiana, por exemplo, em uma linha de produção. Considere a montagem de um carro, em que determinadas etapas na linha ocorrem isoladamente: instalar o motor, montar a carroceria e instalar as rodas (nessa ordem, com etapas intermediárias arbitrárias). Um carro na linha de produção pode ter somente uma das três etapas executadas em um determinado instante. Depois que o carro tem seu motor instalado, move-se para ter sua carroceria montada, deixando o módulo de instalação do motor disponível para o carro seguinte.

O primeiro carro move-se então para a instalação da roda, o segundo carro, para a montagem da carroceria, e um terceiro carro começa a ter seu motor instalado. Se a instalação do motor toma 20 minutos, a montagem da carroceria toma 5 minutos, e a instalação da roda toma 10 minutos, o tempo para terminar todos os três carros, quando somente um carro pode ser montado individualmente, tomaria 105 minutos. Por outro lado, usando a linha de produção, o tempo total para terminar todos os três é 75 minutos. Nesse momento, os carros adicionais sairão da linha de produção a cada 20 minutos.

O exemplo da linha de produção mostra que o pipeline não diminui o tempo de processamento de uma única instrução, que permanece a mesma, mas aumenta a capacidade de processamento total do sistema ao processar,

em sequência, um conjunto de dados. Um pipeline longo conduz ao aumento da latência – o tempo exigido para que um sinal propague através de um pipeline cheio.

Um sistema de pipeline exige tipicamente mais recursos (elementos do circuito, unidades de processamento, memória de computador, etc.) do que um que executa uma instrução de cada vez, porque seus estágios não podem reusar os recursos de um estágio antecedente. Além disso, o pipeline pode aumentar o tempo total para que uma instrução individual termine.

Um ponto importante no projeto de um pipeline é o balanceamento de seus diversos estágios. Voltando ao exemplo da linha de produção de carros, se a instalação do motor levasse 15 minutos em vez de 20 minutos, e a instalação das rodas levasse 15 minutos em vez de 10 minutos, apesar do tempo de produção (tempo entre início e fim da montagem) continuar sendo 35 minutos, a linha de produção poderia entregar um carro a cada 15 minutos, em vez de 20 minutos na organização inicial.

Outra questão importante quanto ao projeto de pipeline á a colocação de memórias (buffers) entre os estágios do pipeline para compensar os tempos de processamento irregulares ou quando dados podem ser criados ou destruídos ao longo do pipeline.

3. Implementações de Pipeline

Existem três categorias de implementação de pipeline: pipeline síncrono com buffer, pipeline assíncrono com buffer e pipeline sem buffer.

3.1. Pipeline Síncrono com Buffer

Os microprocessadores convencionais são circuitos síncronos, assim é natural a construção de pipelines síncronos. Nesses pipelines, um registrador é introduzido entre dois estágios do pipeline e sequenciados com um relógio global. O ciclo de relógio utilizado é ajustado para ser maior tempo de atraso entre estágios do pipeline, de modo a garantir que, no final do ciclo de relógio, todos os dados do pipeline estarão prontos para o estágio seguinte do pipeline.

3.2. Pipeline Assíncrono com Buffer

Os pipelines assíncronos com buffer são usados em circuitos assíncronos e têm seus registradores cronometrados de forma assíncrona. Em linhas gerais, eles usam um comando de requisição/reconhecimento do sistema, onde cada estágio pode detectar quando seu processamento é finalizado. Quando um estágio é finalizado, e o estágio seguinte lhe emitiu um sinal do "requisição", o estágio emite um sinal de "reconhecimento" para o estágio seguinte,

e um sinal do "requisição" ao estágio anterior. Quando um estágio recebe um sinal de "reconhecimento", ele envia um sinal para seu registrador da entrada receber os dados do estágio anterior e processá-lo.

3.3. Pipeline sem Buffer

Os pipelines sem buffer, chamados "pipelines de ondas", não têm registradores (buffers) entre seus estágios. Nesse caso, os atrasos de cada estágio do pipeline são "equilibrados", de modo que, para cada estágio, a diferença entre a estabilização dos dados da saída do primeiro estágio e a estabilização dos dados de entrada do último estágio seja minimizado. Assim, o fluxo de dados nesse pipeline segue em "ondas", e cada onda é mantida o menor tempo possível.

A taxa máxima que os dados podem entrar em um pipeline sem buffer é determinada pela diferença máxima no atraso entre a primeira parte dos dados que saem do pipeline e a última parte de dados que entram. Como não há buffers ao longo do pipeline, os pipelines sem buffers conseguem obter as maiores taxas de processamento.

