

# Unidade II

## 3 NÍVEL DE LÓGICA DIGITAL

O desenvolvimento da lógica digital, que envolve o uso de sinais discretos (0 e 1) para representar informações, ganhou força no final do século XIX, com o trabalho de George Boole na álgebra booleana. Essa álgebra, que utiliza operadores lógicos para manipular valores verdade (verdadeiro ou falso), estabeleceu as bases para a implementação de circuitos digitais. A invenção do eletromecânico, no início do século XX, permitiu a construção de máquinas de calcular mais complexas.

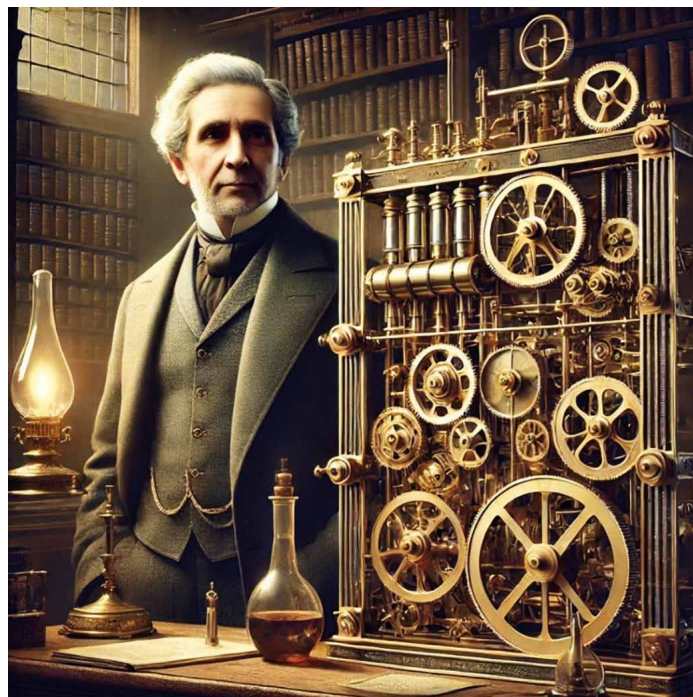


Figura 16 – Charles Babbage e sua criação, a Máquina Analítica, precursora dos computadores modernos, imagem gerada em IA Image Generator

A década de 1930 marcou uma transição significativa para a lógica digital com o desenvolvimento da primeira máquina de calcular eletrônica. O uso de válvulas a vácuo permitiu a construção de circuitos digitais mais rápidos e compactos. A Segunda Guerra Mundial impulsionou o desenvolvimento da lógica digital com a necessidade de computadores para criptografia e cálculo de trajetórias de mísseis. Os computadores da era da guerra, como o Colossus e o ENIAC, usavam milhares de válvulas a vácuo, eram grandes, geravam muito calor e consumiam muita energia, mas a descoberta do transistor, em 1947, revolucionou a eletrônica. Os transistores, que substituíram as volumosas válvulas a vácuo, eram menores, mais eficientes e confiáveis. Assim, a miniaturização dos componentes digitais possibilitou a construção de computadores menores, mais rápidos e acessíveis.



### Lembrete

O impacto da álgebra booleana no mundo tecnológico de hoje é inegável. Sem ela, não teríamos os computadores e dispositivos digitais que usamos todos os dias. Ela é uma ferramenta essencial para o desenvolvimento de novas tecnologias e para a resolução de problemas complexos em áreas como ciência da computação, engenharia e inteligência artificial.

### Tecnologia de transistores e integração em chips

A década de 1960 viu a ascensão dos circuitos integrados (chips), que permitiam a integração de múltiplos transistores em um único chip de silício. A integração em larga escala (LSI) permitiu uma miniaturização ainda maior de componentes digitais, com milhares de transistores em um único chip. A LSI possibilitou a criação de computadores pessoais (PCs) menores e mais acessíveis. A década de 1970 trouxe a integração em muito alta escala (VLSI), com milhões de transistores em um único chip, o que possibilitou o desenvolvimento de microprocessadores cada vez mais poderosos, abrindo caminho para a explosão da indústria de computadores pessoais e a proliferação de dispositivos digitais. A Lei de Moore, observada pela primeira vez no início da década de 1965, previa que o número de transistores em um chip dobraria a cada dois anos. Essa lei se manteve válida por décadas, impulsionando a miniaturização e o aumento da capacidade dos chips.

### Microprocessadores e arquiteturas computacionais digitais

O desenvolvimento de microprocessadores, como o Intel 4004 em 1971, inaugurou uma nova era na computação. O microprocessador, que integrava todas as funções de uma CPU em um único chip, tornou possível a criação de computadores menores, mais poderosos e acessíveis. O avanço na tecnologia de microprocessadores impulsionou a criação de novos sistemas operacionais, softwares e aplicações, expandindo as capacidades e funcionalidades dos computadores. A arquitetura de von Neumann, desenvolvida na década de 1940, tornou-se o padrão para a maioria dos computadores modernos. Essa arquitetura utiliza uma única unidade de memória para armazenar tanto instruções quanto dados, permitindo que o computador acesse e execute instruções de forma eficiente. A arquitetura de Harvard, que utiliza unidades de memória separadas para instruções e dados, permite acesso simultâneo a ambos, resultando em maior velocidade de execução, mas é menos comum em computadores gerais.

### Lógica programável e circuitos reconfiguráveis

Os avanços na lógica digital também levaram ao desenvolvimento de dispositivos de lógica programável, como as Field-Programmable Gate Arrays (FPGAs), que permitem que os usuários configurem a funcionalidade dos circuitos de forma flexível, o que as torna ideais para aplicações que requerem personalização e adaptação. Os circuitos reconfiguráveis oferecem flexibilidade e eficiência, permitindo que os designers criem e modifiquem seus próprios circuitos de acordo com

suas necessidades específicas. A lógica programável também está presente em dispositivos como microcontroladores, que são utilizados em uma ampla variedade de aplicações, desde dispositivos domésticos até sistemas industriais. A capacidade de programar a lógica de um circuito digital abre novas possibilidades para a criação de sistemas personalizados, que podem ser adaptados a diferentes tarefas e cenários.

## **Computação quântica: uma nova fronteira da lógica digital**

A computação quântica representa uma revolução na lógica digital, explorando princípios da mecânica quântica para resolver problemas complexos que estão além das capacidades dos computadores clássicos. Em vez de bits (0 ou 1), os computadores quânticos usam qubits, que podem representar 0, 1 ou uma combinação de ambos. Essa superposição quântica permite que os computadores quânticos processem informações de forma exponencialmente mais rápida, abrindo novas possibilidades para a resolução de problemas em áreas da descoberta de medicamentos, da ciência de materiais e da inteligência artificial. A computação quântica ainda está em seus estágios iniciais, mas tem o potencial de transformar a lógica digital e impactar diversos campos da ciência e da tecnologia.

## **Inteligência artificial e aprendizado de máquina: impacto na lógica digital**

A inteligência artificial (IA) e o aprendizado de máquina (ML) estão transformando a lógica digital, permitindo que os computadores aprendam e se adaptem com base em dados. Algoritmos de IA e ML podem analisar grandes conjuntos de dados, identificar padrões e fazer previsões, o que faz com que suas aplicações estejam cada vez mais presentes em nossas vidas, desde assistentes virtuais até carros autônomos. Os avanços em IA e ML estão impulsionando o desenvolvimento de sistemas de lógica digital mais complexos e eficientes, assim como desempenham um papel crucial na otimização de processos, na tomada de decisões e na personalização de experiências.

A lógica digital continua a evoluir a um ritmo acelerado e, atualmente, as tendências incluem a miniaturização contínua dos chips, a integração de componentes digitais em dispositivos móveis e a crescente importância da computação em nuvem. A nanotecnologia e a computação neuromórfica são áreas promissoras que podem revolucionar a lógica digital no futuro. A nanotecnologia permite a construção de componentes digitais em escala atômica, possibilitando a criação de chips ainda menores e mais poderosos. A computação neuromórfica visa imitar a estrutura e o funcionamento do cérebro humano, abrindo novas possibilidades para a inteligência artificial e o processamento de informações. As tendências futuras também incluem a computação quântica, a Internet das Coisas (IoT) e a realidade virtual e aumentada, que dependerão de avanços contínuos na lógica digital.

## **3.1 Introdução a portas lógicas**

As portas lógicas são os elementos básicos da eletrônica digital, responsáveis por processar e manipular sinais binários, ou seja, informações representadas por 0 e 1. Essas portas realizam operações lógicas, como AND, OR, NOT, entre outros, que formam a base para a construção de circuitos digitais mais complexos, como computadores, dispositivos móveis e sistemas de controle industrial.

Em sua essência, uma porta lógica recebe um ou mais sinais de entrada binários e, de acordo com sua função lógica, produz um único sinal de saída binário. Essas funções lógicas são definidas por tabelas-verdade, que mostram a relação entre as entradas e a saída para todas as combinações possíveis. Cada porta lógica possui uma representação simbólica específica, facilitando a compreensão e o desenho de circuitos digitais.

