CAPÍTULO 2 – ORGANIZAÇÃO DE COMPUTADORES

2.1 Organização de um Computador Típico

- Memória: Armazena dados e programas.
- Processador (CPU - *Central Processing Unit*): Executa programas armazenados na memória, interpretando suas instruções, ou seja, buscando as instruções na memória, decodificando-as e executando-as, uma após a outra.
- Dispositivos de Entrada e Saída (E/S ou I/O - *Input/Output*): estabelecem comunicação com o mundo externo (operador ou outros dispositivos).
- Barramento: Conjunto de conexões elétricas/lógicas paralelas que permite a transmissão de dados, endereços e sinais de controle entre os diversos módulos funcionais do computador.

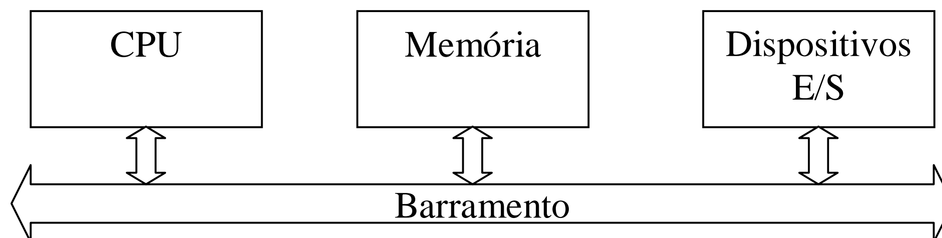


Figura 2.1. Organização de um computador simples.

2.2 Organização do Processador

- Memória de Rascunho (MEM RASC): conjunto pequeno (algumas dezenas) de registradores dedicados rápidos para armazenamento temporário de dados relativos à decodificação e execução de instruções.
- Unidade Lógica Aritmética (ULA): Circuito lógico combinacional que realiza operações booleanas sobre palavras armazenadas na memória de rascunho e armazena o resultado na mesma.
- Vias Internas: Barramentos dedicados que permitem a transmissão de dados da memória de rascunho para a ULA e vice-versa.
- Caminho de Dados: Memória de Rascunho + ULA + Vias Internas.
- Unidade de Controle (UC): Circuito lógico seqüencial responsável pela geração dos sinais de controle do Caminho de Dados na seqüência adequada para implementar interpretação de instruções.

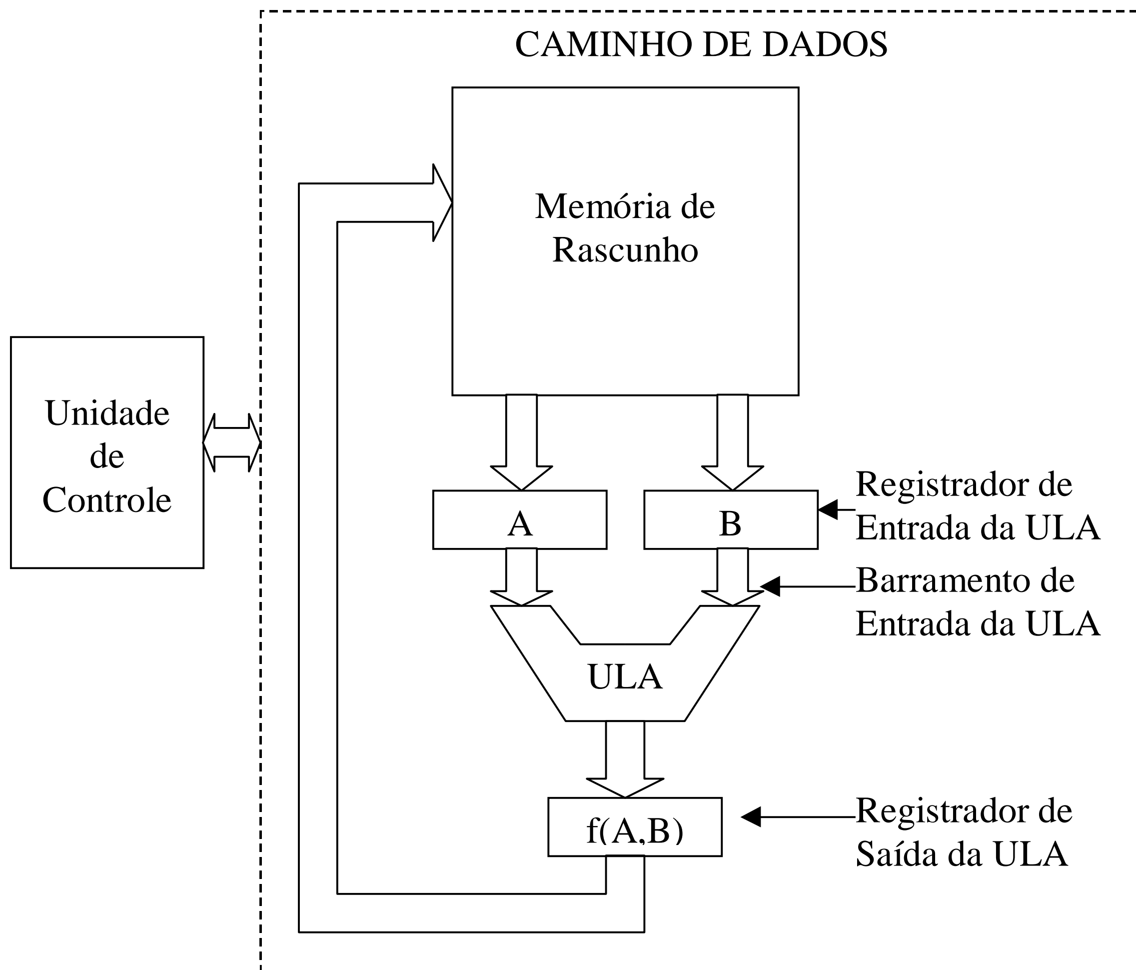


Figura 2.2. Organização do Processador e do Caminho de Dados.

