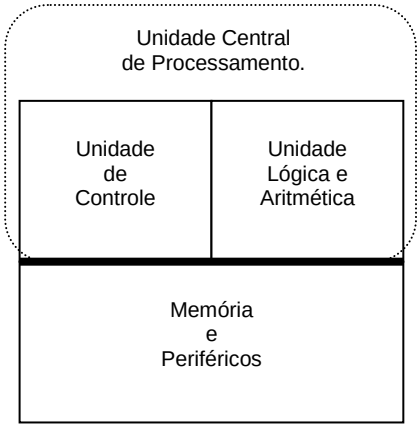
Modelos de arquitetura de computadores

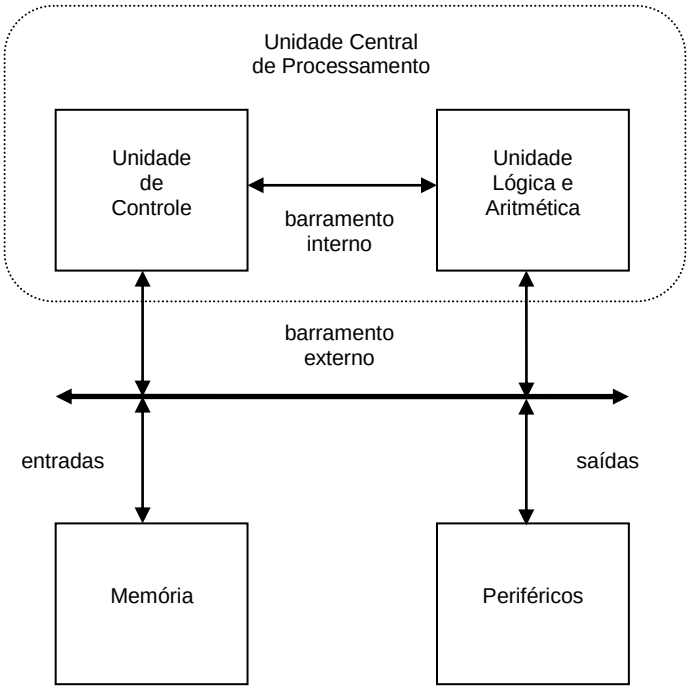
Um modelo simplificado de uma arquitetura genérica para computador procura agregar os seguintes componentes principais:

- Unidade Central de Processamento (UCP)
  - Unidade de Controle
  - Unidade Lógica e Aritmética
- Memória
- Periféricos



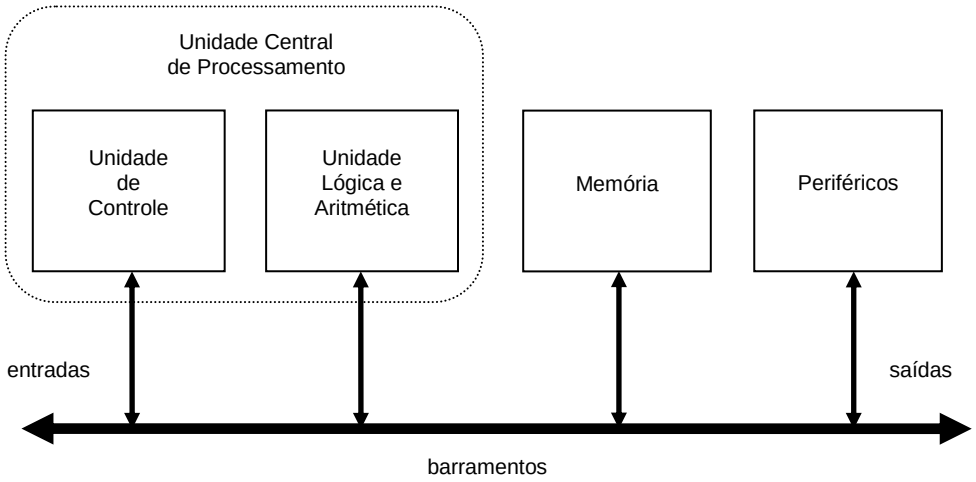
Organização de computador (1).

Esses componentes são conectados por vias ou barramentos, por onde passam as informações entre eles.



Organização de computador (2).

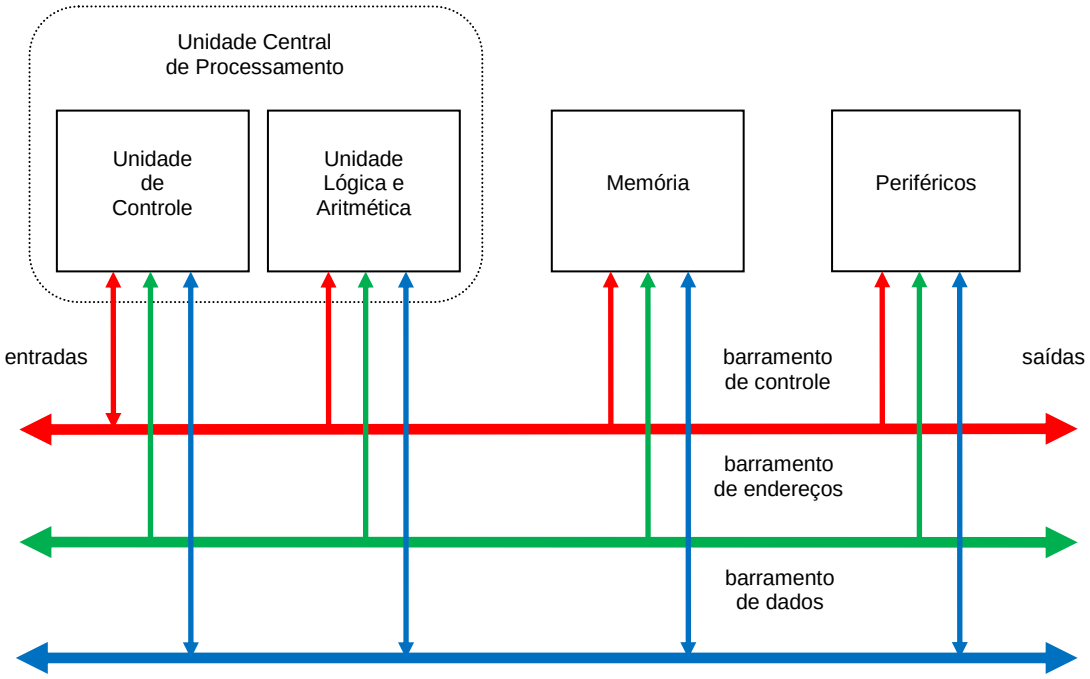
Embora a distinção entre barramento interno e externo exista, e a comunicação entre os componentes na Unidade Central de Processamento seja favorecida pela disponibilidade de um barramento interno exclusivo, a generalização das conexões por barramentos será empregada a seguir.