### Variáveis booleanas e tabela-verdade

A principal diferença entre a álgebra convencional e a álgebra booleana é que, nesta última, as variáveis somente assumem dois valores possíveis ou, no caso, dois bits possíveis, 0 e 1. Essas variáveis booleanas também são utilizadas na representação de níveis de tensão elétrica presentes em determinada conexão de circuitos digitais muito utilizadas em computação, onde, por exemplo, pode-se assumir um valor booleano 0 para representar alguma tensão baixa na faixa de 0 a 0,8 V, enquanto pode assumir o valor 1 para representar alguma tensão elétrica dentro da faixa de 2 a 5 V. Dessa forma, os valores 0 e 1 não representam efetivamente números, mas sim um estado do nível de tensão elétrica que pode ser variável e que será chamado de nível lógico. O sistema lógico da álgebra booleana se baseia em apenas três operações básicas: OU (OR), E (AND) e NÃO (NOT). Essas operações são conhecidas como operações lógicas e os circuitos baseados nessas operações são denominados circuitos lógicos digitais. As portas lógicas individuais são constituídas por diversos dispositivos eletrônicos como transistores, diodos e resistores, os quais estão interconectados de modo que cada um tenha sua respectiva saída para o sistema de operações lógicas AND, OR ou NOT ou de suas variantes. Também faz parte da constituição do projeto de circuitos lógicos uma tabela-verdade, que nada mais é do que uma técnica para descrever a saída de um circuito lógico dependente de suas entradas. A figura 17 mostra uma tabela-verdade considerando um circuito lógico com duas entradas (A, B) e relaciona sua saída (X) com as possíveis combinações de acordo com o operador simbolizado pela incógnita (?).

Entradas		Saída
A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

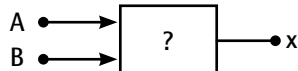
  


Figura 17 – Tabela-verdade com duas entradas e sua respectiva porta lógica

Adaptada de: Tocci, Widmer e Moss (2011, p. 50).

A primeira linha da tabela-verdade da figura 17 advém do resultado de quando A e B possuírem o nível lógico 0 e sua saída X será nível 1. Na segunda linha da tabela-verdade, quando a entrada A possuir nível lógico 0 e a entrada B possuir nível lógico 1, a saída X será 0. Na terceira linha, enquanto a entrada A é 1 e a outra entrada B é 0, a saída X será 1. Finalmente, na quarta linha, quando a entrada A é 1 e

a entrada B também é 1, sua saída X será 0. A quantidade de linhas e colunas em uma tabela-verdade é determinada pela quantidade de entradas da expressão lógica. Por exemplo, a figura 17 possui duas entradas (A e B), logo, possuirá duas colunas, enquanto que, para determinar a quantidade de linhas, basta elevar a quantidade de entradas pela base 2, ou seja,  $2^2$ , resultando em quatro linhas. Note que esse exemplo é fictício e não foram levadas em consideração as operações lógicas AND, OR ou NOT para que o resultado tenha algum sentido real. A figura 18 mostra outro exemplo de uma tabela-verdade, agora baseada em quatro entradas (A, B, C e D) e uma saída X, seguindo a mesma lógica da figura anterior. Note que o número de colunas corresponde ao número de entradas, no caso 4, e, para saber a quantidade de linhas dessa tabela, é só realizar a operação  $2^4$ , resultando em 16 linhas.

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figura 18 – Tabela-verdade com duas quatro entradas

Adaptada de: Tocci, Widmer e Moss (2011, p. 50).

## 3.1.1 Operação com porta lógica OU (OR)

A operação lógica OR é definida a partir da expressão booleana  $x=A+B$  onde, nessa expressão, o sinal de + não representa uma adição convencional algébrica, mas sim a própria operação OR. Tal operação é descrita pela tabela-verdade de duas entradas (A, B) e uma saída (x) na figura 19.

OR		
A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Figura 19 – Tabela-verdade da operação lógica OR

Fonte: Tocci, Widmer e Moss (2011, p. 51).

Note que a expressão resultante de cada linha das operações OR  $A+B$  resultará em um valor  $x$ , de acordo com a combinação das entradas. Em álgebra booleana, a soma de  $1+1$  é um 1, e não 2, já que, na lógica booleana, não é possível representar um número maior do que 1. A representação para o bit 1 é o nível (estado) ALTO, enquanto que o bit 0 representa o nível (estado) BAIXO. Assim, de acordo com a tabela-verdade, lê-se:

$$0 + 0 = 0 \quad (0 \text{ OR } 0)$$

$$0 + 1 = 1 \quad (0 \text{ OR } 1)$$

$$1 + 0 = 1 \quad (1 \text{ OR } 0)$$

$$1 + 1 = 1 \quad (1 \text{ OR } 1)$$

Em termos de lógica booleana, é possível dizer que na operação OR, quando a entrada A estiver em nível BAIXO e B também estiver em nível BAIXO, a resultante  $x$  estará em nível BAIXO. Quando a entrada A estiver em nível BAIXO e a entrada B estiver em nível ALTO, a saída  $x$  resultante será igual ao nível ALTO. Circuitos digitais baseados em portas lógicas OR possuem, geralmente, duas ou mais entradas, cuja saída será igual à combinação das entradas através da operação OR, como mostra a figura 18:

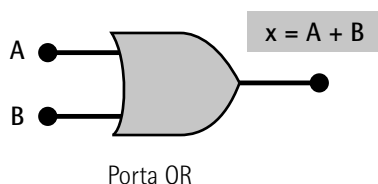


Figura 20 – Porta lógica OR

Fonte: Tocci, Widmer e Moss (2011, p. 51).

Em uma porta lógica com três entradas, como na figura 21, a saída sempre será 1 quando ao menos uma das entradas, ou ambas, forem 1, e a saída será 0 somente quando todas as entradas forem 0.

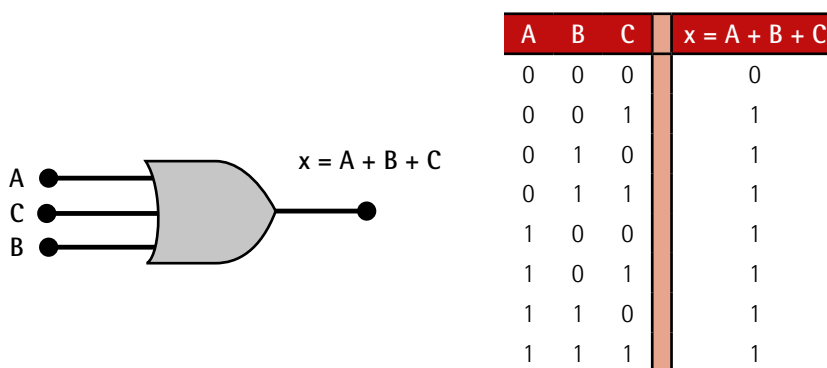


Figura 21 – Porta lógica OR de três entradas e sua respectiva tabela-verdade

Fonte: Tocci, Widmer e Moss (2011, p. 52).

A partir da explanação da porta lógica OR, é possível resolver o exemplo a seguir.

## Exemplo de aplicação

A partir do diagrama de tempo da figura 22, onde a entrada A tem seu início em nível BAIXO no instante  $t_0$  e muda para o estado ALTO no instante  $t_1$ , voltando para o estado BAIXO somente no instante  $t_3$ , alternadamente e sucessivamente. A entrada B também se inicia em estado BAIXO no instante de tempo  $t_0$  e assim permanece até o instante  $t_2$ , alternando seu formato sucessivamente. A partir dessas entradas, determine qual será sua saída, baseando-se no uso de uma porta lógica OR. Note que na figura foram adicionados os bits 0 e 1 para representar as variações entre os níveis ALTO e BAIXO.

## Resolução

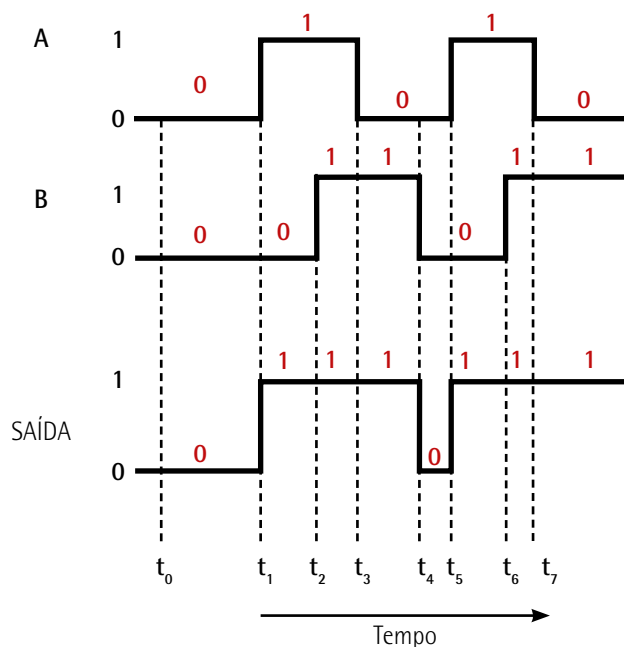


Figura 22 – Ondas digitais A e B e sua respectiva combinação de saída em função do tempo considerando o uso de uma porta OR

Adaptada de: Tocci, Widmer e Moss (2011, p. 53).

Note que, na figura, a saída da porta OR sempre terá nível ALTO para qualquer entrada também em nível ALTO. Por exemplo, nos instantes de tempo entre  $t_1$  e  $t_2$ ;  $t_2$  e  $t_3$ ;  $t_3$  e  $t_4$ ;  $t_5$  e  $t_6$ ;  $t_6$  e  $t_7$  as saídas serão nível ALTO, ou bit 1. Somente entre os tempos  $t_4$  e  $t_5$  a saída terá nível BAIXO ou bit 0.

## 3.1.2 Operação com porta lógica E (AND)

A operação lógica AND é definida a partir da expressão booleana  $x=A.B$ , onde o sinal (.) não representa uma multiplicação convencional algébrica, mas sim a operação AND.

A operação AND é descrita pela tabela-verdade de duas entradas (A, B) e uma saída (x) na figura 23.

AND		
A	B	$x = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Figura 23 – Tabela-verdade da operação lógica AND

Fonte: Tocci, Widmer e Moss (2011, p. 54).