- Operação do Caminho de Dados (Ciclo de Máquina):
  - Leitura dos registradores da memória de rascunho.
  - Escrita nos registradores de entrada da ULA.
  - A operação da ULA correntemente selecionada é executada.
  - O resultado do processamento da ULA é escrito no registrador de saída da ULA.
  - O registrador de saída da ULA é lido.
  - O seu conteúdo é copiado para o registrador de destino na memória de rascunho.

2.3 Interpretação de Instruções

- Conjunto de Instruções: Todas as instruções disponíveis ao programador de um dado nível de máquina virtual. (Nível de linguagem de máquina: tipicamente, de 20 a algumas centenas de instruções).
- Contador de Programa (PC – *Program Counter*): Registrador da memória de rascunho, ponteiro que armazena o endereço na memória principal onde se localiza a próxima instrução a ser interpretada.
- Registrador de Instrução (IR – *Instruction Register*): Registrador da memória de rascunho que armazena a Instrução corrente que foi buscada na memória principal.

- Interpretação de Instruções – Ciclo de Busca-Decodificação-Execução:

  i. Busca da próxima instrução no endereço da memória principal apontado pelo PC e armazenamento da mesma no IR.
  ii. Atualização do PC, fazendo-o apontar para a instrução seguinte (PC := PC +1).
  iii. Determinação do tipo de instrução armazenada no IR.
  iv. Se a instrução precisa de operandos armazenados na memória principal, os seus endereço devem ser determinados.
  v. Caso necessário, busca de operandos na memória principal.
  vi. Execução da instrução.
  vii. Retorno ao passo i para iniciar a execução da instrução seguinte.

Observação: O ciclo de busca-decodificação-execução pode ser implementado em *hardware* ou *software* (interpretador = microprograma).

- Características da interpretação em *software*:

  - Deve-se projetar um *hardware* para rodar o interpretador.
  - O projeto do hardware é simplificado ⇒ redução de custo.
  - Complexidade concentrada no interpretador.
  - Instruções complexas podem ser adicionadas facilmente.
    - Instruções mais complexas levam a execução mais rápida embora individualmente possam ser mais lentas.
    - Seqüências freqüentes de instruções simples são candidatas a serem codificadas numa única instrução complexa.
    - Em máquinas de alto desempenho, instruções complexas podem ser implementadas em *hardware*, mas a compatibilidade exige que instruções complexas sejam incluídas em máquinas simples.

- Novas instruções podem ser incorporadas facilmente.
- O projeto estruturado permite desenvolvimento, teste e documentação fáceis e eficientes.
- Erros de implementação podem ser corrigidos no campo.
- Novas máquinas de uma mesma família podem ser projetadas e colocadas no mercado rapidamente.
- Memórias ROM velozes (Memória de Controle) devem ser utilizadas para armazenar o microcódigo (microprograma, constituído por microinstruções).

- Máquinas CISC x Máquinas RISC

A "inflação" dos conjuntos de instruções interpretadas por microprogramas e a resultante perda de desempenho levaram à busca de diretrizes de projeto que priorizassem a eficiência na execução de instruções.

CISC ***Complex Instruction Set Computer***	**RISC** ***Reduced Instruction Set Computer***
Princípio: microprograma interpreta instruções complexas. Microinstruções primitivas que compõem o microprograma são executadas por um *hardware* simples.	Princípio: Conjunto de instruções constituído por instruções simples de uso muito freqüente. Instruções mais complexas devem ser implementadas como combinação de instruções simples
Projeto visa simplificar o *hardware* e diminuir o fosso semântico entre as linguagens de alto nível e a linguagem de máquina.	Projeto visa um melhor desempenho. Através da execução eficiente de instruções.
Complexidade concentrada no projeto do microprograma.	Complexidade concentrada no projeto do compilador.
Conjunto grande (centenas de instruções). Poucas instruções de uso frequente, muitas raramente usadas.	Conjunto pequeno (não mais do que três dezenas de instruções). Todas as instruções são de uso freqüente.
Instruções complexas, interpretadas por microprograma.	Instruções primitivas simples, interpretadas pelo hardware.
Execução de uma instrução demora vários ciclos de máquina.	Execução de uma instrução em um único ciclo de máquina.
Compatibilidade fácil de ser mantida.	Compatibilidade difícil de ser mantida.

Figura 2.3. CISC x RISC.

2.4 Princípios de Projeto na Atualidade (princípios "RISC"):

- Todas as instruções devem ser executadas diretamente pelo *hardware*.
  - Observação: Para máquinas que seguem a filosofia CISC, solução híbrida: núcleo "RISC" executado diretamente pelo *hardware* em um ciclo de máquina, instruções complexas executadas interpretadas.
- Maximização a taxa de execução de instruções. Explorar ao máximo as possibilidades de paralelismo, (desde que a ordem de execução assim o permita).
- Fácil decodificação das instruções. Explorar regularidade e simplicidade de formato de instruções.
- Referência à memória apenas através de instruções LOAD e STORE. Instruções devem operar sobre registradores, minimizando atrasos devidos a referências à memória principal.
- Disponibilidade de um grande número de registradores. Evitar ao máximo referências à memória principal.