Os barramentos podem ser separados segundo especificidades em:

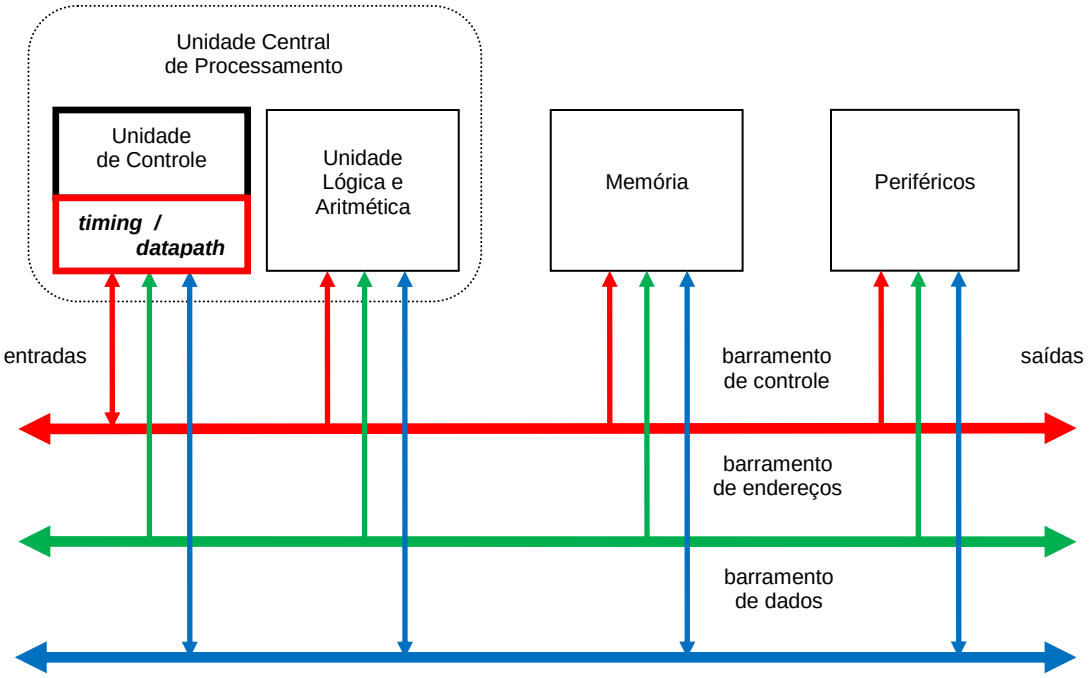
- de controle
- de endereços
- de dados

Organização de computador (3).



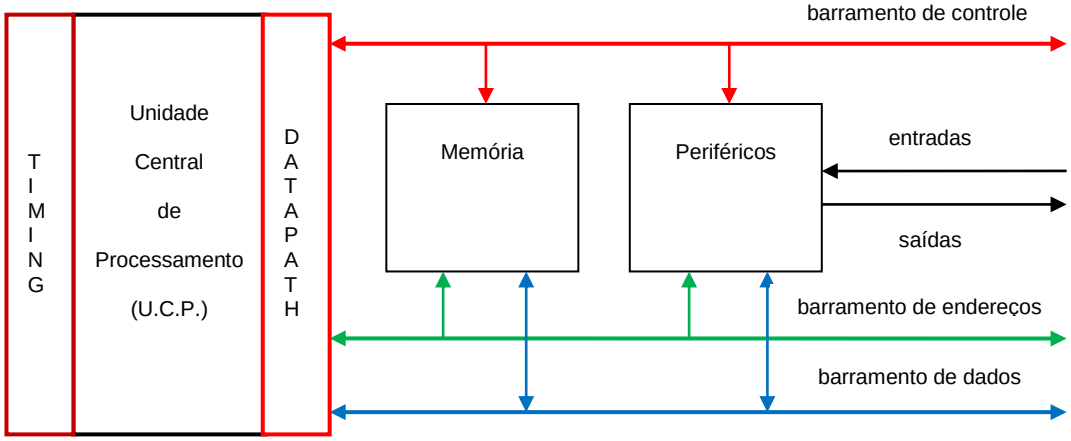
Organização de computador (4).

A Unidade de Controle conta com elementos adicionais para ajustar a sincronização das transferências (*timing*) entre componentes e para administrar os sinais para ativação/desativação de cada um deles (*datapath*).



Organização de computador (5).

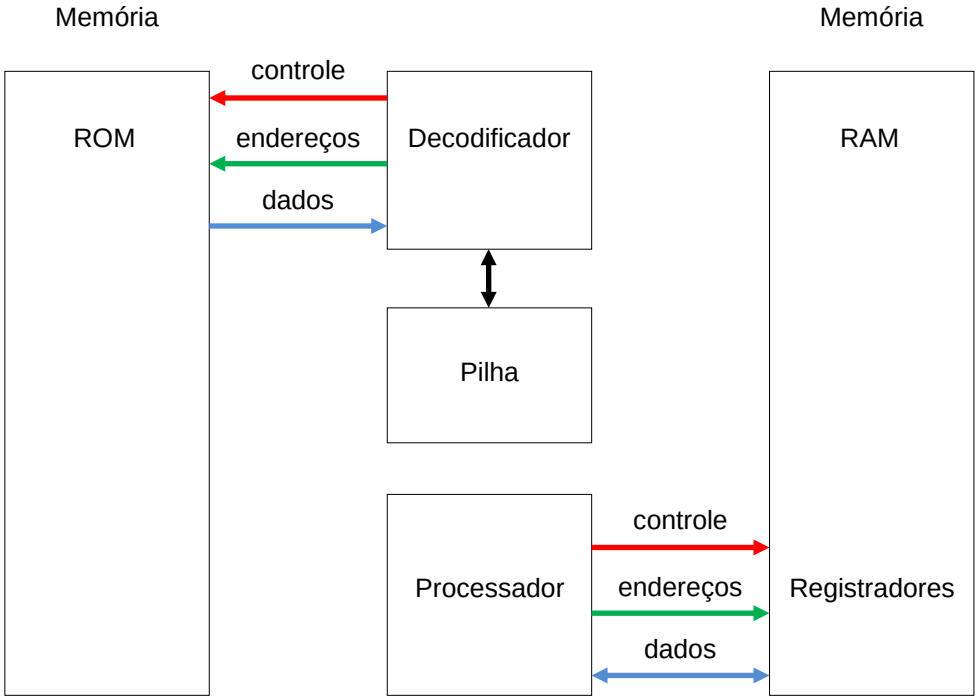
O modelo a seguir dispõe os elementos para permitir futuras expansões.



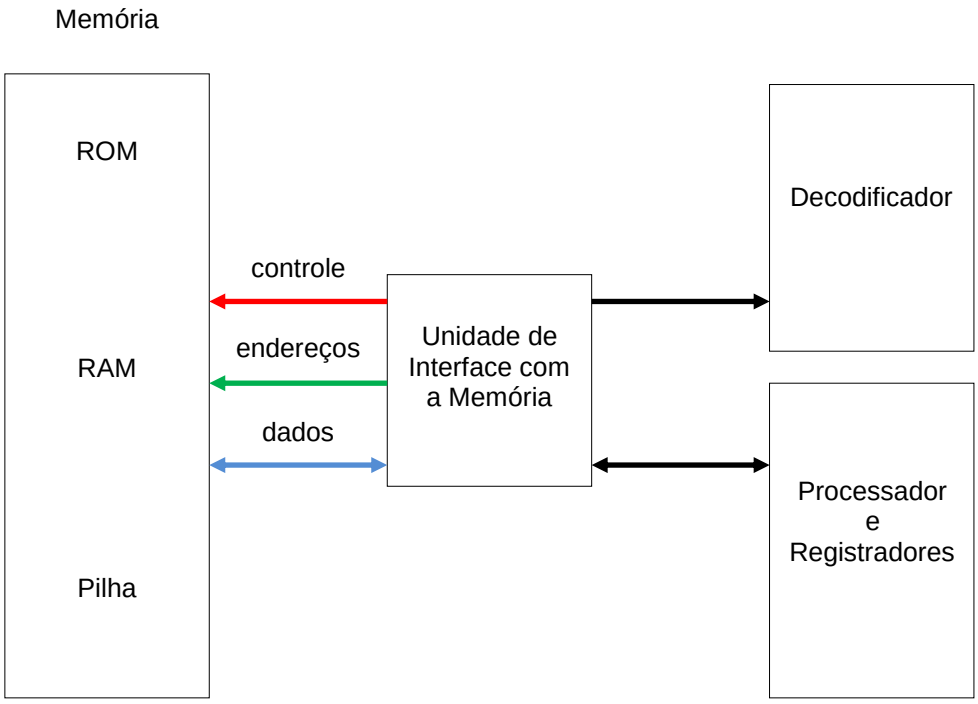
Organização de computador (6).