Note que a expressão resultante de cada linha das operações AND  $A.B$  resultará em um certo valor  $x$  na saída, de acordo com a combinação das entradas. A representação para a operação AND pode ser lida como:

$$0 \cdot 0 = 0 \quad (0 \text{ AND } 0)$$

$$0 \cdot 1 = 0 \quad (0 \text{ AND } 1)$$

$$1 \cdot 0 = 0 \quad (1 \text{ AND } 0)$$

$$1 \cdot 1 = 1 \quad (1 \text{ AND } 1)$$

Ao contrário da operação OR, na operação AND, somente quando as entradas estiverem todas em nível lógico ALTO (bit 1) a saída também estará em nível ALTO. Caso contrário, qualquer combinação que possua alguma das entradas em nível BAIXO (bit 0) também estará com a saída em nível BAIXO. Circuitos digitais baseados em portas lógicas AND possuem entre duas ou mais entradas, cuja saída será igual à combinação das entradas através da operação AND, como mostra a figura 24.

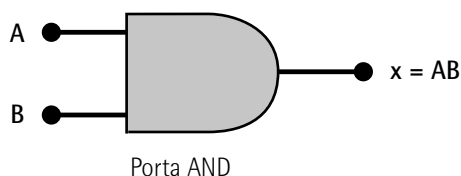


Figura 24 – Porta lógica AND

Fonte: Tocci, Widmer e Moss (2011, p. 54).



Em suma, a saída sempre será 1 quando todas as entradas também forem iguais ao bit 1 ou, caso contrário, será 0, como mostra a tabela-verdade da figura 25 para uma porta lógica AND com três entradas.

A	B	C	$x = ABC$
0	0	0	0
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

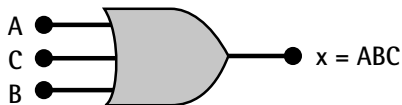


Figura 25 – Tabela-verdade referente a uma porta lógica AND de três entradas

Fonte: Tocci, Widmer e Moss (2011, p. 55).

## Exemplo de aplicação

A partir da explicação sobre a porta lógica AND, é possível determinar uma saída  $x$  referente às duas entradas  $A$  e  $B$ , como mostra o diagrama de tempo da figura 26. Inicialmente, a entrada  $A$  está em nível BAIXO no instante  $t_0$  e muda para o estado ALTO no instante  $t_1$ , voltando para o estado BAIXO no instante  $t_3$  e, assim, sucessivamente. Já a entrada  $B$  está em nível BAIXO do instante  $t_0$ , permanecendo até o instante de tempo  $t_2$  quando muda para o estado ALTO, variando até o final no instante  $t_7$ . A partir do diagrama de ondas das entradas  $A$  e  $B$ , determine qual será sua saída, baseando-se no uso de uma porta lógica AND. Note que na figura foram adicionados os bits 0 e 1 para representar as variações entre os níveis ALTO e BAIXO.

### Resolução

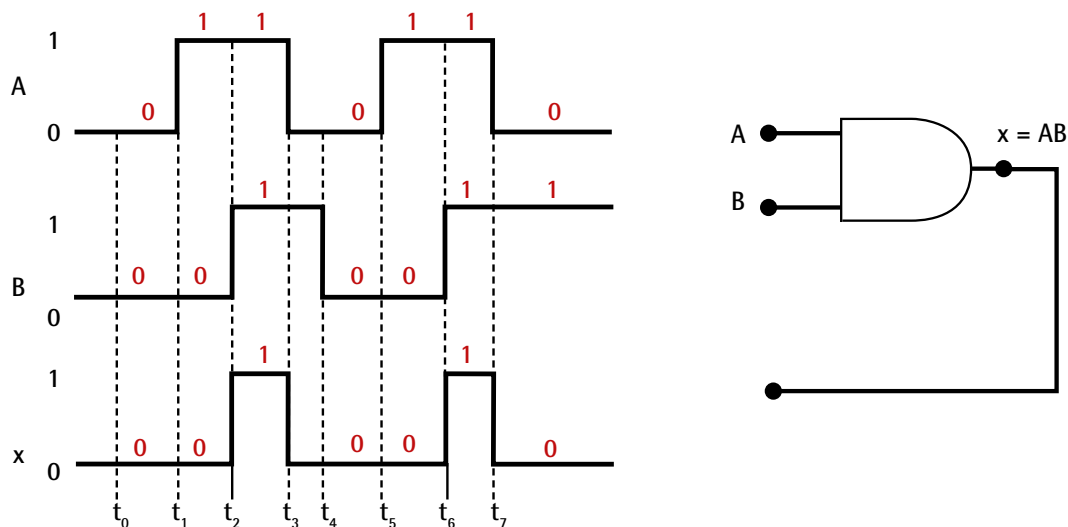


Figura 26 – Ondas digitais A e B e suas respectivas combinações de saída em função do tempo considerando o uso de uma porta AND

Adaptada de: Tocci, Widmer e Moss (2011, p. 56).

Na figura 26, a saída da porta AND de duas entradas somente estará em nível ALTO quando ambas as entradas estiverem também em nível ALTO e a saída será nível BAIXO para qualquer outra combinação. Por exemplo, somente nos instantes de tempo entre  $t_2$  e  $t_3$ ,  $t_6$  e  $t_7$  as saídas serão nível ALTO, ou bit 1, e todos os demais intervalos de tempo estarão em nível BAIXO ou bit 0.

### 3.1.3 Operação com porta lógica NÃO (NOT) ou inversora

A operação lógica do tipo NOT, também denominada de operação de "complemento", "inversão" ou "negação", é totalmente diferente das operações OR ou AND devido ao fato de que ela pode ser realizada sobre apenas uma variável de entrada. A simbologia para a inversão de uma entrada é dada por:  $x = \bar{A}$ , onde ( $\bar{A}$  barrado ou complemento de A) representa a operação inversa de A. Atribuindo bits pode-se representar da seguinte forma:

$A=1$  então  $=0$

$A=0$  então  $=1$

Um circuito inversor possui um formato muito simples de ser representado, visto que é composto de apenas uma entrada e possui em sua representação um pequeno círculo em sua saída para denotar a operação de inversão, como mostra a figura 27.

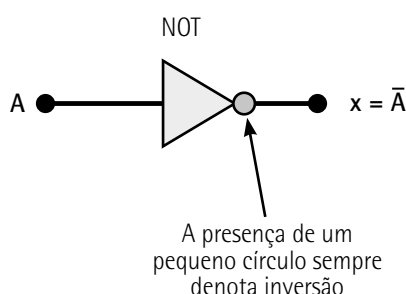


Figura 27 – Porta lógica NOT

Fonte: Tocci, Widmer e Moss (2011, p. 57).

A operação NOT é descrita pela tabela-verdade de uma entrada (A) e uma saída (x), como podemos ver na figura 28.

NOT	
A	$x = \bar{A}$
0	1
1	0

Figura 28 – Tabela-verdade da operação lógica NOT

Fonte: Tocci, Widmer e Moss (2011, p. 57).

A tabela-verdade da figura 29 só possuirá duas linhas, pois a tabela só contém uma entrada (A), logo,  $2^1=2$  e suas respectivas saídas são exatamente o inverso dos valores do bit na entrada. A representação em formato de onda para uma porta NOT é mostrada na figura 29 e representa o sinal de saída (x) com estado (nível) exatamente oposto ao de entrada (A).

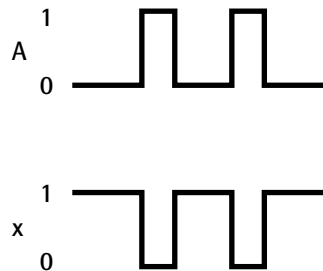


Figura 29 – Ondas digitais de entrada (A) e saída (x) em função do tempo considerando o uso de uma porta NOT

Adaptada de: Tocci, Widmer e Moss (2011, p. 57).

### 3.1.4 Operação com porta lógica NÃO OU (NOR)

A operação lógica NOR é equivalente às operações OR e NOT combinadas, sendo representada pela porta OR seguida de uma inversora, como mostra a figura 30.

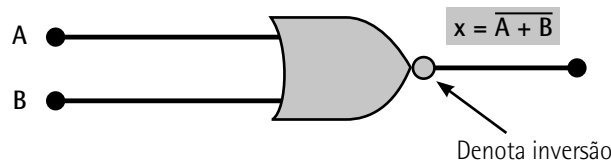


Figura 30 – Porta lógica NOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 64).

Note que a expressão de saída para uma combinação de duas entradas (A e B) é dada por  $x = \overline{A + B}$ . A tabela-verdade da porta NOR terá a saída exatamente contrária a tabela-verdade da porta OR devido ao uso de uma inversora, como mostra a figura 31.

		OR		NOR	
A	B	$A + B$		$\overline{A + B}$	
0	0	0		1	
0	1	1		0	
1	0	1		0	
1	1	1		0	

Figura 31 – Tabela-verdade da operação lógica NOR

Adaptado de: Tocci, Widmer e Moss (2011, p. 64).

A saída  $x = \overline{A + B}$  somente estará em nível lógico ALTO se todas as entradas (A, B etc.) estiverem em nível lógico BAIXO e em qualquer outra combinação de entradas a saída sempre estará em nível lógico BAIXO. A figura 32 mostra como é uma combinação com duas entradas (A, B) e sua respectiva saída ao se utilizar uma porta lógica NOR na representação de ondas digitais. Note que, conforme há uma variação em função do tempo nas entradas, a saída também se modifica.