2.5 Execução Paralela de Instruções

Lei de Moore: "O número de transistores integrados em um *chip* dobra a cada dezoito meses".

Limites Físicos: Velocidade da luz ≅ 20 cm/ns no cobre. Solução: miniaturizar para diminuir tempo de propagação dos sinais no *chip*. Problema: efeitos de dissipação térmica (efeito Joule) potencializados com o aumento da densidade de componentes no *chip*.

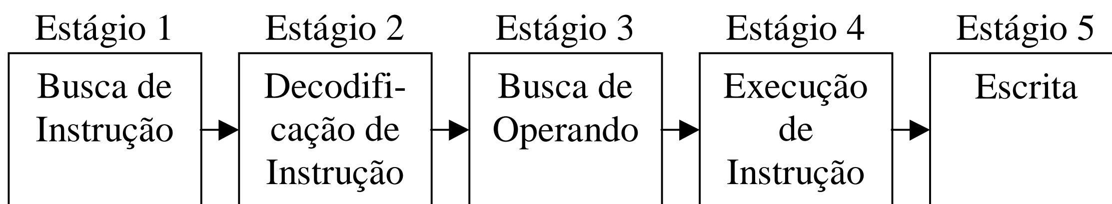
Paralelismo no Nível de Instruções:

- Paralelismo dentro das instruções individuais (sem alterar a seqüência), de modo a que o processador execute mais instruções por segundo.
- Instrução Única - Dado Único (SISD - *Single Instruction Single Data*).

- ***Pipeline***:

  - Acesso à memória principal é o "gargalo" na execução de instruções.
  - Solução simples: Pré-Busca (*Pre-Fetch*).
    - Instruções buscadas antecipadamente e armazenadas em um *buffer* de pré-busca.
    - Próxima instrução a ser executada obtida a partir do *buffer*.
    - Fases de Pré-Busca e Execução realizadas em paralelo.

- Solução Aprimorada: *Pipeline*.
    - Ciclo de busca-decodificação-execução dividido em etapas, processadas simultaneamente por unidades (estágios) de *hardware* dedicadas.
    - Exemplo: considerando um *pipeline* de cinco estágios, enquanto uma instrução está sendo buscada (pelo estágio de busca), a anterior já está sendo decodificada (pelo estágio de decodificação). Por sua vez, os operandos da instrução anterior a esta estão sendo buscados (pelo estágio de busca de operandos). Simultaneamente, a instrução anterior a esta está sendo executada (no estágio de execução). Ao mesmo tempo, os resultados da execução da instrução anterior a esta última são escritos em registradores (pelo estágio de escrita).



Clock→ Estágio	1	2	3	4	5	6	7
E1	1	2	3	4	5	6	7
E2		1	2	3	4	5	6
E3			1	2	3	4	5
E4				1	2	3	4
E5					1	2	3

Figura 2.4. *Pipeline* de cinco estágios.

- Latência: tempo de execução de uma instrução = n.T, onde n é o número de estágios do estágios do *clock* e T é o ciclo do clock.
- Banda Passante: número de instruções executadas por segundo (unidade: MIPS - Milhões de Instruções por Segundo).
- Idealmente, Banda passante = 1000/T MIPS, (com T em ns).
- Instruções de desvio prejudicam o desempenho do *pipeline*.
- Podem ser utilizados dois (ou mais) *pipelines* em paralelo. (Compilador ou *hardware* deve garantir que se executem apenas pares de instruções em que uma não depende da outra).

- **<u>Arquiteturas Superescalares</u>**:

  - Princípio: a fase de execução geralmente demora consideravelmente mais que as outras fases da interpretação.
  - Aplicação: <u>Arquitetura Superescalar</u>. *Pipeline* único, mas dotado de vários estágios de execução trabalhando em paralelo.
  - O ganho de desempenho se deve a que os estágios anteriores do *pipeline* são capazes de distribuir instruções a uma velocidade bem maior do que a capacidade de executá-las do estágio de execução.