Modelos de arquiteturas de computador sequencial

Arquitetura de Computador
(Harvard)



Arquitetura de Computador
(Princeton / von Neumann)



Comparação entre modelos de arquiteturas de computador sequencial

Harvard	**Princeton**
memória com código e dados com barramentos separados	memória com apenas um barramento
potencialmente mais eficiente (paralelismo) menor quantidade de ciclos de memória para se executar instruções	instruções e dados são tratados iguais uso de caches de instruções e dados pré-buscas de instruções e dados
modelo de projeto de processador com mais detalhes	modelo de projeto de processador mais simples
exemplo: Microchip PIC	exemplo: Motorola 68HC11 80x86/Pentium (exceto IO)

Comparação entre os principais tipos de arquitetura de conjunto de instruções (ISA's)
segundo o conjunto de instruções

CISC (*Complex Instruction Set Computer*)	**RISC** (*Reduced Instruction Set Computer*)
conjunto numeroso e variado de instruções	conjunto simples e pequeno de instruções
conjunto único de registradores (16)	múltiplos conjuntos de registradores (256)
fracamente paralelizado	altamente paralelizado (*pipeline*)
tempos de execução diferentes	quase o mesmo tempo de execução
instruções com múltiplos ciclos (mais lentas)	instruções com menos ciclos (mais rápidas)
instruções com tamanho variado	instruções com tamanho fixo
instruções típicas com 1 ou 2 operandos	instruções típicas com 3 operandos
complexidade no código	complexidade no compilador
acesso à memória por várias instruções	acesso à memória por poucas instruções
múltiplos modos de endereçamento	poucos modos de endereçamento
controle microprogramado	controle embutido no *hardware*
menos ortogonal (instruções específicas e pouco usadas)	mais ortogonal (mais potência e maior flexibilidade)

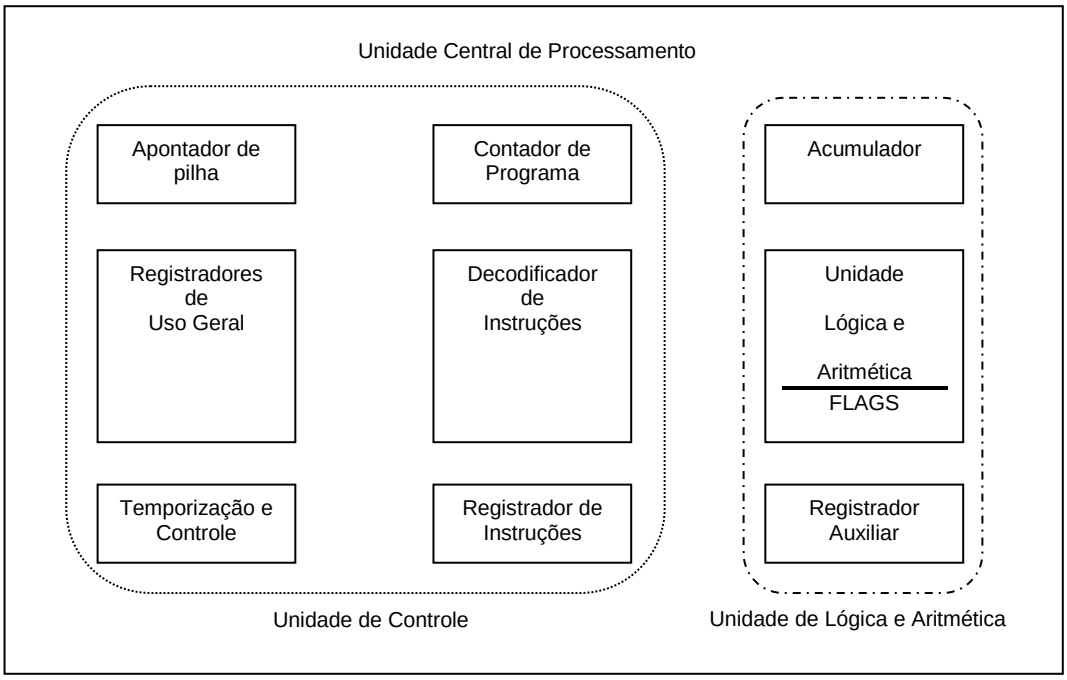
Combinações dos tipos acima constituem arquiteturas híbridas e servem:

- para possibilitar melhor compatibilidade entre *hardware* e *software*;
- para possibilitar a construção e uso de diferentes linguagens;
- para aumentar a capacidade de processamento.

Classificação de tipos de arquitetura de conjunto de instruções (ISA's)
segundo a complexidade

ISA	Detalhes
CISC	**Complex Instruction Set Computer** muitas instruções especializadas, algumas raramente utilizadas
RISC	**Reduced Instruction Set Computer** processador simplificado com implementação eficiente apenas do que for mais usado
VLIW	**VLIW - Very Long Instruction Word** projetada para explorar o paralelismo no nível de instruções (ILP) compiladores mais complexos, hardware simplificado para aproveitar o paralelismo conceito inventado por Josh Fisher e sua equipe na Universidade de Yale, no começo dos anos 1980 Princípios: - identificar paralelismo além de um bloco básico e desenvolver um compilador capaz de organizar a ordem de execução - o compilador e a arquitetura alvo deverão ser desenvolvidos em conjunto Exemplos de implementações: Intel i860 (1990) HP PA-RISC (1990) SOC (*System-On-a-Chip*) GPU (*Graphics Processing Unit*) da Nvidia e da AMD/ATI
EPIC	**Explicity parallel Instruction Computing** termo cunhado em 1997 pela aliança entre HP e Intel para o desenvolvimento de uma arquitetura com maior independência entre o hardware e o software (compilador) Princípios: - explorar o paralelismo ao nível de instruções - eliminar a necessidade de circuitos complexos para organizar a ordem de execução de instruções em paralelo Exemplo: Intel Itanium (IA-64)
MISC	**Minimal Instruction Set Computer** conjunto bem pequeno de instruções básicas, geralmente orientadas para execução em pilha e com modo de endereçamento do tipo *load/store* Exemplos: Manchester Baby (University of Manchester, 1948) EDSAC (University of Cambridge, 1949) Mark1 (University of Manchester, 1949) EDVAC (Ballistic Research Laboratory, 1951) ORDVAC (University ou Illinois, 1951) IAS (Princeton University, 1952) MANIAC I (Los Alamos Scientific Laboratory, 1952) ILLIAC (University of Illinois, 1952)
OISC	**One Instruction Set Computer** máquina abstrata que tem apenas uma instrução, dispensando a necessidade de um código de operação em linguagem de máquina empregado no ensino de arquitetura de computadores e em modelos computacionais para pesquisas em computação estrutural Categorias máquinas para manipulação de bits máquinas ativadas por transporte (*TTA-Transport Triggered Architecture*) máquinas completas de Turing baseadas em aritmética Exemplos de instruções subtrair e desviar se menor ou igual a zero subtrair e desviar se diferente de zero subtrair e desviar se negativo subtrair e desviar se positivo