4. Pipeline de Instruções

Um pipeline da instrução é uma técnica usada em projetos de computadores e de outros dispositivos eletrônicos digitais para aumentar sua capacidade de processar instruções (o número de instruções que podem ser executadas em uma unidade de tempo).

A idéia fundamental é separar o processamento de uma instrução em uma série de etapas independentes, com armazenamento no fim de cada etapa. Isso permite que os circuitos do controle do computador consigam emitir instruções de controle na taxa de processamento da etapa a mais lenta, que ainda é muito mais rápida do que o tempo necessário para executar todas as etapas. O termo *pipeline* refere-se ao fato de que cada etapa está entregando os dados para a etapa seguinte (como a água), e cada etapa é conectada à seguinte (como as ligações de uma tubulação).

A origem do pipeline remonta, provavelmente, no projeto ILLIAC II da IBM, que propôs unidades independentes para processar as etapas de execução de uma instrução: buscar instrução, decodificar e executar.

A maioria de processadores modernos são sincronizados por um pulsos de relógio. O processador consiste internamente em lógica e memória (flip-flops). Quando o sinal de relógio dispara, os flip-flops recebem uma instrução nova, e a lógica exige um período de tempo para decodificar a nova instrução. Então, o impulso de relógio seguinte chega, e os flip-flops recebem outra vez sua nova instrução, e assim por diante. Quebrando a lógica em partes me-

nores e introduzindo flip-flops entre as partes de lógica, o atraso que a lógica impactava no tempo final de saída é reduzido. Dessa maneira, o período de pulso de relógio pode ser reduzido. Por exemplo: o pipeline de processador RISC quebra o processamento de instrução em cinco estágios com um conjunto de flip-flops entre cada estágio.

a) Busca de Instrução

b) Decodifica instrução e busca operando (registrador)

c) Executa instrução

d) Acessa memória

e) Escreve de volta o registrador

Quando um programador (ou o compilador) escrevem o código executável, faz a suposição que cada instrução seguinte está executada após a execução da instrução anterior. Essa suposição é invalidada pelo pipeline, porque a instrução seguinte vai começar a ser processada antes que a instrução anterior termine. Quando isso ocorre, faz com que um programa se comporte incorretamente, situação conhecida como "bolha" (*hazard*). Existem várias técnicas para resolver tais como incluir uma pausa no processamento para aguardar a conclusão do processamento da instrução adiante.

Uma arquitetura sem pipeline é ineficiente porque alguns componentes do processador (módulos) ficam em repouso enquanto um outro módulo estiver ativo durante o ciclo de instrução. O pipeline não elimina completamente o tempo inativo em um processador, mas executar paralelamente a função daqueles módulos inativos melhora significativamente a execução do programa.

Os processadores com pipeline são organizados em estágios semi-independentes, que podem trabalhar separados. Cada estágio é organizado em forma de uma "corrente", isto é, a saída de cada estágio alimenta a entrada de um outro estágio até que o trabalho esteja feito. Essa organização do processador não reduz o tempo total de execução de uma única instrução, mas permite que o tempo total de processamento de um conjunto de instruções subsequentes seja reduzido significativamente.

Um pipeline mais profundo significa que há mais estágios no pipeline, e, consequentemente, menos portas lógicas em cada estágio do pipeline. Isto acarreta, geralmente, que a frequência do processador pode ser aumentada, assim como o tempo do ciclo de relógio pode ser reduzido. Isso acontece porque há poucos componentes em cada estágio do pipeline, assim, o atraso de propagação é diminuído cada estágio.

Infelizmente, nem todas as instruções são independentes. Em um pipeline simples, terminar a execução de uma instrução pode exigir 5 estágios.

Para operar no máximo desempenho, este pipeline precisará executar 4 instruções subsequentes independentes enquanto a primeira instrução não terminar. Se 4 instruções que não dependem do resultado da primeira instrução não estão disponíveis, a lógica de controle do pipeline deve introduzir uma parada ou a perda de um ou mais ciclos de relógios no pipeline até que a dependência esteja resolvida.

Felizmente, técnicas como otimizações na compilação podem reduzir significativamente os casos onde a parada é exigida. Enquanto o pipeline pode aumentar o desempenho de um processador sobre uma arquitetura sem o pipeline por um fator do número de estágios (supondo que a frequência de relógio seja compatível com o número de estágios), na realidade, a maioria do código usado não permite essa otimização ideal.

Atividades de avaliação

1. Pesquise os diversos tipos de pipeline implementados em processadores.
2. Pesquise sobre um processador de sinais digital e identifique funções onde um pipeline poderia ajudar na melhoria do desempenho.