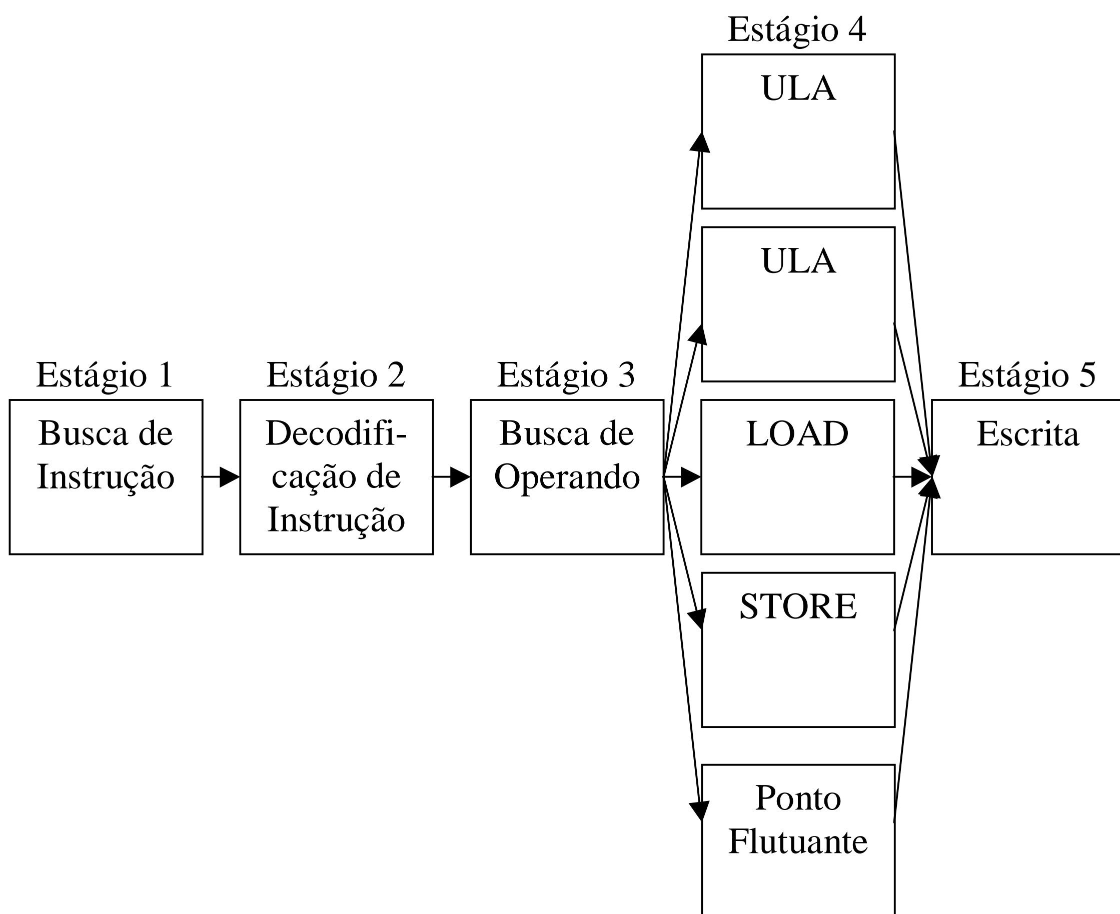


Figura 2.5. Processador Superescalar com Unidades de Execução.

<u>Paralelismo no Nível de Processador</u>:

- Vários processadores trabalhando em paralelo.
- Ao contrário do paralelismo no nível de instrução, que consegue acelerar o desempenho em até, no máximo, dez vezes, o paralelismo em nível de processador permite obter ganhos da ordem de dezenas, centenas, ou mais ainda.

- **<u>Computadores Matriciais</u>:**

    - Instrução Única - Dados Múltiplos (SIMD - *Single Instruction Multiple Data*).

    - <u>Processadores Matriciais</u>:

        - Grande número de processadores, arranjados matricialmente, que executam a mesma seqüência de instruções sobre diferentes conjuntos de dados.
        - Uma unidade de controle distribui instruções em *broadcast* para todos os processadores.

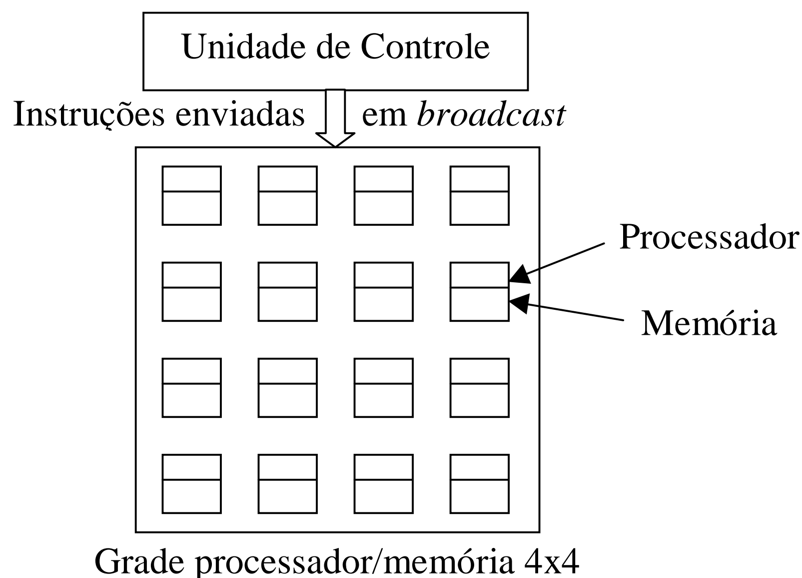


Figura 2.6. Processador Matricial.

- Processadores Vetoriais:

  - Parecidos com os processadores matriciais, vários processadores compartilham uma única unidade de controle
  - Eficientes na execução de uma mesma seqüência de instruções sobre pares de elementos de dados.
  - Mais lentos que os processadores matriciais, mais com um *hardware* bem mais simples (e mais barato).
  - Mais fáceis de serem programados do que os processadores matriciais.
  - Diferentemente dos processadores matriciais, as operações lógicas e aritméticas são realizadas por uma única ULA que opera em *pipeline*.
  - Baseados em ULA Vetorial opera sobre Registradores Vetoriais (registradores que podem ser carregados por meio de uma única instrução.
  - O processamento vetorial pode ser facilmente incorporado a processadores convencionais. Instruções que podem ser vetorizadas são assim executadas bem mais rapidamente.