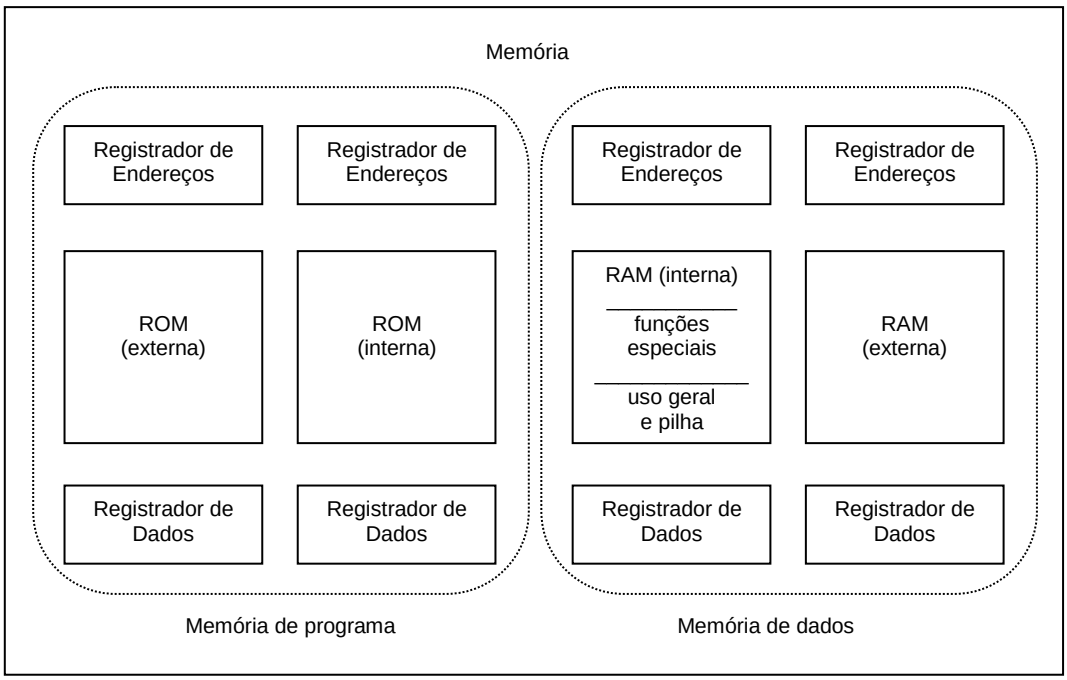
Os componentes principais para controle, operações e memória apresentam-se melhor agrupados por unidades funcionais.



Componentes principais para controle e operações

A unidade da Memória segundo sua constituição pode ser separada em:

- memória de programa (Read-Only Memory – ROM)
- memória de dados (Random-Access Memory – RAM)



Tipos de componentes de memória

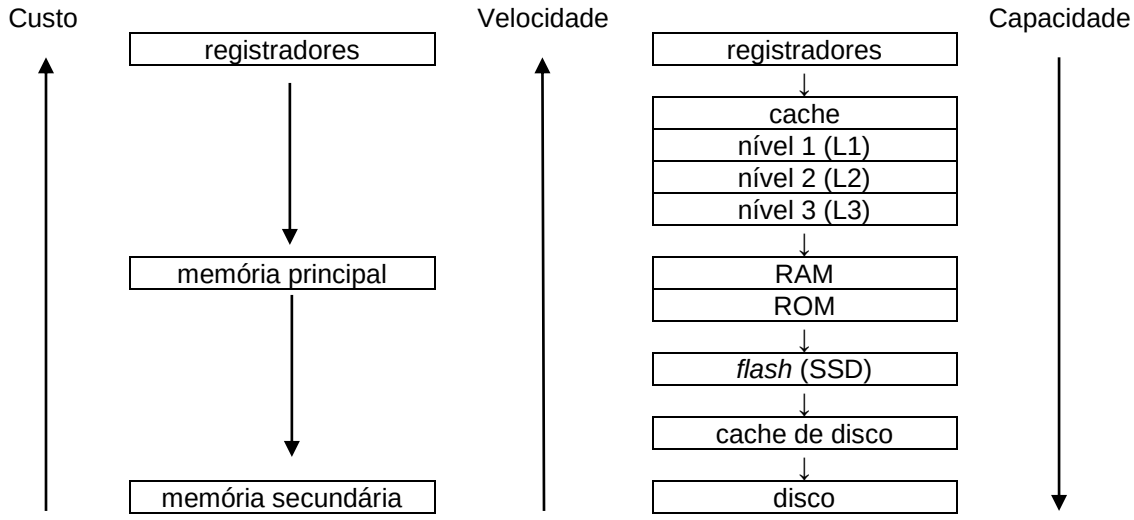
Tipos de memórias

Tipo de memória	Operação	Apagamento	Escrita	Volatilidade
RAM	leitura e escrita	elétrico/byte	elétrica	volátil (temporária)
ROM	leitura	(indisponível)	por máscara	não volátil (microprogramação)
PROM			elétrica	
EPROM	leitura (preferencial)	ultravioleta/*chip*		
EEPROM		elétrico/byte		
flash		elétrico/bloco		

As memórias do tipo RAM podem ser classificadas como dinâmicas ou estáticas.

RAM dinâmica	RAM estática
necessita refrescamento	dispensa refrescamento
construção simples	construção complexa
menor área/bit	maior área/bit
mais barata	mais cara
mais lenta	mais rápida
memória principal	memória cache