Capítulo 10

Processamento Paralelo

Introdução

A computação paralela é uma técnica da computação em que muitos cálculos podem ser realizados simultaneamente, baseados no princípio que os grandes problemas podem, normalmente, ser dividido em problemas menores, que são resolvidos simultaneamente (paralelamente). Há diversas formas diferentes de computação paralela: nível de bit, nível da instrução, nível de dados e paralelismo de tarefas.

O paralelismo foi empregado por muitos anos, principalmente na computação de alto desempenho, mas o interesse tem crescido ultimamente devido às limitações físicas dos semicondutores que têm impedido o aumento da frequência nos circuitos integrados. Outra limitação física é que a temperatura e consumo de energia crescem exponencialmente, enquanto o desempenho cresce linearmente, tornando o aumento das taxas de relógio um problema.

Como o consumo de energia (e, consequentemente, a geração de calor) por computadores têm se transformado em um desejo nos últimos anos, a computação paralela transformou-se no paradigma dominante de arquitetura de computadores para o aumento do desempenho, principalmente sob a forma dos processadores com múltiplos núcleos (multicore).

Os computadores paralelos podem ser classificados de acordo com o nível em que o *hardware* suporta o paralelismo. Os computadores que usam processadores de múltiplos núcleos (multicore) são os que têm elementos do processamento múltiplo dentro de uma única máquina. Por outro lado, cluster, MPPs (Massive parallel processing) e as grades computacionais usam computadores múltiplos para trabalhar em uma mesma tarefa. As arquiteturas paralelas especializadas podem ser usadas ao lado dos processadores tradicionais para acelerar tarefas específicas.

Os programas para computação paralela são mais difíceis de escrever do que os programas sequenciais, porque a simultaneidade introduz diversas novas classes de erros de *software*, como condições corrida. Comunicação e sincronização entre subtarefas diferentes são tipicamente os grandes obstáculos para se obter um bom desempenho de programa paralelo.

O aumento de desempenho de programas em paralelo são governados pela Lei de Amdahl.

1. Lei de Amdahl

Em princípio, o aumento do desempenho de sistemas paralelizados deveria ser linear, dobrando o desempenho quando dobramos o número de processadores. No entanto, para trabalhar em paralelo, é necessário compartilhar alguns recursos (como memória) e muito poucos algoritmos paralelos conseguem ótimo desempenho. A maioria deles consegue um crescimento aproximadamente linear para um pequeno número de elementos de processamento, que reduz o incremento com o aumento de processadores até estabilizar com um valor constante quando usamos um grande número de elementos de processamento.

O potencial de incremento de desempenho de um algoritmo em uma plataforma da computação paralela é dado pela lei de Amdahl, formulada originalmente por Gene Amdahl, nos anos 60. Essa equação indica que uma parcela pequena do programa, que não pode paralelizada, limitará o aumento da capacidade geral do sistema paralelo. Qualquer grande problema matemático ou de Engenharia consistirá tipicamente em diversas partes paralelizáveis e em diversas partes não paralelizáveis (sequenciais). Esse ganho é expresso pela equação:

$$S = \frac{1}{1 - P}$$

onde: S é o aumento de desempenho do sistema, e P é o percentual do algoritmo que pode ser paralelizado.

A Lei de Amdahl pode ser mais bem visualizada através do gráfico mostrado na figura a seguir. Nele podemos notar que, se um algoritmo pode apenas ser 90% paralelizável, o desempenho máximo que podemos conseguir é 10 vezes, não importando a quantidade de processadores usados.