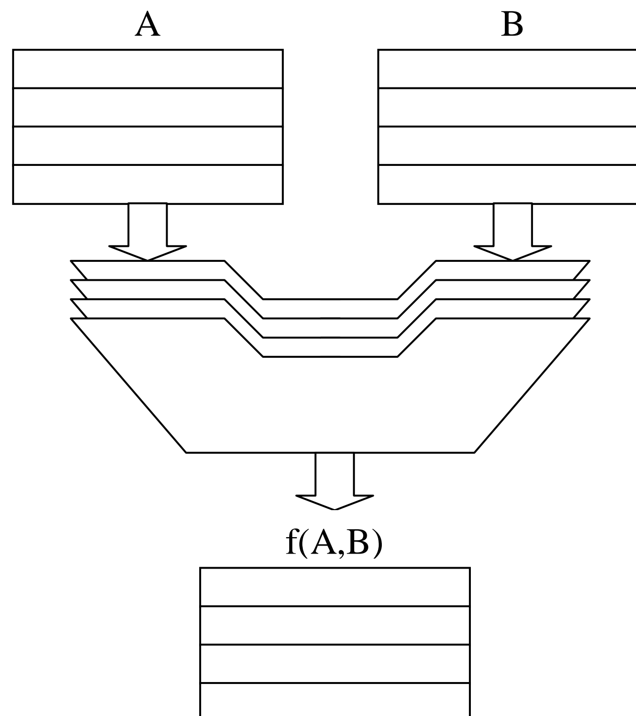


Figura 2.7. ULA vetorial.

- Multiprocessadores:

  - Ao contrário dos computadores matriciais, que compartilham uma única unidade de controle, os multiprocessadores são compostos por processadores independentes, cada qual com a sua própria unidade de controle.
  - Instruções Múltiplas - Dados Múltiplos (MIMD - *Multiple Instruction Multiple Data*).
  - Processadores compartilham a mesma memória através de um barramento comum.
  - Devem implementar técnicas que garantam a integridade dos dados compartilhados.
  - Tipicamente, o número de processadores é menor do que 64. Para números maiores, aumenta muito a probabilidade de conflito no acesso aos dados, derrubando o desempenho.
  - Processadores podem incluir alguma memória local, de uso exclusivo, para armazenar dados e programas que não precisam ser compartilhados, de modo a minimizar conflitos.
  - O modelo de memória compartilhada torna a sua programação extremamente fácil.

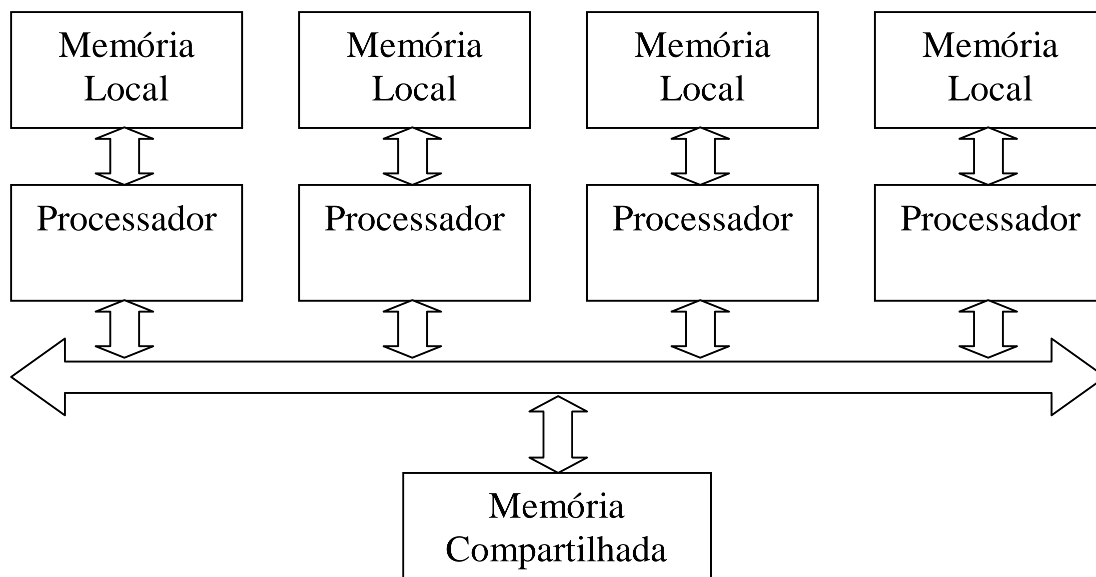


Figura 2.8. Multiprocessador

- Multicomputadores:

  - Ao contrário dos computadores matriciais, que compartilham uma única unidade de controle, os multicomputadores são compostos por processadores independentes, cada um com a sua própria UC.
  - Diferentemente dos multiprocessadores, não compartilham memória (LOAD e STORE só acessam memória local).
  - São mais fáceis de construir do que os multiprocessadores.
  - São mais difíceis de programar do que os multiprocessadores.
  - Constituídos por um grande número de nós, (ex.: 10.000 nós).
  - Cada nó é constituído por um ou mais processadores dotados de memória RAM local, memória secundária, dispositivos de entrada e saída e um processador de comunicação.
  - Através dos seus processadores de comunicação, os nós são interligados por uma rede de interconexão de alto desempenho.
  - Os nós se comunicam por um sistema de troca de mensagens (primitivas *send* e *receive*).
  - O número elevado de nós torna inviável a ligação de cada nó com todos os demais. É necessário o uso de topologias tais como grades 2D ou 3D, árvores, anéis, etc., aliadas a técnicas de roteamento de mensagens.
  - Duas grandes categorias:
    - Processadores Fortemente Paralelos (MMPs - *Massively Parallel Processors*): baseados em redes de interconexão proprietárias de alto desempenho.
    - Agrupamentos de Estações de Trabalho (COWs - *Cluster of Workstations*): baseados em componentes comerciais conectados através de rede comercial.