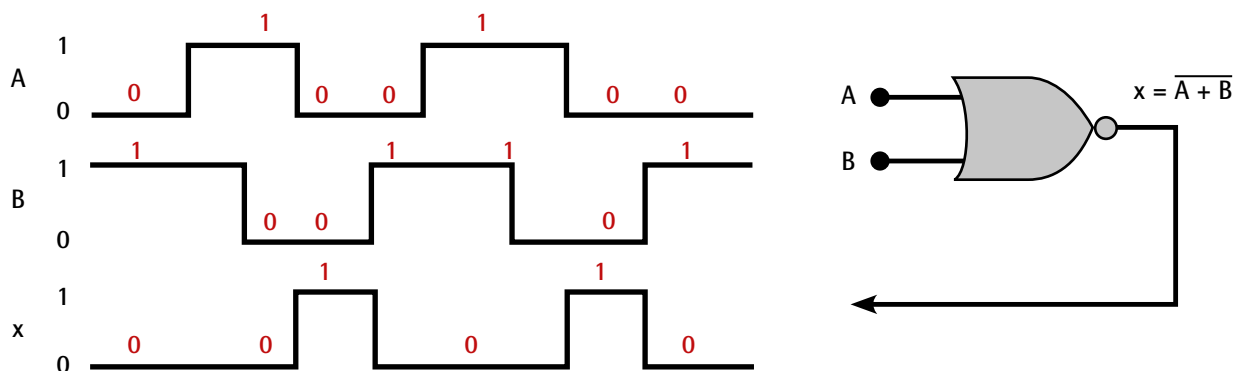


Figura 32 – Ondas digitais A e B e suas respectivas combinações de saída em função do tempo considerando o uso de uma porta NOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 65).

Na figura 32, a saída da porta NOR de duas entradas (A e B) somente estará em nível ALTO quando ambas as entradas estiverem em nível BAIXO e a saída será nível BAIXO para qualquer outra combinação.

### 3.1.5 Operação com porta lógica NÃO E (NAND)

A operação lógica NAND é equivalente as operações AND e NOT combinadas e é representada pela porta AND seguida de uma inversora, como mostra a figura 33.

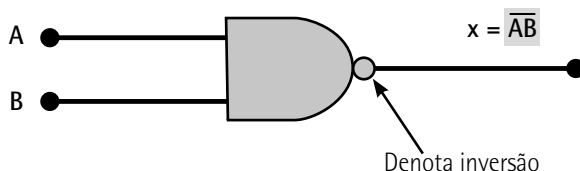


Figura 33 – Porta lógica NAND

Adaptado de: Tocci, Widmer e Moss (2011, p. 66).

Note que a expressão de saída para uma combinação de duas entradas (A e B) é dada por  $x = \overline{AB}$ . A tabela-verdade da porta NAND possuirá a saída exatamente contrária à tabela-verdade da porta AND devido ao uso de uma inversora, como mostra a figura 34.

		AND		NAND	
A	B		AB		$\overline{AB}$
0	0		0		1
0	1		0		1
1	0		0		1
1	1		1		0

Figura 34 – Tabela-verdade da operação lógica NAND

Adaptada de: Tocci, Widmer e Moss (2011, p. 66).

A saída  $x = \overline{A \cdot B}$  estará em nível lógico BAIXO somente quando todas as entradas (A, B etc.) estiverem em nível lógico ALTO e em qualquer outra combinação de entradas a saída sempre estará em nível lógico ALTO. A figura 35 mostra como se comporta uma combinação com duas entradas (A, B) e sua respectiva saída ao se utilizar uma porta lógica NAND na representação de ondas digitais.

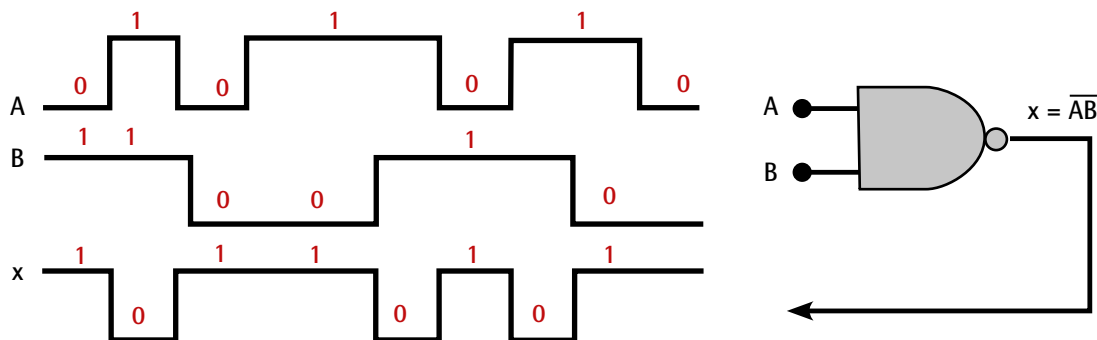


Figura 35 – Ondas digitais A e B e suas respectivas combinações de saída em função do tempo considerando o uso de uma porta NAND

Adaptada de: Tocci, Widmer e Moss (2011, p. 65).

Na figura 35, a saída da porta NAND de duas entradas (A e B) somente estará em nível BAIXO quando ambas as entradas estiverem em nível ALTO e a saída será nível BAIXO para qualquer outra combinação.

### 3.1.6 Operação com porta lógica OU EXCLUSIVO (XOR)

A operação XOR, também conhecida como OU-exclusivo (eXclusive OR), é uma variação da porta lógica OR e é simbolizada por  $x = A \oplus B$ , que corresponde à expressão lógica dada por  $x = \overline{A}B + A\overline{B}$ . A figura 36 mostra uma típica porta lógica XOR com duas entradas (A e B).

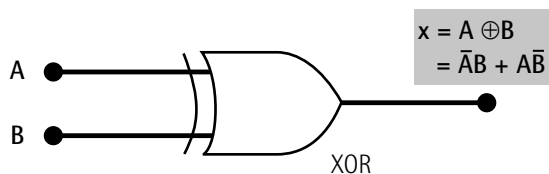


Figura 36 – Porta lógica XOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 818).

A composição da tabela-verdade da porta XOR, considerando duas entradas (A e B), é mostrada na figura 37.

A	B	XOR $A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 37 – Tabela-verdade da operação lógica XOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 818).

Note que, para o caso da função lógica XOR, a saída x será nível BAIXO (bit 0) somente quando as entradas contiverem bits idênticos, como 0 e 0 ou 1 e 1, e saída com nível lógico ALTO para qualquer outra combinação com bits diferentes (0 e 1 ou 1 e 0).

### 3.1.7 Operação com porta lógica NÃO OU EXCLUSIVO (XNOR)

A operação XNOR é uma associação da porta lógica XOR adicionada a uma negação NOT e é simbolizada por  $x = A \oplus B$ , que corresponde à expressão lógica dada por  $x = AB + \bar{A}\bar{B}$ . A figura 38 mostra uma típica porta lógica XNOR com duas entradas (A e B).

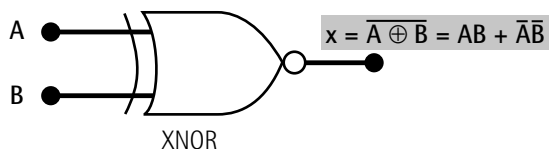


Figura 38 – Porta lógica XNOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 818).

A tabela-verdade de uma porta XNOR, considerando duas entradas (A e B), é mostrada na figura 39.

A	B	XNOR $A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 39 – Tabela-verdade da operação lógica XNOR

Adaptada de: Tocci, Widmer e Moss (2011, p. 818).

Note que, para o caso da função lógica XNOR, a saída x será nível BAIXO (bit 0) somente quando as entradas contiverem bits diferentes, ou seja, quando contiver na entrada bits idênticos como 0 e 0 ou 1 e 1, a saída possuirá nível lógico ALTO.

### 3.1.8 Circuitos lógicos interconectados

Os circuitos lógicos podem ser compostos por várias portas digitais e representar seu comportamento de saída através das operações booleanas apresentadas anteriormente. Por exemplo, considere o circuito da figura 40 que possui três entradas (A, B e C) e uma saída x.

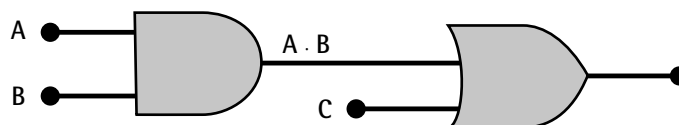


Figura 40 – Circuito lógico composto com duas portas digitais

Adaptada de: Tocci, Widmer e Moss (2011, p. 58).

A expressão lógica resultante na saída é descrita inicialmente através da operação AND entre as entradas A e B, resultando em  $A.B$ , que, na sequência, é novamente combinada através de uma operação OR com uma nova entrada C, resultando na operação  $x = (A.B) + C$ . Em um novo exemplo, descrito na figura 41, as entradas A e B se combinam em uma operação OR formando a primeira saída  $A + B$ , que é novamente recombinaada por uma nova entrada C, resultando na operação de saída  $x = (A + B).C$ .

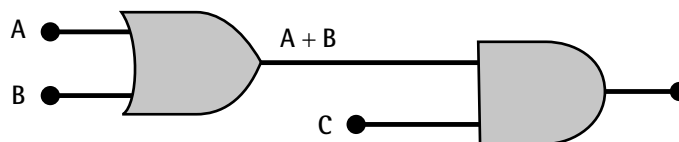


Figura 41 – Circuito lógico composto com duas portas digitais

Adaptada de: Tocci, Widmer e Moss (2011, p. 58).

A partir dos exemplos das figuras anteriores, é possível propor a solução para encontrar a expressão resultante em um circuito com mais componentes lógicos, como o da figura 42.

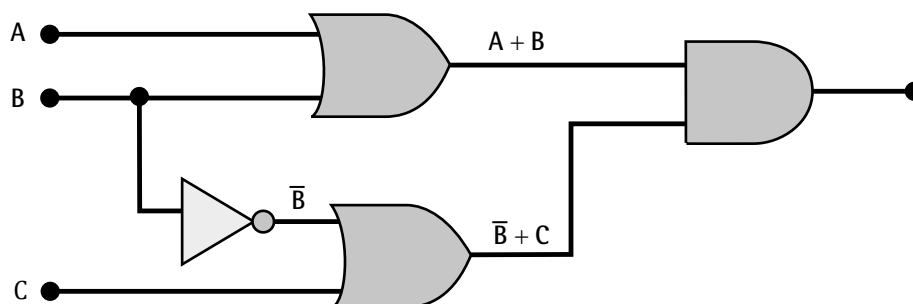


Figura 42 – Circuito lógico composto de quatro portas digitais

Adaptada de: Tocci, Widmer e Moss (2011, p. 63).