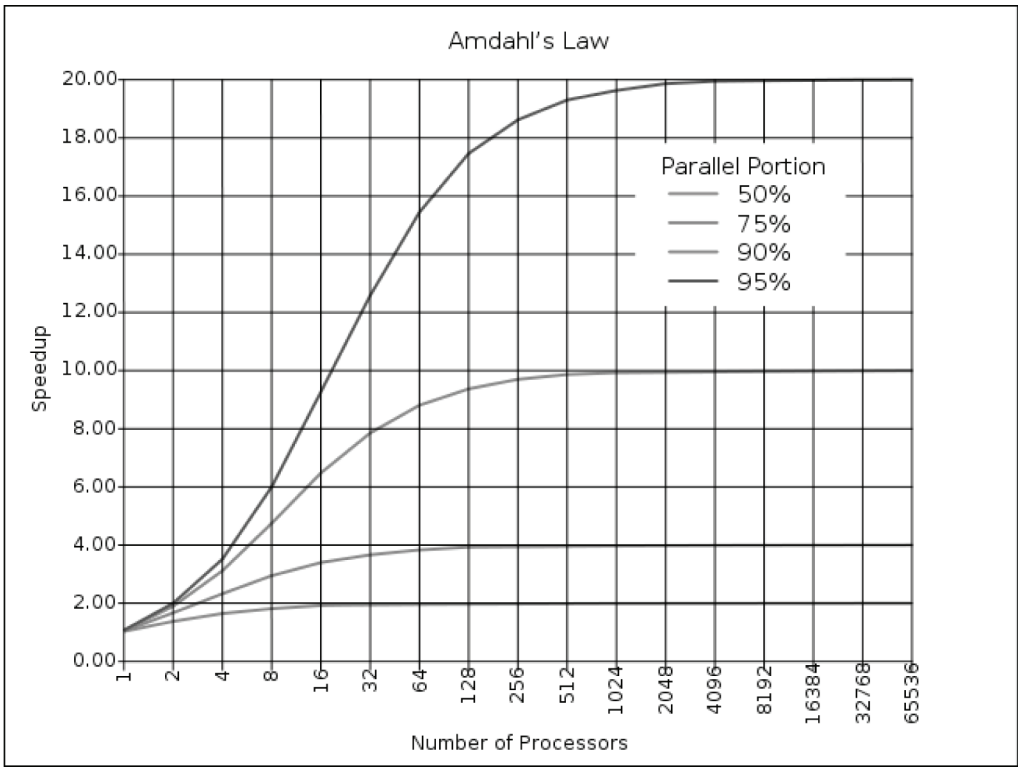


Figura 46 - Gráfico demonstrando a Lei de Amdahl

2. Taxonomia de Flynn

Michael J. Flynn criou um dos sistemas de classificação para computadores paralelos e sequenciais e para programas conhecido como a Taxonomia de Flynn. Ela classifica programas e computadores quanto ao modo de executar instruções de a busca de dados, sejam elas individuais ou múltiplas.

	Instrução Única	Instrução Múltipla
Dado Único	SISD	MISD
Dado Múltiplo	SIMD	MIMD

Figura 47 - Taxonomia de Flynn

A classificação SISD (*single-instruction-single-data*) indica Instrução-única-Dado-único representa um programa inteiramente sequencial. A classificação SIMD (*single-instruction-multiple-data*) indica Instrução-única-Dados-múltiplos e é análoga a fazer uma mesma operação repetidamente sobre um grande número de dados. Isso é feito, geralmente, em aplicações de processamento de sinais, como tratamento de som e de imagem. A categoria MISD (*multiple-instruction-single-data*) é uma classificação raramente usada. Alguns pesquisadores consideram o pipeline um exemplo dessa categoria, mas não

é consenso. A classificação MIMD (*multiple-instruction-multiple-data*) significa Múltiplas-instruções-múltiplos-dados é a mais comum e que se aplica a maioria dos programas paralelos.

De acordo com alguns pesquisadores, como David A. Patterson e John L. Hennessy, a maioria das máquinas são híbridas com essas categorias, mas esse modelo clássico sobreviveu porque é simples, fácil de compreender e dá ao estudante uma primeira visão dos sistemas paralelos.

3. Arquiteturas de Paralelismo em Hardware

Os conceitos de paralelismos podem ser implementados de diversas maneiras, conforme a arquitetura existente e aos objetivos específicos da arquitetura. Apresentaremos a seguir várias técnicas usadas para implementar o paralelismo em *hardware*, não dizendo que sejam exclusivas. Na verdade, a maioria das arquiteturas paralelas usa uma combinação de duas ou mais técnicas.

3.1. Memória e Comunicação

A memória central em um computador paralelo é uma memória compartilhada (entre todos os elementos de processamento em um único espaço de endereçamento) ou memória distribuída (em que cada elemento de processamento tem seu próprio espaço de endereçamento local). A memória distribuída refere-se ao fato de que ela está distribuída logicamente, mas implica, normalmente, que está distribuída fisicamente, isto é, fisicamente localizada em placas e unidades separadas.

Memória compartilhada distribuída e virtualização de memória combinam as duas técnicas, onde o elemento de processamento tem sua própria memória local e acesso à memória compartilhada em processadores não-locais. Os acessos à memória local são tipicamente mais rápidos do que os acessos à memória compartilhada não-local.

3.2. Arquitetura NUMA

A arquitetura NUMA (*Non-Uniform Memory Access*), Acesso à Memória Não Uniforme, usa a idéia que os processadores em um diretório podem alcançar a memória desse diretório com menos latência do que a memória de acesso na memória de outro processador (diretório). Apresentamos a seguir um diagrama de um sistema com arquitetura NUMA.