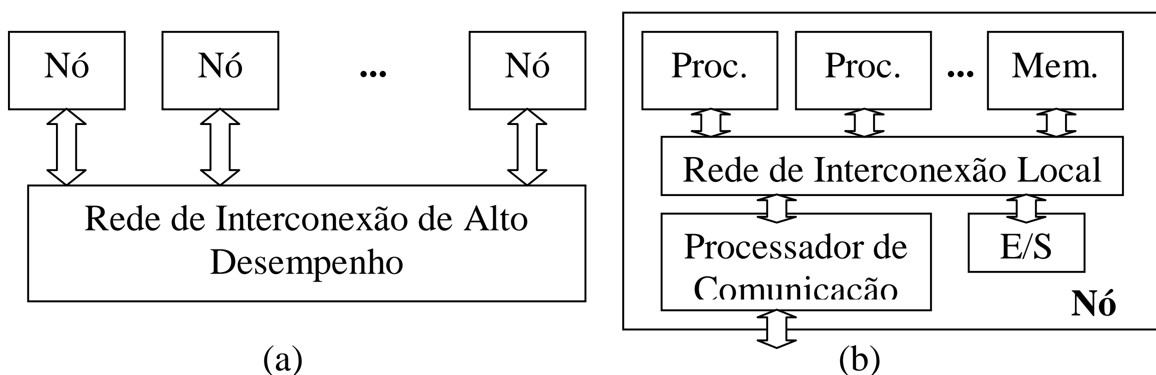


Figura 2.9. a) Multicomputador típico. b) Nó típico.

2.6 Organização de Memória

Hierarquia de Memória:

- Para uma maior eficiência na referência à memória, a mesma é estruturada hierarquicamente.
- Informações referidas com maior freqüência são trazidas "mais perto" da CPU, armazenadas em memórias mais rápidas, (também mais caras, o que implica em menor capacidade de armazenamento).

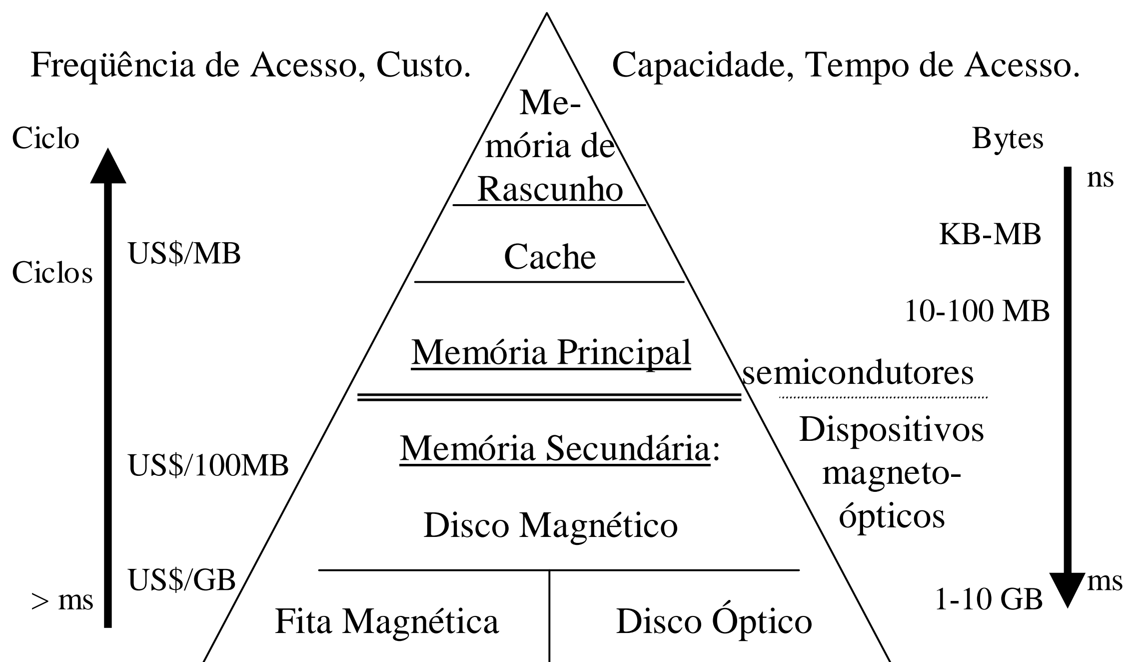


Figura 2.10. Hierarquia de Memória.

Memória de Rascunho:

- Memória semicondutora de pequena capacidade (no máximo, algumas dezenas de registradores), localizada na CPU.
- Armazena informações relativas à interpretação da instrução de linguagem de máquina corrente.
- Em máquinas microprogramadas, é o espaço de endereçamento acessível às microinstruções.
- Memória rápida (e cara). Tempo de acesso da ordem de nanossegundos.
- Freqüência de acesso altíssima. Acessada a cada ciclo de máquina.

<u>Memória Principal</u>:

- Memória semicondutora que armazena os dados e programas em linguagem de máquina em execução corrente.
- Razoavelmente barata.
- Tempo de acesso da ordem de nanossegundos a dezenas de nanossegundos.
- Freqüência de acesso alta. Se não existisse a memória Cache, seria acessada a cada ciclo de busca-decodificação-execução.

- <u>Unidade básica de armazenamento</u>:

  - Bit (*Binary digIT*): assume dois estados possíveis (0 ou 1), quantidade mínima para distinguir duas informações diferentes.
  - Eficiente do ponto de vista de implementação físico: dispositivos capazes de detectar um dentre dois estados diferentes são confiáveis.