A solução para esse exemplo segue como nos casos anteriores, ou seja, deve-se, inicialmente, olhar o circuito e começar a localizar suas conexões sempre da esquerda para a direita. Veja que há uma combinação envolvendo a operação OR entre as entradas A e B, formando a expressão lógica  $A+B$ . Essa expressão serve de entrada para realizar uma combinação AND e não poderá ser finalizada enquanto as entradas envolvidas na parte inferior da figura não estiverem completas. Na parte de baixo da figura, primeiramente, há uma inversão lógica da entrada B, tornando-a  $\bar{B}$ , que, na sequência, é combinada em uma operação OR com a entrada C, formando a expressão de saída dessa operação  $\bar{B}+C$ . Após as operações da parte de cima e da parte de baixo da figura serem finalizadas separadamente, o resultado das duas operações é combinado com a última porta lógica envolvida no processo (AND), resultando na expressão booleana  $x = (A + B) \cdot (\bar{B} + C)$ .



### Saiba mais

Para conhecer um pouco mais sobre portas lógicas, recomendamos o capítulo dois do livro abaixo:

IDOETA, I. V.; CAPUANO, F. G. *Elementos de eletrônica digital*. 40. ed. São Paulo: Érica, 2008.

As portas lógicas são representadas por símbolos padronizados, facilitando a compreensão e o desenho de circuitos digitais. Esses símbolos indicam a função lógica da porta e a quantidade de entradas e saídas. Além dos símbolos, as portas lógicas são frequentemente representadas por tabelas-verdade, que mostram a relação entre as entradas e a saída para todas as combinações possíveis.

A representação esquemática das portas lógicas é fundamental para a comunicação entre engenheiros e técnicos, permitindo que projetos de circuitos digitais sejam facilmente compreendidos e implementados. A padronização dos símbolos garante que qualquer pessoa familiarizada com eletrônica digital possa interpretar os diagramas de circuitos, independente da linguagem ou da região.



As portas lógicas são os blocos de construção fundamentais da eletrônica digital, que encontram aplicações em uma ampla variedade de dispositivos e sistemas. Desde computadores e smartphones até sistemas de controle industrial e equipamentos médicos, as portas lógicas são essenciais para o funcionamento de tecnologias modernas. Algumas aplicações práticas comuns das portas lógicas incluem:

- Processamento de dados em computadores, permitindo a realização de operações matemáticas e lógicas complexas.
- Controle de dispositivos e sistemas, como semáforos, elevadores e máquinas industriais.
- Comunicação digital, incluindo a transmissão e recepção de dados em redes de computadores.
- Armazenamento de dados em memórias digitais, como discos rígidos, memórias flash e unidades de estado sólido.
- Implementação de funções lógicas em sistemas de controle, permitindo a tomada de decisões e a realização de ações automatizadas.

As portas lógicas são os componentes básicos da eletrônica digital, permitindo a realização de operações lógicas e a construção de circuitos digitais complexos. Sua importância reside em sua capacidade de manipular sinais binários, o que possibilita o processamento de informações e a realização de tarefas complexas, como cálculos matemáticos, controle de dispositivos e comunicação digital.

Sem as portas lógicas, a eletrônica digital não seria possível, pois são elas que permitem a criação de sistemas digitais que processam informações de forma rápida, eficiente e precisa. A eletrônica digital, por sua vez, impulsiona a revolução tecnológica que molda o mundo moderno, desde computadores e dispositivos móveis até sistemas de controle industrial e comunicação global.

A capacidade de manipular informações binárias utilizando portas lógicas é crucial para a realização de uma ampla gama de tarefas, o que as torna essenciais para a construção de um mundo cada vez mais digitalizado. Com a combinação de portas lógicas básicas, é possível criar circuitos lógicos simples que realizam funções específicas. Esses circuitos podem ser utilizados para construir blocos de construção mais complexos e, posteriormente, sistemas digitais completos. Alguns desses circuitos lógicos simples são:

- **Circuito XOR (OU Exclusivo):** produz uma saída de 1 apenas se somente uma de suas entradas for 1. Pode ser implementado usando portas NOT, AND e OR.
- **Circuito NAND (NOT AND):** produz a inversão da saída da porta AND, podendo ser implementado combinando uma porta AND com uma porta NOT.
- **Circuito NOR (NOT OR):** produz a inversão da saída da porta OR, podendo ser implementado combinando uma porta OR com uma porta NOT.

Esses circuitos simples podem ser usados como blocos de construção para criar circuitos mais complexos, como contadores, registradores e unidades lógicas aritméticas (ALUs). A capacidade de combinar portas lógicas para criar circuitos complexos é uma das características mais importantes da eletrônica digital.

### Integração das portas lógicas em sistemas digitais

As portas lógicas, ao serem combinadas, formam a base para a construção de sistemas digitais complexos. Esses sistemas são amplamente utilizados em uma variedade de aplicações, como computadores, dispositivos móveis, sistemas de controle industrial, equipamentos de telecomunicações e muitos outros.

A integração de portas lógicas em sistemas digitais envolve a conexão de várias delas de forma organizada para realizar uma função específica. A combinação de portas lógicas pode ser utilizada para criar circuitos mais complexos, como contadores, registradores, unidades lógicas aritméticas (ALUs), memórias e outros blocos de construção essenciais para a eletrônica digital.

Os sistemas digitais modernos são construídos com milhões de portas lógicas integradas em chips de silício. Essa integração de alta densidade permite a criação de dispositivos complexos e eficientes, impulsionando o avanço tecnológico em diversas áreas.

### Simbologia e notação das portas lógicas

A simbologia e a notação utilizadas para representar portas lógicas são essenciais para a compreensão e a comunicação de circuitos digitais. A padronização garante que todos os envolvidos na eletrônica digital possam interpretar os diagramas de circuitos de forma consistente, independentemente do idioma ou da região. Existem diferentes notações para representar portas lógicas, incluindo:

- **Simbologia American National Standards Institute (ANSI):** sistema de simbologia amplamente utilizado nos Estados Unidos e em outros países. Os símbolos ANSI são caracterizados por formas geométricas distintas para cada porta lógica.
- **Simbologia International Electrotechnical Commission (IEC):** sistema de simbologia utilizado internacionalmente, sendo mais comum em países europeus. Os símbolos IEC são caracterizados por formas geométricas mais abstratas, mas ainda facilmente identificáveis.
- **Notação booleana:** essa notação matemática utiliza operadores lógicos para representar as funções das portas lógicas. Por exemplo, a porta AND é representada por um ponto " $\cdot$ " e a porta OR por um sinal de adição "+".

A escolha da notação pode variar de acordo com o contexto e a preferência do projetista. No entanto, é importante que todos os envolvidos em um projeto de circuito digital estejam familiarizados com a simbologia e a notação utilizada para garantir a clareza e a precisão da comunicação.

## 3.2 Clocks (o relógio)

Os processadores e as memórias são dispositivos digitais formados por portas lógicas. Esses circuitos alternam entre dois estados (0 e 1) milhões de vezes por segundo enquanto executam instruções específicas de um programa. Para que essas operações ocorram de forma coordenada e eficiente, é necessário um mecanismo de sincronização: o relógio, ou clock.

O clock funciona como um gerador de pulsos regulares, medidos em ciclos, que controlam o ritmo das operações. Cada pulso alterna entre níveis lógicos – alto (1) e baixo (0) – permitindo sincronizar e orquestrar as atividades do processador. Assim, o clock pode ser visto como o maestro que mantém todos os componentes do sistema trabalhando em harmonia (Monteiro, 2019). O clock nada mais é do que um contador de tempo, desenvolvido para gerar pulsos, cuja duração é denominada de ciclo de clock. De forma geral, os pulsos comutam de valor para intensidade alta, que corresponde ao bit 1, para a intensidade baixa, que corresponde ao bit 0. Essa alternância de estado faz com que seja possível sincronizar e cadenciar as ações ou atividades do processador. De certa forma, o clock também pode ser entendido como um dispositivo que realiza um controle, sincronizando todos os componentes do computador. figura 43 mostra um esquema com um conjunto de pulsos gerados por um clock e alguns de seus elementos principais, sendo eles:

- **Ciclo de clock ou ciclo de relógio:** determina o intervalo de tempo do início da borda de subida (ou descida) do pulso até o início da próxima borda de subida (ou descida) do outro pulso.
- **Período ou ciclo de tempo:** é o intervalo de tempo necessário para que o pulso execute uma oscilação completa.
- **Borda de subida:** é constituído pelo período utilizado pelo pulso para realizar a transição de subida.
- **Borda de descida:** é constituído pelo período utilizado pelo pulso para realizar a transição de descida.
- **Frequência ou taxa de clock:** é a quantidade de ciclos por segundo do clock, determinada também pelo inverso do período e medido em Hertz (Hz), onde 1 Hz será igual a um ciclo por segundo.

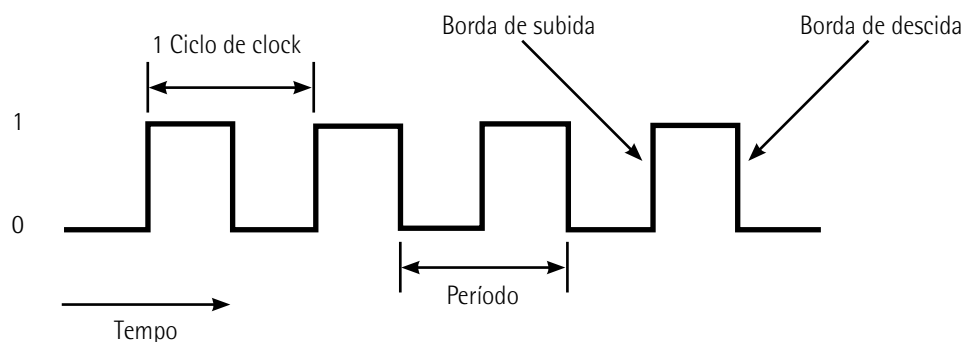


Figura 43 – Diagrama de pulsos de clock

Adaptada de: Monteiro (2019, p. 173).