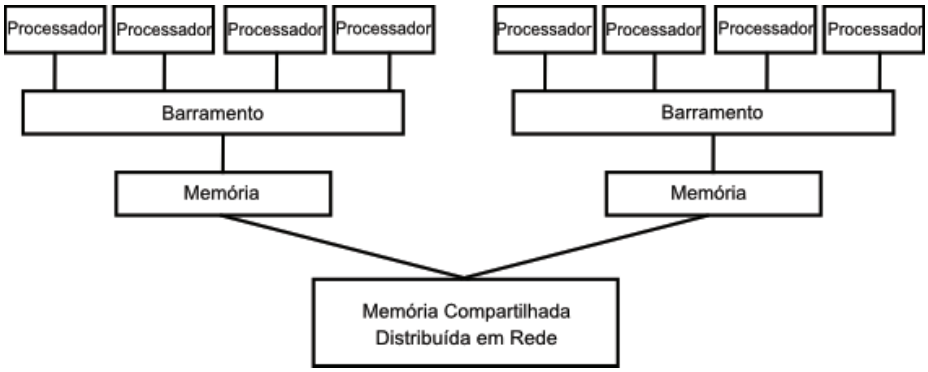


Figura 48 - Diagrama representando uma Arquitetura NUMA

As arquiteturas de computadores em que cada elemento da memória central pode ser acessado com uma latência e taxa de leitura/escrita iguais são conhecidas como sistemas de Acesso Uniforme à Memória (UMA - *Uniform Memory Access*). Tipicamente, isso pode ser conseguido somente por um sistema de memória compartilhada, em que a memória não está distribuída fisicamente. Um sistema que não tenha essa propriedade é conhecido como uma arquitetura Acesso à Memória Não Uniforme (NUMA). Os sistemas de memória distribuídos sempre têm o acesso NUMA.

Os sistemas de computadores monoprocessados geralmente empregam pequenas memórias cache rápidas situadas perto do processador, física e logicamente que vão armazenar cópias provisórias de valores da memória (instrução e dado). Os sistemas de computadores paralelos têm dificuldades com caches que podem armazenar o mesmo valor em mais de uma posição de memória, com a possibilidade de execução incorreta de programa.

Esses computadores exigem um sistema para manter a coerência dos caches, que acompanhem a mudança nos valores nele guardados para atualizar na memória principal, bem como nos demais caches, assegurando, assim, a correta execução dos programas. O *bus snooping* ("escuta do barramento") é um dos métodos mais comuns para manter coerentes os valores de cache que estão sendo alterados.

Projetar grandes sistemas de capacidade elevada para manter coerência dos caches é um problema muito difícil na área de arquitetura de computadores. Em consequência, as arquiteturas de computadores com memória compartilhada não têm escalabilidade que podemos obter com os sistemas de memória distribuída.

A comunicação processador-processador e a comunicação processador-memória podem ser implementadas no *hardware* de diversas maneiras, incluindo comunicação através da memória compartilhada, matriz de comu-

tação, barramento compartilhado ou de uma rede da interconexão, e um conjunto grande de topologias, compreendendo estrela, anel, árvore, o hipercubo ou malha multidemensional.

Os computadores paralelos baseados em redes da interconexão precisam ter algum tipo da roteamento para permitir a passagem das mensagens entre os nós que não estão conectados diretamente.

4. Classes de computadores paralelos

Os computadores paralelos podem ser classificados de acordo com o nível de paralelismo que o *hardware* suporta. Essa classificação é análoga à classificação de ligação de nós de computação básicos. Essas classificações não são mutuamente exclusivas, por exemplo: um cluster de multiprocessadores simétricos é relativamente comum.

4.1. Processador de Múltiplos Núcleos

Um processador com múltiplos núcleos (*multicore*) inclui várias unidades de processamento ("núcleos") em um mesmo circuito integrado. Esses processadores diferem dos processadores superescalar, que podem executar múltiplas instruções por ciclo de um fluxo de instrução (*thread*); ao contrário, um processador multicore pode executar instruções múltiplas por ciclo de fluxo de instruções múltiplas. Cada núcleo em um processador multicore pode ser superescalar, então, em cada ciclo, cada núcleo pode executar instruções múltiplas de um fluxo de instrução.

Multithreading simultâneo (onde o exemplo mais conhecido é o *HyperThreading da Intel*) foi um das primeiras implementações de processador pseudo multicore. Um processador capaz de realizar multithreading simultâneo tem somente uma unidade de execução ("núcleo"), mas quando essa unidade de execução está em repouso (por exemplo, durante uma falta de cache), usa-se essa unidade de execução para processar uma segunda tarefa. O microprocessador Cell da IBM, projetado para ser usado no console Sony Playstation 3, é um outro exemplo de processador multicore.