Hiearquia de memória

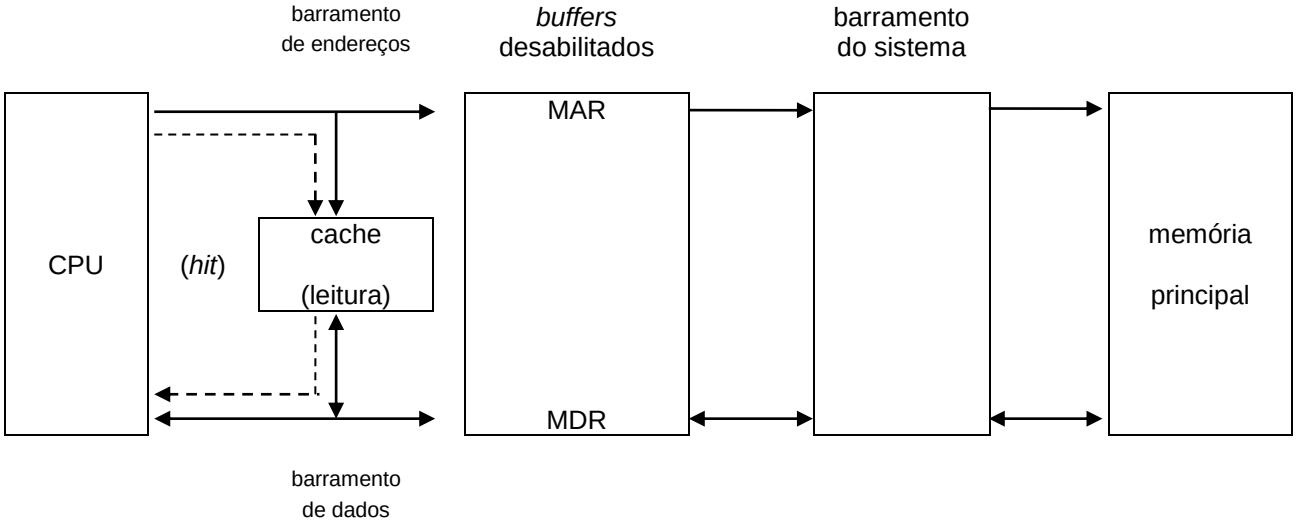


Memória cache

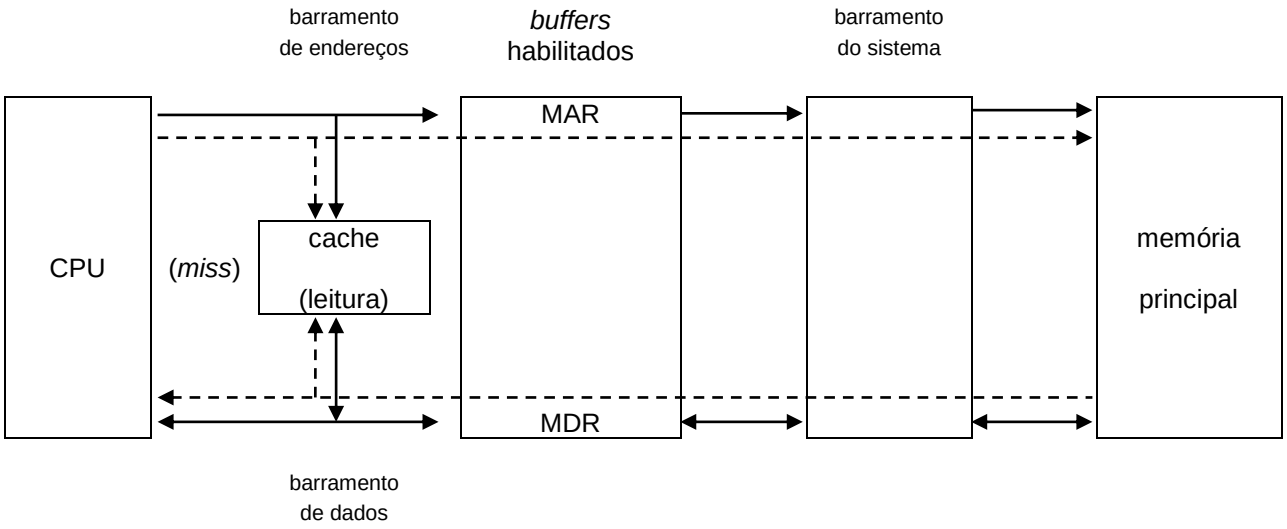
A memória cache serve como dispositivo auxiliar para melhorar a velocidade de acesso à informação.

Com capacidade menor que a da memória principal, guarda cópia de uma porção desta.

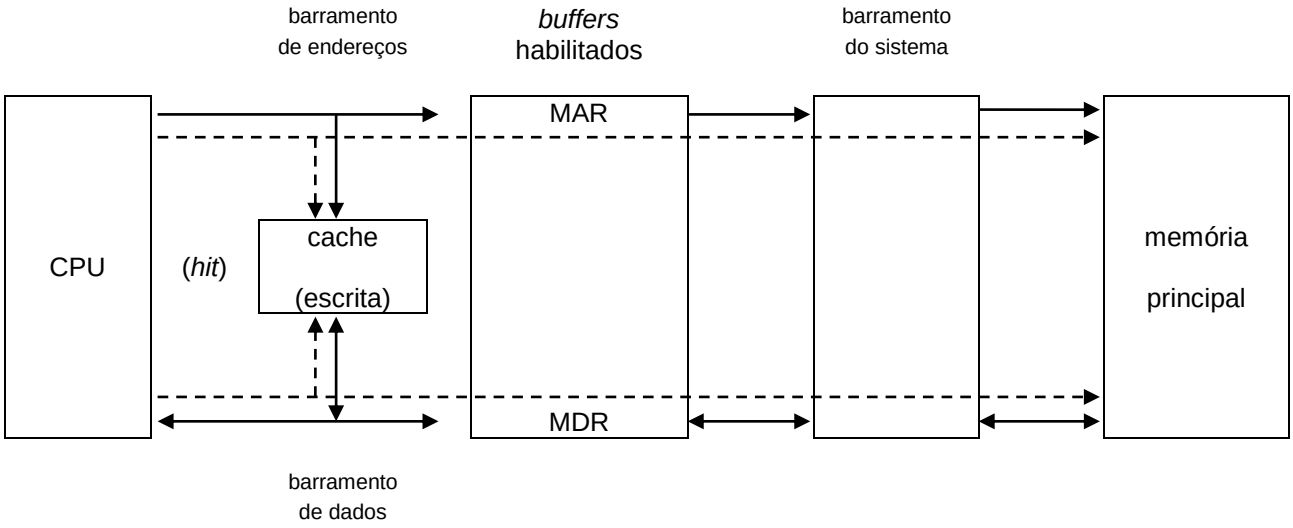
Se, durante uma operação de leitura, o conteúdo já estiver na mesma (*hit*), não é necessário buscá-lo na memória principal.



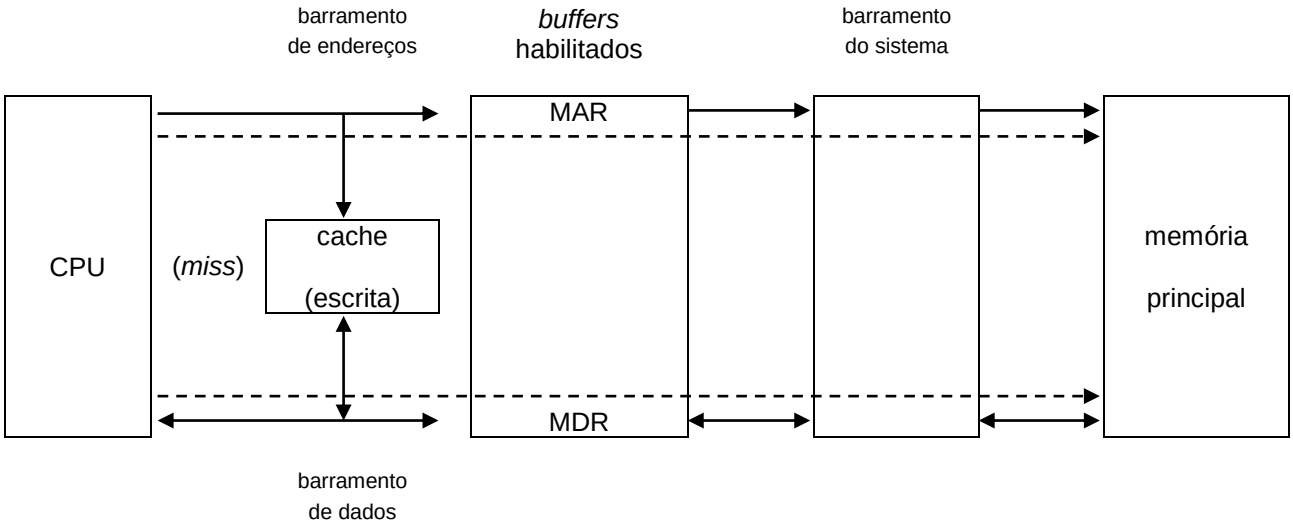
Caso contrário, ao ser necessário buscar um conteúdo não disponível (*miss*), ela também deverá ser atualizada.



Ao ser realizada uma operação de escrita, tanto o conteúdo da cache (*hit*) quanto da memória principal devem ser atualizados para manter coerência.



Caso o conteúdo não esteja disponível na cache (*miss*), a atualização será executada apenas na memória principal.