- <u>Endereçamento</u>:

  - A memória principal é organizada como um conjunto de n <u>células</u> (ou <u>posições</u>) capazes de armazenar, cada uma, m bits.
  - Cada célula é identificada por um <u>endereço</u> (código binário associado) de k bits através do qual é referenciada.
  - Os endereços são numerados de zero a n-1.
  - Existem 2^k possíveis endereços. Assim, o máximo número de células endereçáveis é 2^k.
  - A célula é a menor unidade de memória endereçável.
  - Uma célula poderá armazenar qualquer uma das 2^m possíveis combinações diferentes dos seus m bits. m é independente de n.
  - Os m de bits de uma célula são acessados simultaneamente.
  - m pode ser qualquer número inteiro mas, nos últimos anos, os fabricantes padronizaram um tamanho de 8 bits (1 byte).
  - A capacidade de armazenamento de uma memória é C = n x m bits (ou C = n x m/8 bytes).
  - Uma memória de C bits pode ser organizada de diversas maneiras. Exemplo: 96 bits = 12 x 8 ou 8 x 12 ou 6 x 16, etc.
  - Bytes são agrupados em <u>Palavras</u>. A maioria das instruções opera sobre palavras. Registradores da memória de rascunho geralmente são do tamanho de uma palavra.

- Ordenação dos Bytes na Palavra:

  - Os bytes de uma palavra podem ser numerados da esquerda para a direita ou da direita para a esquerda.
  - Quando a numeração dos bytes começa da esquerda para a direita da palavra, a ordenação dos bytes é dita *Big Endian*.
  - Quando a numeração dos bytes começa da direita para a esquerda da palavra, a ordenação dos bytes é dita *Little Endian*.

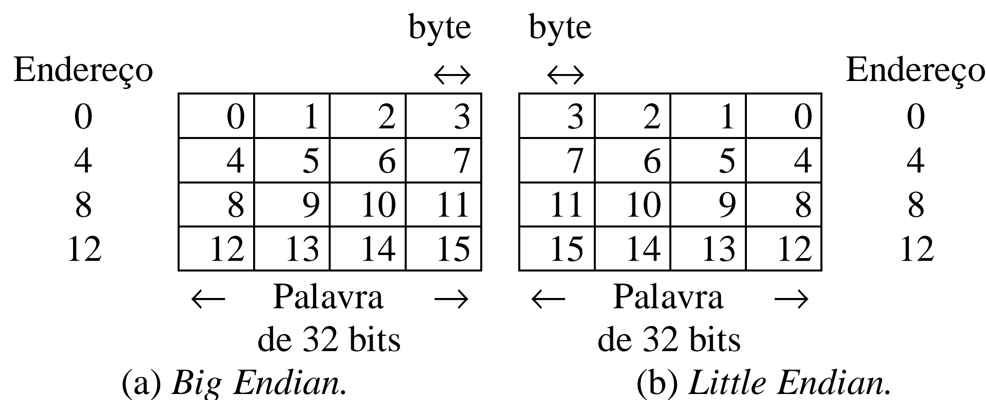


Figura 2.11. (a) Memória *Big Endian*. (b) Memória *Little Endian*.

  - Problema: ao transmitir informações de uma máquina *big endian* para uma *little endian*, (ou vice-versa), os bytes de uma palavra são invertidos (o seu valor numérico muda completamente).

- Detecção e Correção de Erros:

  - Para verificar a integridade dos dados armazenados na memória, um ou mais bits redundantes adicionais de paridade podem ser acrescentados à palavra.
  - Podem ser utilizados códigos para detecção ou mesmo para detecção e correção de um ou mais bits errados.
  - Como a probabilidade de erro nas memórias atuais é extremamente pequena (um em dez anos), na prática, os fabricantes de computadores não fazem uso deste recurso.

<u>Memória Cache</u>:

- Observações:
  - Por razões históricas, o aperfeiçoamento das memórias centrou-se no aumento da sua capacidade de armazenamento, enquanto a o aperfeiçoamento das CPU's centrou-se no seu desempenho.
  - A memória principal é um "gargalo" para a CPU, que deve esperar vários ciclos para ter atendidas as suas requisições à memória.
  - É possível projetar uma memória com velocidade compatível com a CPU, mas é muito caro (deve ser embutida dentro do *chip* do processador). Existem restrições ao aumento do tamanho do *chip*.
- Solução mais eficiente: Memória Cache, memória semicondutora rápida (e cara), mas de pequena capacidade, que, associada à memória principal, (barata e de grande capacidade), resulta numa memória razoavelmente barata, razoavelmente rápida e de grande capacidade.

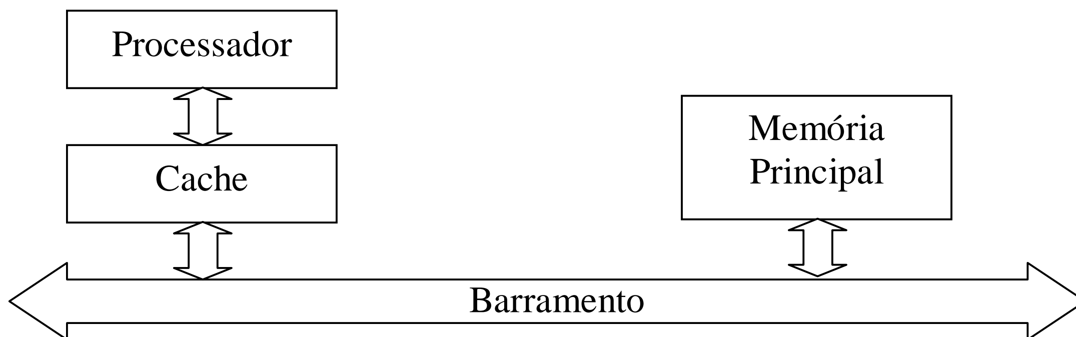


Figura 2.12. Conexão Lógica entre CPU, Cache e Memória Principal.