4.2. Multiprocessamento Simétrico

Um multiprocessador simétrico (SMP - *Symmetric Multiprocessor*) é uma arquitetura de computadores com múltiplos processadores idênticos que compartilham a memória e se conectam através de um barramento. A disputa do barramento único de comunicação limita a escalabilidade dessa arquitetura. Em consequência, os equipamentos SMP geralmente não uti-

lizam mais de 32 processadores. No entanto, por causa do tamanho pequeno dos processadores atuais e da redução significativa das exigências de comunicação com a memória compartilhada devido ao uso de grandes caches, os multiprocessadores simétricos são extremamente interessantes devido à sua boa relação custo-benefício, desde que haja memória e largura de banda suficiente.

4.3. Computação Cluster

Um cluster é um grupo de computadores independentes que funcionam junto através de interfaces de comunicação de alta velocidade, de forma que possam ser considerados como um único computador. Os clusters são compostos por múltiplas máquinas autônomas conectadas por uma rede de alta velocidade. Quando as máquinas em um cluster não fores idênticas (simétricas), a distribuição de carga é muito mais difícil. Um exemplo muito conhecido de cluster é o Beowulf.

A tecnologia de Beowulf foi desenvolvida originalmente por Thomas Esterlino e Donald Becker. O Beowulf é um cluster construído com múltiplos computadores tipo IBM-PC idênticos comprados no comércio e conectados com uma rede local do tipo Ethernet e TCP/IP. Uma máquina especial gerencia o conjunto de computadores do cluster Beowulf. A maioria dos supercomputadores da lista TOP500 é formada por clusters.

5. Computação Grade (Grid)

A Computação em Grade, também conhecida como Grid, é mais uma forma de computação paralela distribuída. Na computação em grade, os computadores se comunicam através da internet para realizar um trabalho de processamento de informações. Mas, ao contrário da computação em cluster, que pressupõe que todos os computadores sejam iguais e instalados em um mesmo local, na computação em grade, podemos usar computadores diferentes instalados em locais distintos, bastando apenas que estejam ligados na internet.

Por causa da baixa largura de banda e da latência extremamente elevada encontrada na Internet, uma grade consegue processar somente problemas que possam ser divididos e que os resultados não tenham dependência entre si (esse tipo de problema é chamado de *embarrassingly parallel*, ainda sem uma tradução para Português). Alguns exemplos desse tipo de problema são: renderização de imagens, quebra de senha por força bruta, bioinformática, simulações de fisa de partículas e previsões climáticas. Muitas aplicações de computação da grade foram criadas, onde o SETI@home e Folding@ Home são os exemplos os mais conhecidos.

A maioria das aplicações de computação em grade usam algum tipo de *middleware*, o *software* instalado entre o sistema operacional e a aplicação para controlar recursos de rede, e padroniza a API de *software*. O *middleware* mais conhecida de computação em grade é o *Berkeley Open Infrastructure for Network Computing* (BOINC), Frequentemente, o *software* de computação em grade usa os ciclos de folga dos computadores hospedeiros, executando as computações quando um computador está em repouso e sem executar outras atividades importantes.

6. Arquitetura Superescalar

Um processador com arquitetura superescalar executa uma forma de paralelismo chamado **paralelismo ao nível de instrução** dentro de um único processador. Ela permite uma execução do programa mais rápida no processador do que usar outras formas como aumento da taxa de relógio. Um processador superescalar executa mais de uma instrução durante um ciclo de relógio, disparando, simultaneamente, instruções múltiplas à unidades funcionais redundantes do processador.

Cada unidade funcional não é um outro núcleo separado do processador central, mas um recurso de execução dentro de um único processador central, tal como uma unidade de lógica aritmética, um registrador de deslocamento ou um multiplicador.

Quando um processador superescalar também implementar um pipeline de instrução, o desempenho também é aumentado, porque são duas técnicas diferentes e não conflitantes que melhoram o desempenho do processador.

A técnica de processador superescalar é associada a diversas técnicas tradicionais, todas aplicadas a um mesmo núcleo de processador:

- As instruções são extraídas de um fluxo sequencial de instruções.
- O *hardware* do processador central verifica dinamicamente se há dependências de tempo entre instruções e dados em tempo de execução (em vez de fazer a verificação prévia durante a compilação).
- Aceita múltiplas instruções por ciclo de relógio.

6.1. História

O CDC 6600 de Seymour Cray de 1965 é mencionado frequentemente como o primeiro projeto superescalar. O processador Intel i960CA (1988) e o AMD 29000 séries 29050 (1990) foram os primeiros microprocessadores superescalares em um único chip disponíveis comercialmente. Os fabricantes de processadores RISC preferem utilizar a arquitetura superescalar nos seus micro-

-processadores porque o projeto de microcomputadores RISC geralmente permite a expedição direta de instrução e a inclusão de unidades funcionais múltiplas (tais como ALUs) em um único processador central. Essa era a razão pela qual os projetos do RISC eram mais rápidos do que projetos do CISC com os anos 80 e 90.