A figura 44 mostra um diagrama de ciclo de clock original ( $T_0$ ) de um processador constituído por outros cinco pulsos gerados ( $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$  e  $T_5$ ). O ciclo de clock geralmente é relacionado a uma operação elementar que ocorre durante o ciclo de instrução. Porém, mesmo as operações elementares não se realizam em um único passo de ciclo, de forma que um ciclo pode se subdividir em outros ciclos menores, conhecidos como subciclos. Os subciclos podem estar defasados no tempo, de maneira que cada um pode acionar um passo diferente da operação elementar inicial. Essas diferenças de operações do ciclo fundamental também são conhecidas como microoperações e possibilitam, entre outras coisas, o processamento paralelo.

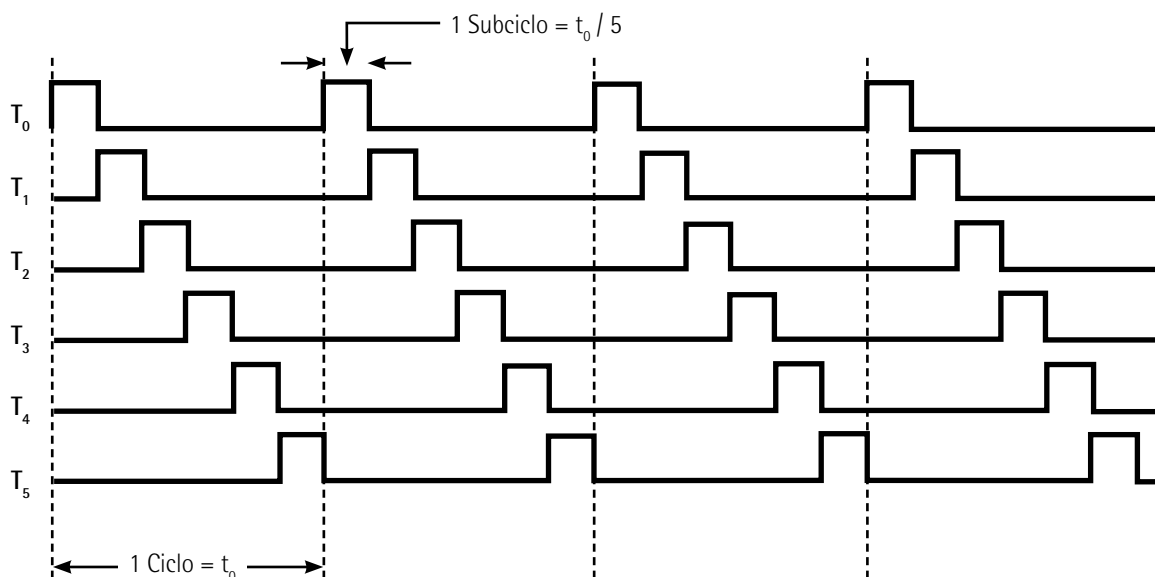


Figura 44 – Diagrama de ciclos de um processador com cinco subciclos

Fonte: Monteiro (2019, p. 175).

### 3.2.1 Taxa de execução de instruções por segundo

Como já definido antes, um processador é controlado através de um clock em uma frequência constante  $f$  ou, de forma equivalente, por um tempo de ciclo constante  $\tau$ , onde:  $\tau = \frac{1}{f}$ .

#### Exemplo de aplicação

A partir da frequência de operação de um processador de 133 MHz, encontre o tempo de duração de cada ciclo.

#### Resolução

Utilizando a fórmula  $\tau = \frac{1}{f}$ , tem-se  $\rightarrow \tau = \frac{1}{133 \times 10^6} = 7,51 \times 10^{-9}$  segundos ou 7,51 nanosegundos (ns).

A tabela 1 mostra alguns dos prefixos para valores na base decimal (base 10) e base binária (base 2), organizados pela International Electrotechnical Commission.

**Tabela 1 – Prefixo de valores e bases utilizados em computação**

Pref.	Símb.	Potência de 10	Potência de 2	Pref.	Símb.	Potência de 10	Potência de 2
Kilo	K	1 mil = $10^3$	$2^{10} = 1024$	Mili	m	1 milésimo = $10^{-3}$	$2^{-10}$
Mega	M	1 milhão = $10^6$	$2^{20}$	Micro	$\mu$	1 milionésimo = $10^{-6}$	$2^{-20}$
Giga	G	1 bilhão = $10^9$	$2^{30}$	Nano	n	1 bilionésimo = $10^{-9}$	$2^{-30}$
Tera	T	1 trilhão = $10^{12}$	$2^{40}$	Pico	p	1 trilionésimo = $10^{-12}$	$2^{-40}$
Peta	P	1 quadrilhão = $10^{15}$	$2^{50}$	Femto	f	1 quadrilionésimo = $10^{-15}$	$2^{-50}$
Exa	E	1 quintilhão = $10^{18}$	$2^{60}$	Atto	a	1 quintilionésimo = $10^{-18}$	$2^{-60}$
Zetta	Z	1 sextilhão = $10^{21}$	$2^{70}$	Zepto	z	1 sextilionésimo = $10^{-21}$	$2^{-70}$
Yotta	Y	1 setilhão = $10^{24}$	$2^{80}$	Yocto	y	1 setilionésimo = $10^{-24}$	$2^{-80}$

Adaptado de: Null e Lobur (2006, p. 39).

Outro fator preponderante no cálculo da execução de instruções é a contagem de instruções  $I_c$  para um certo programa que execute todas as instruções até um intervalo de tempo definido. Dessa forma, pode-se determinar um parâmetro importante que é a média de ciclos por instrução, Cycles Per Instruction (CPI). Para instruções que exigem o mesmo número de ciclos de clock a serem executados, a CPI será dada por um valor constante, de acordo com a frequência do processador. Dessa forma, é possível calcular o tempo  $T$  necessário para a execução de um determinado programa, dado por  $T = I_c \times CPI \times \tau$ .

## Exemplos de aplicação

**Exemplo 1.** Qual será o tempo total de resposta para a execução de um determinado programa que possua 440 instruções ( $I_c$ ), 8 ciclos por instrução e uma frequência ( $f$ ) de processamento de 1,4 GHz?

### Resolução

$T = I_c \times CPI \times \tau$ , mas é importante lembrar que  $\tau = \frac{1}{f}$ , então:

$$T = I_c \times CPI \times \frac{1}{f} \rightarrow T = 440 \times 8 \times \left( \frac{1}{1,4 \times 10^9} \right) \rightarrow T = 2,51 \text{ microssegundos } (2,51 \mu s).$$

Outra medida muito utilizada para o desempenho de um processador é a taxa de execução expressa em milhões de instruções por segundo, Millions of Instructions Per Second (MIPS), dada por:  $\frac{f}{CPI \times 10^6}$ .

**Exemplo 2.** Considere a execução de um programa contendo 1000 instruções, executados em um processador capaz de operar 0,6 ciclos por instrução a uma frequência de 200 MHz. Qual será a quantidade de MIPS para esses valores?

### Resolução

Substituindo na fórmula por segundo.

$$\frac{f}{CPI \times 10^6} \rightarrow \frac{200 \times 10^6}{0,6 \times 10^6} = 333,33 \text{ MIPS ou } 333,33 \text{ milhões de instruções}$$

### 3.3 Máquina de von Neumann

#### 3.3.1 Computador IAS

Embora atualmente a microarquitetura de um processador seja bem aceita, no início da computação, principalmente durante as gerações dos computadores da década de 1940/50, os computadores ainda não tinham um padrão bem definido de quais componentes eram necessários para que seu funcionamento ocorresse de forma otimizada, assim como não tinham uma estruturação de quais componentes seriam necessários para melhor tratar dados/instruções. Coube ao matemático húngaro John von Neumann (1903 – 1957), que trabalhava no Instituto de Estudos Avançados da Universidade de Princeton, nos EUA, realizar modificações no projeto do computador Electronic Numerical Integrator and Computer (ENIAC), em 1946.

As tarefas de processamento e armazenamento de dados no ENIAC eram extremamente enfadonhas. Von Neumann verificou que a programação poderia ser facilitada se o programa fosse representado de forma adequada para a realização do armazenamento na memória juntamente com os dados. Assim, um computador poderia obter suas instruções lendo-as diretamente da memória e um programa poderia ser criado ou alterado através de endereços da memória, de forma que essa ideia ficou conhecida como conceito de programa armazenado (Stallings, 2010). A publicação do novo projeto de von Neumann ocorreu em 1945 e foi a base para a construção de um computador com uma nova proposta de arquitetura, o Electronic Discrete Variable Computer (EDVAC), finalizado em 1949. Após o término do projeto do EDVAC, von Neumann se dedicou à construção de um novo computador desenvolvido nos laboratórios do Instituto de Estudos Avançados de Princeton, o computador Institute of Advanced Studies (IAS), como ficou conhecido. O IAS ficou pronto em 1952 e é conhecido como o protótipo para todos os computadores modernos de uso geral. A figura 45 mostra como foi projetada a estrutura geral do computador IAS.