- Idéia básica: manter no Cache as palavras mais usadas pelo processador. Se a maior parte dos acessos for satisfeita pelo Cache, o tempo médio de acesso será próximo do tempo de acesso ao Cache, que é pequeno.
- Constatação: o acesso à RAM não é totalmente aleatório. referências à memória num certo intervalo de tempo pequeno tendem a acessar uma pequena parte da memória total.
- <u>Princípio da Localidade</u>: Existe grande probabilidade que palavras próximas a uma palavra recentemente referenciada também sejam referenciadas nos próximos acessos.

- Aplicação: memória Cache. Quando uma palavra é referenciada pelo processador, se ela não estiver no Cache, ela é trazida para o mesmo junto com palavras de endereços vizinhos na memória principal. Em novas referências, o tempo de acesso a essas palavras será bastante reduzido, pois serão acessadas rapidamente no Cache.
- Dados: m = tempo de acesso à Cache, c = tempo de acesso à memória principal, t = tempo médio de acesso. k = número de referências a uma determinada palavra.
  - Taxa de acertos = h = (k-1)/k.
  - t = c + (1 - h).m.
  - Observação: se $h \to 1$, $t \to c$; se $h \to 0$, $t \to c+m$.
- Cache e memória principal são divididas em blocos de endereços de tamanho fixo (Linhas de Cache).
- Sempre que uma palavra procurada não estiver no Cache (falha de acesso à Cache), a linha correspondente é trazida da memória principal.
- É mais eficiente trazer k palavras de uma vez do que uma palavra k vezes.
- Cache unificada: instruções e dados usam a mesma Cache. Mais simples de projetar.
- Caches divididas (arquitetura Harvard): utiliza uma Cache para instruções e uma Cache para dados. Permite paralelizar a busca de dados e instruções em processadores *pipeline*. Como as instruções não são modificadas, o Cache de instruções não precisa ser atualizado na memória principal.
- A Cache pode ser estruturada em vários níveis: Cache Primária (dentro do *chip*), Cache Secundária (fora do *chip*, mas no mesmo invólucro), Cache Terciária (totalmente dissociada da CPU).

Memória Secundária:

- Memória de grande capacidade (dezenas de Gigabytes).
- Armazenamento massivo.
- Implementada em meio magnético (*hard disk*, fitas magnéticas) ou ótico (CD-ROM, DVD-ROM).
- Armazena programas e dados não processados correntemente, mas que poderão eventualmente ser utilizados. (freqüência de acesso pequena).
- Memória lenta e barata. Tempo de acesso da ordem de milissegundos.
- Pode também ser utilizada para emular memória principal, “aumentando” o espaço de endereçamento disponível através de técnicas de memória virtual.

2.7 Organização de Entrada e Saída

Arquitetura de Entrada e Saída:

- Barramento: meio de transmissão de dados entre a CPU, a memória principal e os dispositivos de entrada e saída. Compartilhado por todos.
- Controlador de Dispositivo: conjunto de circuitos lógicos de interface entre o barramento e o dispositivo de entrada/saída. Responsável por controlar o dispositivo e por tratar do seu acesso ao barramento.

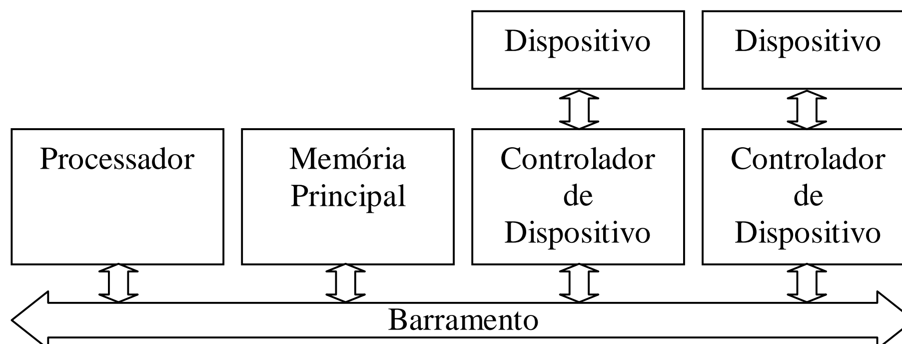


Figura 2.13. Estrutura lógica de um computador simples.

- Exemplo de procedimento de E/S:
  - Programa que precisa de dados de um dispositivo envia comando ao controlador correspondente.
  - Controlador envia os comandos necessários ao *drive* do disco.
  - O disco envia os dados requisitados ao controlador.
  - Controlador organiza os dados recebidos e os escreve na memória.
    - Acesso Direto à Memória (DMA – *Direct Memory Access*): processo de leitura ou escrita na memória principal por parte do controlador de dispositivo sem a intervenção do processador.
  - Concluída a transferência, o controlador sinaliza ao processador através de uma Interrupção. Processador interrompe o programa corrente e executa Rotina de Tratamento de Interrupção, que verifica possíveis erros, encerra a operação e informa ao Sistema Operacional o seu fim. A seguir, o processador retoma o programa interrompido.
- Árbitro de Barramento: *chip* que controla o acesso ao barramento, resolvendo conflitos quando há tentativas de acesso simultâneo.
- Dispositivos mais rápidos têm maior prioridade. CPU tem a menor prioridade. O uso do barramento por dispositivos de E/S resulta no Roubo de Ciclos de barramento do processador, reduzindo o desempenho.