Excetos os processadores usados em microcontroladores usados em equipamentos de baixa capacidade, os sistemas embutidos e os dispositivos alimentados por pilhas, todos os processadores centrais de uso geral implementam a arquitetura superescalar a partir de 1998.

O Pentium foi o primeiro processador da família x86 com arquitetura superescalar. O Nx586, o Pentium Pro e o AMD K5 estavam entre os primeiros projetos que decodificavam instruções x86 de forma assíncrona e dinâmica, criando uma sequência de microcódigo antes da execução real em uma arquitetura superescalar. Isso abriu a possibilidade a implementação de escalonamento dinâmico de instruções, o que permitiu a extração de mais paralelismos das instruções do que as gerações anteriores de processadores. Isso também simplificou a execução especulativa e possibilitou a utilização de frequências mais elevadas comparadas aos projetos anteriores.

6.2. Escalar e Superescalar

Os processadores mais simples são processadores escalares. Cada instrução executada por um processador escalar manipula tipicamente um ou dois campos de dados de cada vez. Por outro lado, cada instrução executada por um processador vetorial opera, simultaneamente, em muitos campos de dados. Essa analogia mostra a diferença entre uma operação escalar e a aritmética vetorial. Um processador superescalar é uma mistura dos dois tipos. Cada instrução processa um campo de dados, mas, como há muitas unidades funcionais redundantes dentro de cada processador, as instruções múltiplas podem processar múltiplos dados separados simultaneamente.

O projeto de um processador superescalar enfatiza o melhoramento da acurácia do expedidor de instrução para permitir que mantenha as múltiplas unidades funcionais em uso todas as vezes. Isso se tornou cada vez mais importante quando o número de unidades de processamento aumentou. Quando os primeiros processadores superescalares tinham duas ULAs e um único UPF, um projeto moderno, tal como o PowerPC 970, incluía quatro ULAs, dois UPFs e duas unidades de SIMD. Se o expedidor é ineficaz em manter todas estas unidades ocupadas com instruções, o desempenho do sistema será prejudicado.

Um processador superescalar geralmente mantém uma taxa da execução superior a uma instrução por o ciclo de máquina. Mas processar múltiplas

instruções simultaneamente não faz uma arquitetura ser superescalar, porque processadores com pipeline ou as arquiteturas multicore também conseguem essa funcionalidade, mas com métodos diferentes.

Em um processador superescalar, o expedidor lê as instruções da memória e decide qual delas podem ser executadas paralelamente, despachando para as unidades funcionais redundantes contidas dentro do único processador central. Consequentemente, um processador superescalar pode ser projetado para ter os vários pipelines paralelos, cada qual processando as instruções simultaneamente a partir de um único fluxo de instruções.

6.3. Limitações

A melhoria do desempenho das técnicas de arquiteturas superescalar é limitada por dois pontos chave:

- o grau de paralelismo intrínseco no fluxo de instruções, isto é, a quantidade limitada do paralelismo ao nível de instrução;
- o custo da complexidade e do tempo dispendido pelo expedidor de instrução e da dependência associada a verificação da lógica do programa.

Os programas binários executáveis têm vários graus de paralelismo intrínseco. Em alguns casos, as instruções não são dependentes de outras e podem ser executadas simultaneamente. Em outros casos, são interdependentes: uma instrução depende de recursos ou resultados da outra instrução. As instruções a = b + c; d = e + f podem ser executadas paralelamente porque nenhums dos resultados depende de outros cálculos. Entretanto, as instruções a = b + c; b = e + f não puderam ser executadas paralelamente, dependendo da ordem em que as instruções terminam e como os dados se movem através das unidades.

Quando o número de instruções que podem ser executadas simultaneamente aumenta, o custo da verificação de dependências aumenta extremamente rápido. Isso é agravado pela necessidade de se verificar as dependências no tempo de execução e na taxa de relógio do processador central. Esse custo inclui as portas lógicas adicionais exigidas para executar as verificações e os atrasos de tempo através dessas portas. As pesquisas mostram que o aumento de custo de portas, em alguns casos, pode ser linear, e o atraso no tempo tem um aumento logarítmico, criando um problema matemático de permutação combinatória.

Mesmo que o fluxo da instrução não contenha nenhuma interdependência de instruções, um processador superescalar deve verificar para ver se há essa possibilidade, já que não há nenhuma garantia que isso não ocorra, e a falha decorrida pelo um erro de dependência produziria resultados incorretos.