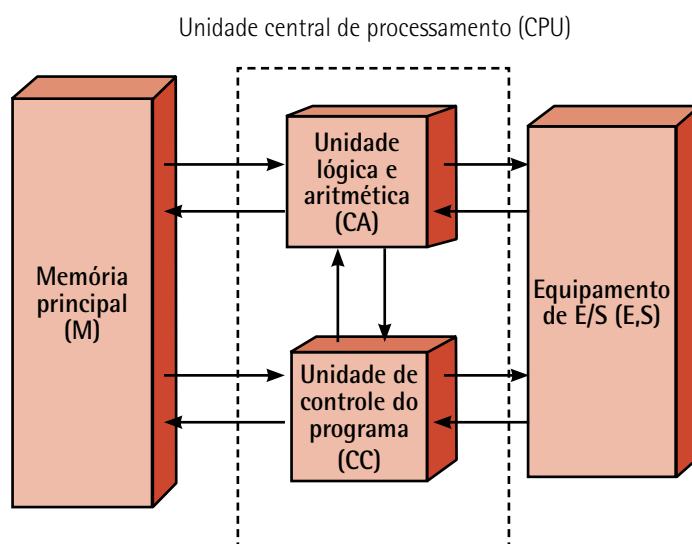


Figura 45 – Estrutura simplificada de um computador IAS

A partir de suas criações, von Neumann definiu que todos os computadores deveriam ter as seguintes características: memória principal para armazenamento dos dados; unidade lógica e aritmética (ULA) para realizar operações em dados binários; unidade de controle para interpretar/executar todas as instruções da memória principal; e dispositivos de entrada e saída (E/S) controlados pela unidade de controle.

Segundo von Neumann, um computador precisa ser capaz de realizar as operações elementares da aritmética (adição, subtração, multiplicação e divisão). Assim, é imprescindível que ele contenha as unidades especializadas para realizar tais tarefas. O controle lógico do dispositivo tem como função organizar a sequência apropriada de como as operações devem ser executadas. Von Neumann estabeleceu também que, para que os dispositivos possam executar operações em sequências longas e complicadas, se faz necessário o uso de uma memória que seja capaz de armazenar um volume grande de dados. Um computador deve ser capaz de estabelecer contato de entrada e saída com dispositivos internos/externos com a finalidade de ler/gravar dados. Como praticamente todos os computadores atuais têm a mesma função e estrutura, eles também são conhecidos como máquinas de von Neumann.

A memória do computador IAS era constituída em 1000 locais para armazenamento, chamados de palavras (words), com 40 dígitos binários. Uma palavra pode ser definida como um conjunto ordenado de bytes na qual a informação pode ser armazenada, processada e transmitida dentro de um computador. Geralmente, um processador possui um conjunto de instruções de tamanho fixo, de forma que o tamanho da instrução será igual ao tamanho da palavra. Por exemplo, um computador de 64 bits processará palavras de 64 bits ou mesmo duas palavras de 32 bits. A figura 46 mostra a formatação desses dados onde cada número é representado por um bit de sinal e um valor de 39 bits contendo a palavra.

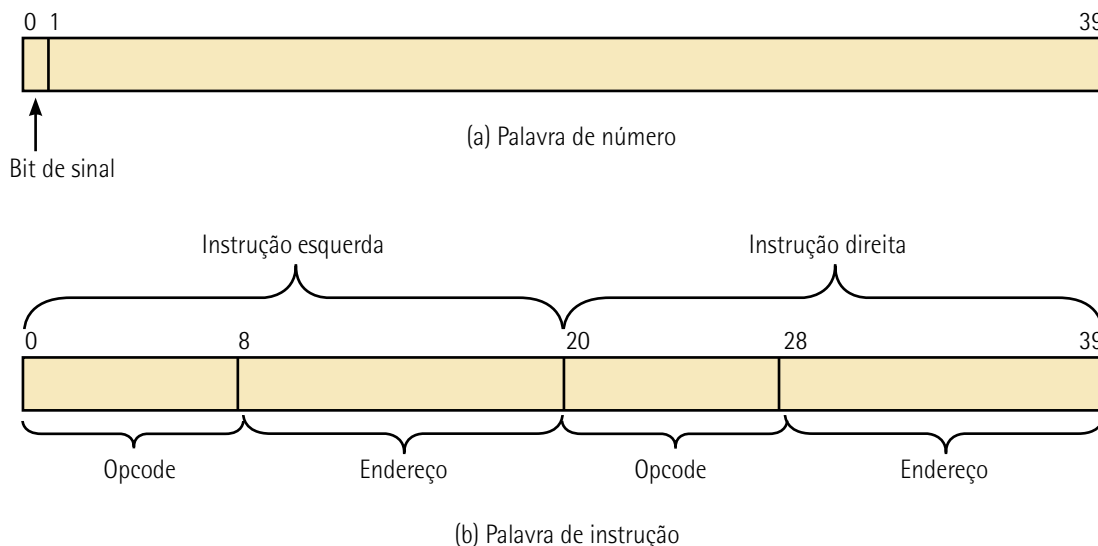


Figura 46 – Estrutura simplificada de um computador IAS

Fonte: Stallings (2010, p. 15).

Uma palavra pode ser dividida em duas instruções de 20 bits (figura 46), onde cada instrução é composta por um opcode (código de operação) de 8 bits, sendo responsável por indicar a operação a ser executada, e um endereço de 12 bits, que aponta para uma localização específica na memória, numerada de 0 a 999. A arquitetura de von Neumann foi projetada com base no uso de componentes de memória, como registradores, que desempenham um papel fundamental no suporte às tarefas da unidade de controle e da unidade lógica aritmética, garantindo eficiência no processamento das instruções.

### 3.4 Chips de memória

Os chips de memória são componentes essenciais em dispositivos eletrônicos, armazenando dados e instruções para o funcionamento dos sistemas. Existem diversos tipos de chips de memória, cada um com características e aplicações específicas:

- **Static Random Access Memory (SRAM):** caracterizada por alta velocidade de acesso e baixo consumo de energia, ideal para caches e registradores.
- **Dynamic Random Access Memory (DRAM):** mais comum em computadores, com maior capacidade de armazenamento e menor custo, porém com velocidade de acesso menor que a SRAM.
- **Synchronous Dynamic Random Access Memory (SDRAM):** versão sincronizada da DRAM, com maior velocidade e desempenho em relação à DRAM tradicional.
- **Double Data Rate (DDR):** evolução da SDRAM, com taxas de transferência de dados duplicadas, oferecendo maior desempenho.
- **NAND:** tipo de memória flash não volátil, utilizada em dispositivos de armazenamento como pen drives e cartões SD, com alta densidade e menor consumo de energia.

Os chips de memória desempenham um papel fundamental na operação de diversos dispositivos eletrônicos, armazenando dados e instruções essenciais para o funcionamento dos sistemas. Sua funcionalidade se baseia na capacidade de armazenar informações de forma organizada e acessível, permitindo que os dispositivos executem tarefas complexas de forma eficiente.

A aplicação dos chips de memória é ampla, abrangendo uma variedade de dispositivos eletrônicos, desde computadores pessoais e smartphones até servidores e dispositivos Internet das Coisas (IoT). Em computadores, os chips de memória armazenam o sistema operacional, programas e dados em uso ativo, garantindo um acesso rápido e eficiente à informação.

Em dispositivos móveis, os chips de memória armazenam aplicativos, fotos, músicas e outros dados, proporcionando uma experiência fluida e eficiente para o usuário. Nos servidores, os chips de memória são essenciais para o processamento de grandes volumes de dados, garantindo alta performance e confiabilidade para as aplicações.



A tecnologia de chips de memória tem evoluído significativamente ao longo dos anos, impulsionada pela demanda por maior capacidade, velocidade e eficiência energética. Os avanços na litografia e na fabricação de semicondutores permitiram a miniaturização dos chips de memória, aumentando sua densidade e capacidade de armazenamento.

A introdução de novas tecnologias, como a Double Data Rate (DDR) e a memória flash NAND, permitiu aumentos substanciais nas taxas de transferência de dados e na eficiência energética, proporcionando um desempenho aprimorado em dispositivos eletrônicos.

A pesquisa e desenvolvimento contínuos na área de chips de memória estão direcionados para a criação de tecnologias mais avançadas, com foco em maior capacidade, velocidade, menor consumo de energia e maior confiabilidade.

Os chips de memória desempenham um papel essencial na tecnologia moderna, impulsionando o desenvolvimento de dispositivos eletrônicos cada vez mais sofisticados e eficientes. Com a crescente demanda por maior capacidade, velocidade e eficiência energética, a pesquisa e desenvolvimento na área de chips de memória continuam a avançar a passos largos.

As tendências futuras incluem a miniaturização ainda maior dos chips de memória, o desenvolvimento de novas tecnologias, como as memórias Magnetoresistive Random Access Memory (MRAM) e 3D NAND, e a integração de chips de memória com outros componentes eletrônicos, como processadores e sensores.

A evolução dos chips de memória é fundamental para atender às necessidades de um mundo cada vez mais digital, impulsionando a inovação e o progresso tecnológico em diversas áreas, desde a computação pessoal e a comunicação móvel até a inteligência artificial e a internet das coisas.

### 3.5 Organização de memória

A memória humana é um sistema complexo que envolve a retenção e recuperação de informações. Existem vários tipos de memória, cada um com suas características e funções específicas.

- **Memória de curto prazo (MPT):** armazena informações por um curto período de tempo, geralmente por alguns segundos ou minutos, e tem capacidade limitada. É responsável pelo processamento de informações presentes no momento, como lembrar um número de telefone por alguns segundos.
- **Memória de longo prazo (MPL):** armazena informações por um período mais longo, desde minutos até anos, e tem capacidade ilimitada. É responsável por armazenar informações de eventos passados, conhecimentos e habilidades.
- **Memória operacional (MO):** é um sistema de memória de curto prazo que permite manipular e integrar informações durante tarefas complexas. Envolve a retenção temporária de informações enquanto elas são usadas para realizar uma tarefa, como resolver um problema matemático ou lembrar instruções.