Paralelismo

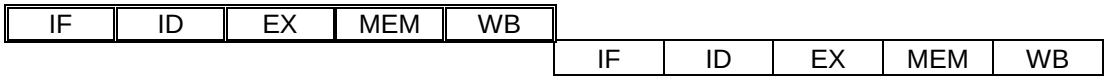
Tipos de paralelismo

Há quatro tipos básicos de paralelismo:

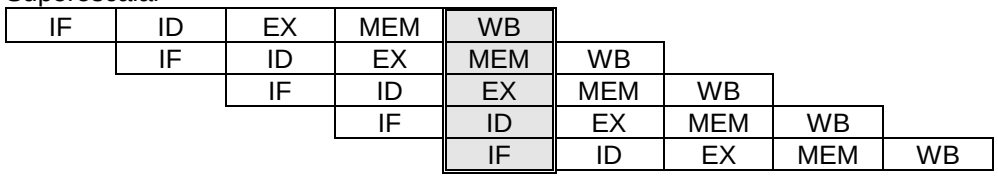
- ao nível de bits: o qual se obtém pelo aumento da palavra do processador, permitindo-se executar operações com dados maiores que o comprimento da mesma;

- ao nível de instrução: o qual se obtém otimizando os passos básicos para o tratamento de uma instrução: busca (IF – ***instruction fetch***), decodificação (ID – ***instruction decode***), execução (EX – ***execute***), acesso à memória (MEM) e reescrita (WB – ***writeback***).

Subescalar



Superescalar



- de dados: o qual se obtém distribuindo-se dados entre diversos nodos computacionais para que sejam processados em paralelo;

- de tarefas: o qual se obtém executando-se processamentos diferentes sobre um mesmo conjunto de dados ou sobre diferentes conjuntos de dados.

Tipos de computadores paralelos

Os tipos mais comuns de computadores paralelos são: com vários núcleos (***multicore***); simétricos (vários processadores com uma mesma memória compartilhada); distribuídos (conectados por rede: ***cluster***, massivamente paralelos; ***grid***); e os especializados (reconfiguráveis, de uso geral, específicos para aplicações e vetoriais).

Taxonomia segundo Flynn

SISD	SIMD
Single Instruction Single Data	***Single Instruction Multiple Data***
MISD	MIMD
Multiple Instruction Single Data	***Multiple Instruction Multiple Data***

- ***Single Instruction – Single Data***

Single instruction: apenas uma sequência de instruções pode ser executada por uma unidade de controle durante um ciclo de ***clock***
Single data: apenas uma sequência de dados pode ser usada como entrada durante um ciclo de ***clock***

Tipo de problema: uso geral
Tipo de execução: determinística
Tipo de computador: mainframes, minicomputadores e a maioria dos computadores atuais

- ***Single Instruction – Multiple Data***

Single instruction: apenas uma sequência de instruções pode ser executada emr todas as unidades de controle durante o mesmo ciclo de ***clock***
Multiple data: apenas uma sequência de dados pode ser usada como entrada durante o mesmo ciclo de ***clock***

Tipo de problema: com grande regularidade tais como os de processamento de imagens
Tipo de execução: síncrona (***lockstep***) e determinística
Tipo de computador: arranjos de processadores, ***pipelines*** vetoriais, processadores com GPU's

- ***Multiple Instruction – Single Data***

Multiple instruction: sequências independentes de instruções podem ser executadas em todas as unidades de controle durante o mesmo ciclo de ***clock***
Single data: apenas uma sequência de dados pode ser usada como entrada durante um ciclo de ***clock***

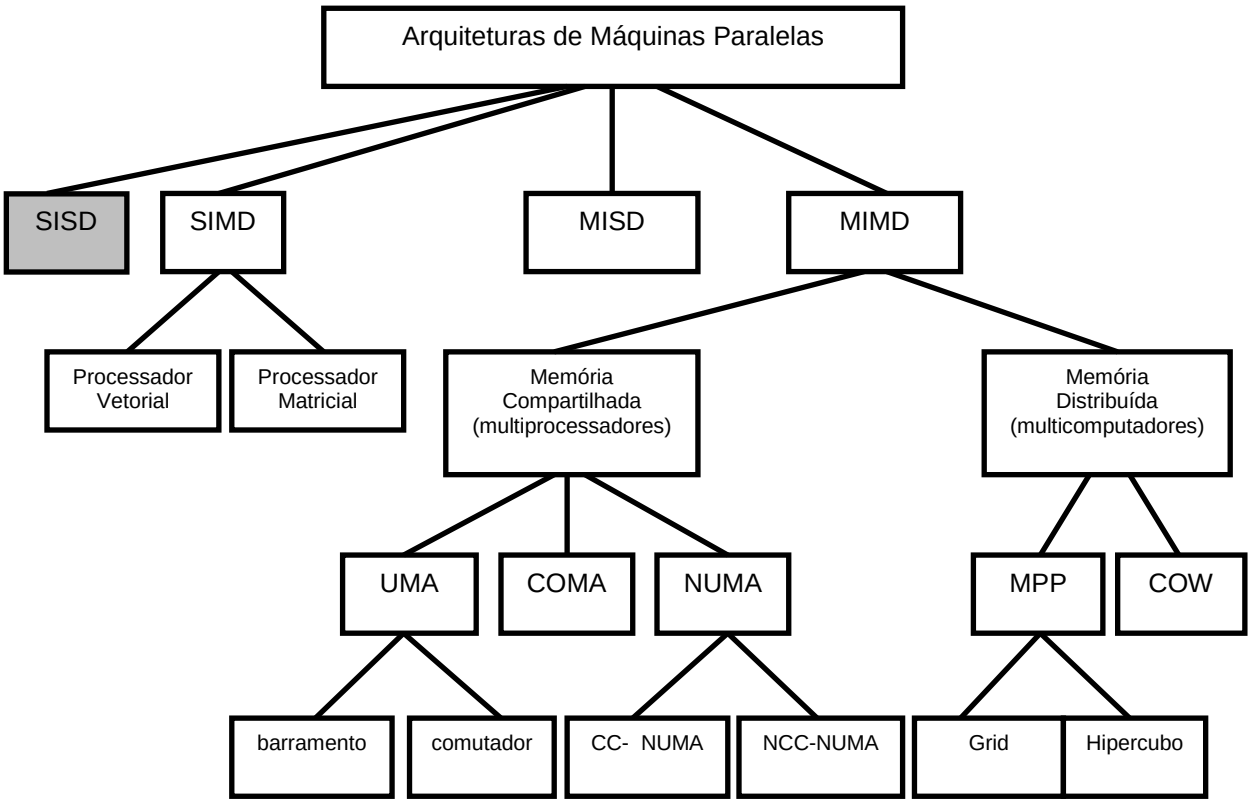
Tipo de problema: filtros de frequência e decifrar mensagem por vários algoritmos
Tipo de execução: assíncrona
Tipo de computador: Carnegie-Mellon C.mmp Computer (1971)

- ***Multiple Instruction – Multiple Data***

Multiple instruction: sequências independentes de instruções podem ser executadas em todas as unidades de controle durante o mesmo ciclo de ***clock***
Single data: sequências independentes de dados podem ser usadas como entrada durante um ciclo de ***clock***

Tipo de problema: filtros de frequência e decifrar mensagem por vários algoritmos
Tipo de execução: síncrona e assíncrona, determinística ou não-determinística
Tipo de computador: supercomputadores, ***clusters, grids***, computadores com múltiplos núcleos