Não importa o quão avançado é o processo de fabricação do semicondutor ou como rapidamente a velocidade de comutação é aumentada, existe um limite prático em termos de quanto as instruções podem ser despachadas simultaneamente. Quando os avanços nos projetos de processadores permitirão números sempre maiores de unidades funcionais (por exemplo, ULAs), a carga de verificação das dependências de instrução cresce tão rapidamente, que o limite para aumento de desempenho da arquitetura superescalar chega a um limite muito rápido, na ordem de cinco a seis instruções despachadas simultaneamente.

Porém, mesmo que a verificação de dependência fosse realizada a uma velocidade infinitamente rápida em um processador superescalar, por outro lado, ainda teríamos a dependência do próprio fluxo de instruções, que também limitaria o aumento da velocidade. Assim, o grau de paralelismo intrínseco no fluxo de instruções seria uma segunda limitação.

Síntese do capítulo

Neste capítulo apresentamos alguns tópicos avançados em Arquitetura de Computadores. São técnicas modernas que tiveram como objetivo aumentar o desempenho dos computadores construindo novas alternativas arquiteturais.

A primeira técnica apresentada foi o Pipeline, em que unidades de processamento diferentes realizam operações em série da mesma forma que uma linha de produção. A outra técnica apresentada trata de processamento paralelo, envolvendo todas as formas de paralelizar o processamento de instruções e dados com o objetivo de aumentar o desempenho dos sistemas.

Atividades de avaliação

1. Consulte a lista dos maiores 30 computadores do mundo (vide Top500) e calcule as diversas arquiteturas utilizadas e faça uma tabela com o percentual de cada tipo.

2. Pesquise sobre computação em nuvem e diga qual é a arquitetura paralela mais recomendada para essa aplicação.
3. Leia a especificação do processador Intel QuadCore e relacione todas as técnicas utilizadas para melhorar o desempenho.

Leituras, filmes e sites

Leituras

KIRK, David B. e HWU, Wen-mei W. **Programando para processadores paralelos:** uma abordagem prática à programação de GPU. Rio de Janeiro: Elsevier, 2011.

PITANGA, Marcos. **Construindo Supercomputadores com Linux.** 3. Ed. Rio de Janeiro: Brasport, 2008.

Sociedade Brasileira de Computação e Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informatica. **III Simpósio Brasileiro de Arquitetura de Computadores e Processamento Paralelo:** III SBAC-PP, Rio de Janeiro, RJ : Anais. Rio de Janeiro : Sbc, 1990.

Filmes

Técnica do Pipeline. Duração: 32min 19s. Aula de Arquitetura de Computadores - Técnica do Pipeline com o Professor José Eduardo. Acesso em: https://www.youtube.com/watch?v=XkHwAvm61sQ

Execução de uma instrução Load Word em Pipeline. Duração: 18min 44s. Aula que demonstra a execução de uma instrução Load Word em Pipeline. Acesso em: https://www.youtube.com/watch?v=NmDdrl9ALxg

Taxonomia de Flynn. Duração: 17min 18s. Aula que demonstra a taxonomia de Flynn arquitetura de sistemas paralelos e distribuídos. Acesso em: https://www.youtube.com/watch?v=53lw9N5pEJY

Introdução à Computação Paralela e Distribuída. Duração: 17min 27s. Aula que introduz o conceito de Computação paralela e distribuída. Acesso em: https://www.youtube.com/watch?v=FO0DV_cEYXs

Sites

Informações sobre Computação Paralela (em Inglês)

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

Tutorial sobre Pipeline (em Inglês)

http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html

Tutorial sobre arquitetura Super-escalar (em Português)

http://worldlingo.com/ma/enwiki/pt/Superescalar

Listagem dos 500 maiores super-computadores do mundo (em Inglês)

http://www.top500.org/

Referências

STALLINGS, W. **Arquitetura e Organização de Computadores** 5ª Ed. Editora: Prentice Hall, 2002.

HENNESSY, J. L. & PATTERSON, D. A. **Arquitetura de Computadores:** Uma Abordagem Quantitativa 3ª Ed. Editora: Campus, 2003.

Sobre o autor

Marcial Porto Fernandez: nasceu no Rio de Janeiro em 1964, é Engenheiro Eletrônico pela UFRJ (1988), tem Mestrado (1998) e Doutorado (2002) em Engenharia Elétrica, na área de Teleinformática na COPPE/UFRJ. Atualmente é pesquisador e professor adjunto do curso de Ciências da Computação na Universidade Estadual do Ceará (UECE), onde realiza pesquisas sobre redes de computadores, gerenciamento de redes, qualidade de serviços e redes móveis.