Taxonomia adaptada segundo Tanenbaum



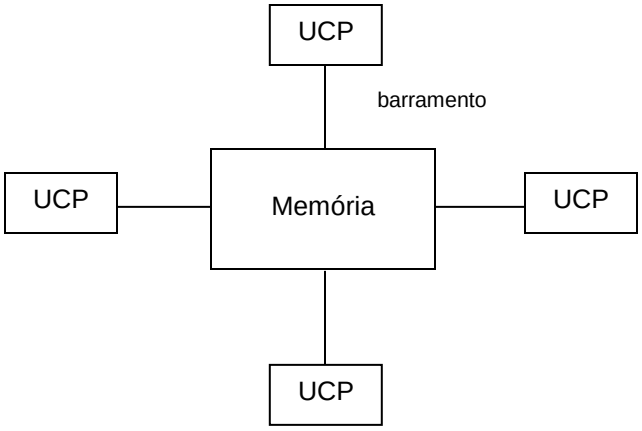
Arquiteturas de memória para computadores paralelos

- Memória compartilhada (***shared memory***)

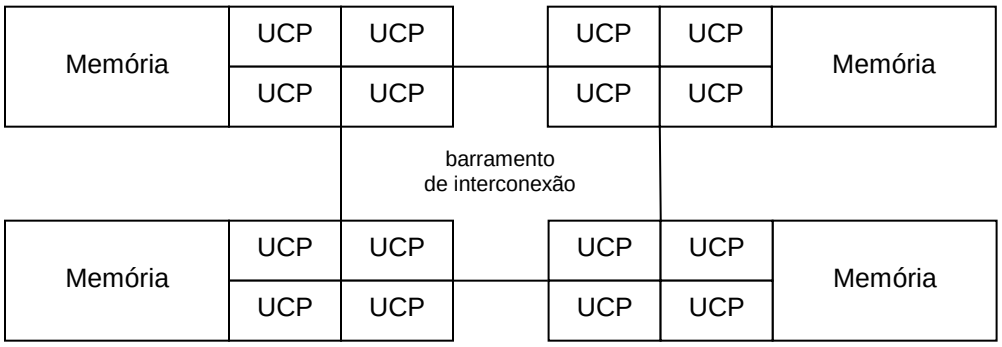
Vantagens: espaço de endereçamento global facilitado para o usuário
compartilhamento rápido e uniforme pelos processadores
coerência de cache mantida por ***hardware***

Desvantagens: falta de escalibilidade
(aumentar processadores implica maior tráfego no barramento)
maior responsabilidade para o programador para garantir acesso "correto"
(controle de sincronização)
maior custo
(para projetar e produzir memórias com múltiplos acessos)

- Modelo UMA (***Uniform Memory Access***)
  - Tipo de memória: a mesma com tempo de acesso igual para todos os processadores
  - Tipo de computador: multiprocessadores simétricos (SMP)



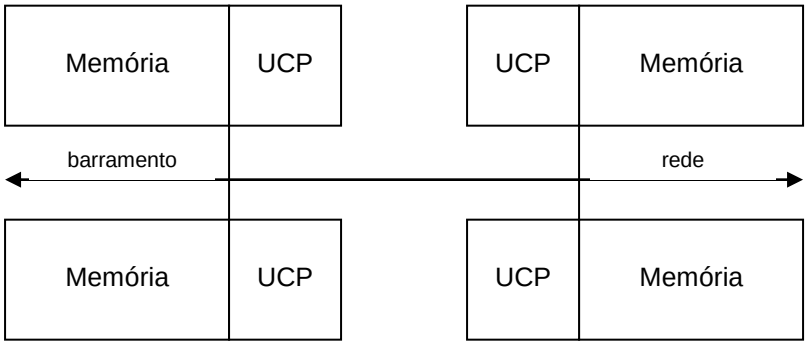
- Modelo NUMA (***Non-Uniform Memory Access***)
  - Tipo de memória: a mesma com tempo de acesso diferente entre processadores
  - Tipo de computador: conexões entre multiprocessadores simétricos (SMP)



- Memória distribuída (***distributed memory***)

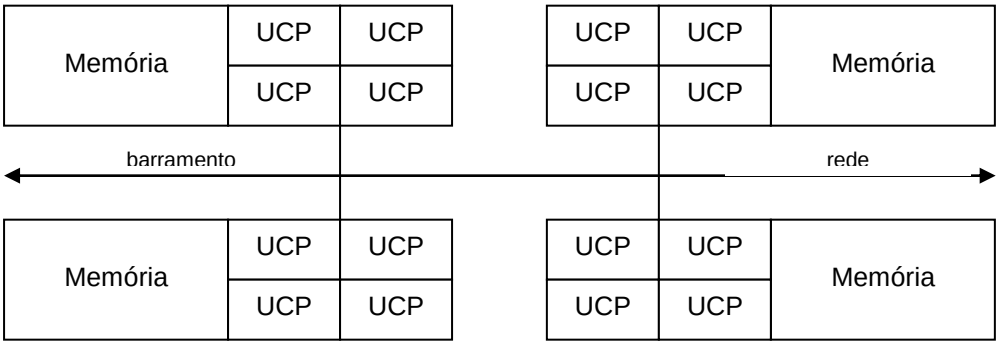
Vantagens: escalabilidade
acesso rápido à memória local pelo processador
não há trabalho extra (***overhead***) para manter coerência
custo compatível com processadores comuns e recursos de rede

Desvantagens: maior responsabilidade para o programador
para garantir comunicação de dados entre processadores
dificuldade em manter estruturas de dados
baseadas em memória global
tempo de acesso não uniforme



- Memória híbrida (***hybrid distributed-shared memory***)

Modelo comum à maioria dos processadores simétricos (SMP) atuais.
Os requisitos de acesso à memória dependem da infraestrutura de rede.



Modelos de programação paralela

- Memória compartilhada

Tarefas compartilham um espaço de endereçamento comum no qual podem ler e escrever assincronamente. Mecanismos como travas (***locks***) e semáforos (***semaphores***) podem ser usados para o controle de acesso à memória compartilhada.

Vantagens: simplicidade no desenvolvimento de programas
não há necessidade de um controle explícito da comunicação entre tarefas

Desvantagens: dificuldade em gerenciar a localidade dos dados
(manter dados locais conserva os acessos à memória, evita refrescamento de cache e tráfego no barramento, mas isso pode ser dificultado pelo uso de vários processadores ao mesmo tempo)

- ***Threads***

Um processo simples pode ter múltiplas sequências de execução concorrentes, cada um com acesso a dados locais, mas compartilhando recursos de um mesmo programa, como o espaço de memória global, através do qual se comunicam usando mecanismos de sincronização.

- Troca de mensagens

Um conjunto de tarefas que trabalham sobre suas próprias memórias locais podem se comunicar através de troca de mensagens, e assim agir de forma cooperativa.

- Dados paralelos

Um conjunto de tarefas trabalha sobre o mesmo conjunto de dados, mas cada uma delas lida com uma porção específica da estrutura de dados.

- Híbridos

Combinações de dois ou mais modelos de programação paralela, como por exemplo, troca de mensagens com ***threads*** ou memória compartilhada. Outro modelo pode combinar dados paralelos com troca de mensagens.

- SPMD (***Single Program Multiple Data***)

Um mesmo programa é executado por todas as tarefas simultaneamente, mas não exatamente as mesmas instruções ao mesmo tempo, uma vez que cada tarefa lidará com uma porção diferente do conjunto de dados.

- MPMD (***Multiple Program Multiple Data***)

Vários programas podem estar ativos, trabalhando em paralelo.

Prova de teoremas

As relações descritas podem ser utilizadas em Lógica Formal para provar teoremas.

Exemplo 1:
Provar (s') dados:

1. t - premissa
2. $t \rightarrow q'$ - premissa
3. $q' \rightarrow s'$ - premissa

4. q' - *modus ponens* entre 1 e 2
5. s' - *modus ponens* entre 3 e 4 (c.q.d.)

Exemplo 2:
Provar (a) dados :

1. $a' \rightarrow b$ - premissa
2. $b \rightarrow c$ - premissa
3. c' - premissa

4. b' - *modus tolens* entre 2 e 3
5. (a')' - *modus tolens* entre 1 e 4
6. a - dupla negação (c.q.d.)

Quantificadores

A Lógica de Predicados, ou Lógica de Primeira Ordem, associa o uso de quantificadores.

Quantificador Universal

$(\forall x \in S) \;\; P(x)$ - ***para todo*** x em S, P(x) é verdadeiro

Quantificador Existencial

$(\exists x \in S) \;\; P(x)$ - ***para algum*** x em S, P(x) é verdadeiro

Os quantificadores admitem complementação :

não $\left((\forall x) \;\; P(x) \right)$ = $(\exists x)$ (não P(x))

não $\left((\exists x) \;\; P(x) \right)$ = $(\forall x)$ (não P(x))

Exemplo :
Supor o predicado abaixo, devidamente quantificado :

$$(\forall x \in R) \;\; (x^2 + 2x - 1 > 0)$$

não $\left((\forall x \in R) \;\; (x^2 + 2x - 1 > 0) \right) \rightarrow (\exists x \in R) \;\; (x^2 + 2x - 1 \leq 0)$

o que pode ser verificado quando x = 0 !

Exercícios propostos

1. Provar, pela álgebra, que :

   a) $(p \cdot q') + p' = p' + q'$

   b) $p \cdot (p' + q) = p \cdot q$

   c) $p \cdot q + (p' + q')' = (p' + q')'$

   d) $(p \cdot q + r) \cdot ((p \cdot q)' \cdot r') = 0$

   e) $(p \cdot q + r') \cdot (p \cdot q \cdot r' + p \cdot r') = p \cdot r'$

2. Fazer as tabelas e os circuitos do exercício anterior.

3. Demonstrar as relações abaixo através de tabelas:

   a) $x + x \cdot y = x$ (absorção)

   b) $x \cdot (x + y) = x$ (absorção)

   c) $(x + y) \cdot (x + z) = x + y \cdot z$

   d) $x + x' \cdot y = x + y$

   e) $x \cdot y + y \cdot z + y' \cdot z = x \cdot y + z$

4. Verificar o resultado das seguintes relações :

   a) $(x \rightarrow y) \cdot x \rightarrow y$ ("modus ponens")

   b) $(x \rightarrow y) \cdot y' \rightarrow x'$ ("modus tolens")

   c) $(x + y) \cdot x' \rightarrow y$ (silogismo disjuntivo)

   d) $(x \rightarrow y) \cdot (y \rightarrow z) \rightarrow (x \rightarrow z)$ (silogismo hipotético)

   e) $((x \rightarrow y) \cdot (w \rightarrow z)) \cdot (x+w) \rightarrow (y+z)$ (dilema)

   f) $(x \rightarrow y) \rightarrow (x \rightarrow x \cdot y)$ (absorção)

   g) $x \cdot y \rightarrow x$ (simplificação)

   h) $x \rightarrow x + y$ (adição)

   i) $(x' \rightarrow x) \rightarrow x$ (contradição)

   j) $(x' \rightarrow y) \rightarrow ((x' \rightarrow y') \rightarrow x)$ (contradição)

   h) $y \cdot (y \cdot x' \rightarrow z) \cdot (y \cdot x' \rightarrow z') \rightarrow x$ (contradição)

   i) $(x \rightarrow y) = (x \cdot y') \rightarrow (y \cdot y')$ (redução ao absurdo)

5. Verificar o resultado das seguintes relações

a) $x \cdot (y \cdot z) \Leftrightarrow (x \cdot y) \cdot z$ (associatividade)

b) $x + (y + z) \Leftrightarrow (x + y) + z$ (associatividade)

c) $x \cdot y \Leftrightarrow y \cdot x$ (comutatividade)

d) $x + y \Leftrightarrow y + x$ (comutatividade)

e) $x \cdot (y+z) \Leftrightarrow (x \cdot y) + (x \cdot z)$ (distributividade)

f) $x+(y \cdot z) \Leftrightarrow (x+y) \cdot (x+z)$ (distributividade)

g) $(x \cdot y)' \Leftrightarrow x' + y'$ (De Morgan)

h) $(x+y)' \Leftrightarrow x' \cdot y'$ (De Morgan)

i) $(x')' \Leftrightarrow x$ (involução)

j) $x \cdot x \Leftrightarrow x$ (idempotência)

k) $x + x \Leftrightarrow x$ (idempotência)

l) $x \rightarrow y \Leftrightarrow x' + y$ (implicação)

m) $(x \Leftrightarrow y) \Leftrightarrow (x \rightarrow y) \cdot (y \rightarrow x)$ (equivalência)

n) $x \rightarrow y \Leftrightarrow x' \rightarrow y'$ (contraposição)

o) $(x \cdot y) \rightarrow z \Leftrightarrow x \rightarrow (y \rightarrow z)$ (exportação)

6. Transformar as afirmativas abaixo em proposições :

a) A soma de dois inteiros é maior que o primeiro.
b) A soma de dois inteiros é maior que a diferença entre eles.
c) O produto de um inteiro por zero é menor que outro inteiro.
d) Se um de dois números são nulos, o produto deles é zero.
e) Se dois números são iguais a um terceiro, então são iguais.

7. Construir um circuito para identificar se dois bits são iguais.

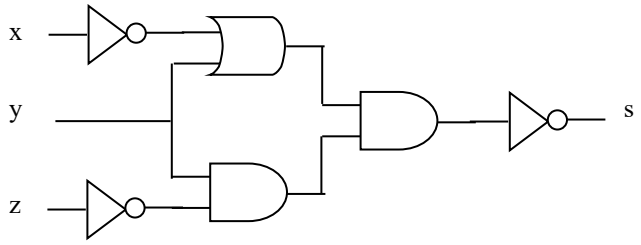
8. Construir um circuito capaz de fornecer a tabela abaixo :

x y z	f (x,y,z)
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

9. Simplificar a expressão lógica abaixo :

(x > 0 e y = 0) ou não (x < 0 ou z > 0)

10. Identificar a equação do circuito abaixo pela soma de produtos (SoP):



11. Identificar a equação do circuito anterior pelo produto das somas (PoS).

12. Construir um circuito equivalente ao da questão (10) usando apenas portas NAND.

13. Construir um circuito equivalente ao da questão (10) usando apenas portas NOR.

14. Montar um diagrama de um somador completo de 2 palavras de 2 bits cada.

15. Construir um circuito equivalente ao da questão (14) usando apenas portas NAND.

16. Construir um circuito equivalente ao da questão (14) usando apenas portas NOR.

17. Simplificar por mapa de Karnaugh: a'b'c'd'+a'b'c'd+a b'c'd+a b c'd'+a b c d'+a b c'd+a b c d

18. Demonstrar a validade do seguinte argumento:

(1) x < 6
(2) y > 7 ou x = y $\rightarrow$ (y = 4 e x < y)
(3) y = 4 $\rightarrow$ x < 6
(4) x < 6 $\rightarrow$ x < y
$\therefore$ x = y

19. Verificar se as proposições são falsas ou verdadeiras:

a) $(\forall n \in N)$ (n+4 > 3)

b) $(\forall n \in N)$ (n+3 > 7)

c) $(\exists n \in N)$ (n+4 < 7)

d) $(\exists n \in N)$ (n+3 < 2)

20. Sendo A = {1,2,3,4,5} determinar a negação de:

a) $(\exists n \in A)$ (x+4 = 10)

b) $(\forall n \in N)$ (x+4 < 10)