### 3.5.1 Memória de curto prazo (MPT)

A MPT funciona como um buffer temporário para informações recebidas pelos sentidos, sendo responsável por armazenar informações que são relevantes para a tarefa atual, como lembrar o que você leu na última frase. A capacidade da MPT é limitada, sendo possível armazenar em torno de 7 itens distintos por vez. Se a informação não for repetida ou transferida para a MPL, ela será esquecida rapidamente. Imagine, por exemplo, que você está lendo um livro, a MPT (figura 47) está armazenando as informações da última frase que você leu, para que você possa conectar as ideias e compreender o texto como um todo.



Figura 47 – Memória de curto prazo (MPT), imagem gerada em IA Fux pro

### 3.5.2 Memória de longo prazo (MPL)

A MPL é um sistema de armazenamento de informações mais permanente, onde as informações relevantes são transferidas da MPT após repetição e processamento. Dentro da MPL, temos a memória declarativa, que armazena informações sobre fatos, eventos e conceitos e que é dividida em:

- **Memória episódica:** armazena informações sobre eventos específicos da nossa vida, como o dia do seu casamento ou a primeira vez que você andou de bicicleta.
- **Memória semântica:** armazena informações gerais sobre o mundo, como o significado de palavras, nomes de países e datas históricas.
- **Memória procedimental:** armazena informações sobre como realizar ações, por exemplo, andar de bicicleta ou tocar um instrumento musical.

Imagine, por exemplo, que você está aprendendo um novo idioma. As palavras que você está aprendendo são armazenadas na sua memória semântica; a memória procedimental permite que você aprenda a conjugar os verbos e a formar frases; já a memória episódica guarda a lembrança de quando você aprendeu essa nova língua, incluindo o local, a data e as pessoas que estavam presentes.

### 3.5.3 Memória operacional (MO)

A MO é um sistema de memória crucial para o desempenho cognitivo, atuando como um centro de controle para a realização de tarefas complexas. Ela permite que você retenha informações importantes, enquanto trabalha com elas, manipulando e integrando-as para realizar uma tarefa. Imagine que você precisa realizar uma receita de bolo, a MO (figura 48) mantém as informações importantes, como a sequência de ingredientes e os passos da receita, enquanto você trabalha para preparar o bolo. Ela também integra as informações, como a quantidade de cada ingrediente, para que você possa fazer o bolo de acordo com a receita.



Figura 48 – Memória de curto prazo (MO), imagem gerada em IA Fux pro

Vários fatores podem influenciar a capacidade de retenção e recuperação de informações, entre eles se destacam:

- **Fatores biológicos:** a idade, o estado de saúde, o nível de estresse e o sono podem influenciar o funcionamento da memória.
- **Fatores psicológicos:** a atenção, a motivação, a emoção e o estado de ânimo podem afetar a codificação e a recuperação de informações.
- **Fatores ambientais:** a condição do ambiente, como o nível de ruído, a iluminação e a temperatura, pode influenciar o desempenho da memória.
- **Fatores culturais:** a cultura, como o idioma, as crenças e os valores, pode influenciar a forma como a memória é organizada e como as informações são armazenadas.

Existem diversas estratégias que podem contribuir para melhorar a capacidade de memorização e retenção de informações. Algumas delas são:

- **Repetição:** repetir a informação que você deseja memorizar várias vezes ajuda a fortalecer a conexão neural e facilita a recuperação. Use flashcards, revisão de conteúdo ou outros métodos de repetição espaçada.
- **Organização:** organizar as informações em categorias, hierarquias ou listas facilita o aprendizado e a recuperação. Use mapas mentais, resumos ou outros métodos de organização.
- **Associação:** associar novas informações a algo que você já sabe, como imagens, histórias ou músicas, facilita a memorização. Use a técnica da memória por associação, criando conexões entre novas informações e informações existentes.
- **Concentração:** evite distrações e concentre-se totalmente na informação que você deseja memorizar. Use técnicas de concentração como meditação, respiração profunda ou crie um ambiente propício para o aprendizado.
- **Sono:** o sono é essencial para a consolidação da memória. Dormir bem permite que o cérebro processe as informações aprendidas durante o dia e as armazene na memória de longo prazo.

Lembre-se de que a melhoria da memória é um processo gradual e exige prática e persistência. Aplique as estratégias de forma consistente e observe como sua capacidade de memorização aumenta ao longo do tempo.

### 3.6 Chips de CPU

Um chip de CPU é basicamente dividido em três tipos: endereço, dados e controle. Dessa forma, cada tipo pode ser conectado a pinos similares na memória e conectados a chips de E/S através de um conjunto de fios paralelos, conhecidos também como barramentos. Ao buscar uma instrução na memória, inicialmente, a CPU coloca o endereço da posição de memória desejada daquela instrução em seus pinos de endereços. Após esse primeiro passo, ela ativa uma ou mais linhas de controle com a finalidade de informar à memória que ela necessita ler uma palavra. A partir disso, a memória responde colocando a palavra requisitada nos pinos de dados da CPU e ativa um sinal que informará o que acabou de ser realizado. Ao receber esse sinal, a CPU aceita a palavra e executa a instrução solicitada; tal instrução pode ser uma requisição para realizar uma leitura ou uma escrita de palavra de dados, onde todo o processo deverá ser repetido para cada palavra adicional processada.

Nos chips de CPU existem dois parâmetros fundamentais que determinam seu desempenho: a quantidade de pinos de endereços e o número de pinos para dados. Em um chip com  $m$  pinos de endereços, é possível realizar o endereçamento de até  $2^m$  localizações de memória. Os valores comuns do número de pinos  $m$  são 16, 32 e 64 (Tanenbaum; Austin, 2013). De maneira semelhante, um chip consistindo em  $n$  pinos de dados pode ler ou escrever uma palavra de  $n$  bits em uma única operação. Assim, um chip de 64 pinos de dados será muito rápido, porém terá um custo muito maior. Além dos pinos para endereçamento e dados, a CPU também possui alguns pinos de controle, que servem para

regular o fluxo e a temporização de dados oriundos da CPU e aqueles que passam por ela, além de outras funcionalidades.

As CPUs também possuem pinos para alimentação de energia elétrica (entre 1,2 V a 1,5 V), um pino de aterramento e um pino para o sinal de sincronismo de clock (onda quadrada de frequência bem definida). Existem outros diversos pinos que também realizam o controle e que podem ser agrupados em categorias, como: controle de barramento; interrupções; arbitragem de barramento; sinalização de coprocessador; estado; e diversos.

A figura 49 mostra um chip de CPU genérico baseado no uso desses grupos de pinos de sinais.

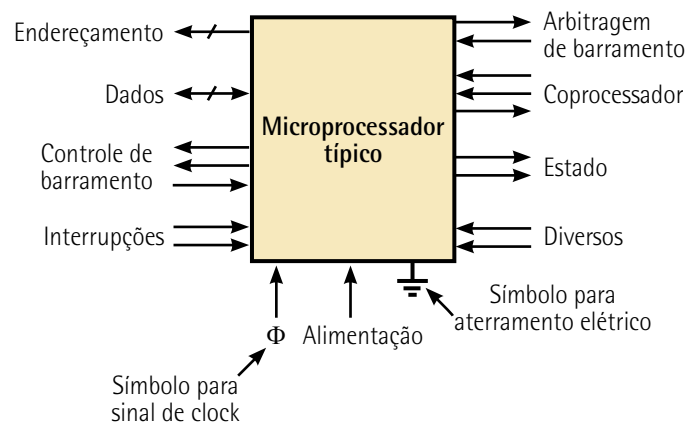


Figura 49 – Pinagem lógica de uma CPU genérica

Fonte: Tanenbaum e Austin (2013, p. 147).

Como podemos observar na figura, as setas indicam sinais de entrada e de saída, os segmentos de reta diagonais indicam que são utilizados vários pinos para realizar a comunicação de dados e instruções, a pinagem para alimentação elétrica, aterramento e a entrada de sincronismo de clock.

### **Observação**

A maioria dos pinos de controle do barramento são saídas de comunicação da CPU para o barramento, sendo, assim, linhas de comunicações para E/S e para memória principal.

Outros pinos como os de interrupção são entradas oriundas dos dispositivos de E/S que se conectam à CPU. A principal função nesse caso é o de controlar algum dispositivo de E/S, ativando um sinal dos pinos para interromper a CPU ou fazê-la realizar algum tipo de serviço para o dispositivo de E/S como, por exemplo, a verificação de ocorrências de erros de comunicação de E/S. Algumas CPUs possuem um pino de saída para confirmação do sinal de interrupção. Os pinos de arbitragem de barramento são utilizados para realizar o controle do tráfego no barramento de modo a impedir que dois ou mais dispositivos tentem usá-lo simultaneamente. Outros chips de CPU são projetados para funcionar como processadores auxiliares ou coprocessadores, como os de ponto flutuante, além de outros coprocessadores especializados em processamento gráfico.

### 3.7 Intel Core i7

O processador Intel i7, lançado na sua primeira geração em 2008, é uma evolução da CPU Intel 8088 com 29 mil transistores utilizada no desktop IBM PC. Em 2008, o i7 possuía cerca de 731 milhões de transistores distribuídos em quatro processadores operando em uma frequência de clock de 3,2 GHz (3,2 bilhões de hertz ou ciclos por segundo), utilizando uma largura de linha (fios de cobre de ligação entre os transistores) de 45 nanômetros ( $45 \times 10^{-9}$  metros).

Quanto menor for a largura de linha, que também implica transistores menores, mais transistores são empacotados no chip, aumentando, assim, o poder de processamento da CPU. A primeira geração da arquitetura i7 era baseada na arquitetura "Nahalem", que na época substituiu a arquitetura "Core", evoluindo, por fim, para uma arquitetura "Sandy Bridge" composta por 1,16 bilhões de transistores, trabalhando a uma velocidade de até 3,5 GHz, com largura de linha de 32 nanômetros (Tanenbaum; Austin, 2013).

O i7 também difere dos processadores mais antigos por ser uma máquina completa de 64 bits (tamanho da palavra), multicore (múltiplos núcleos) e vendida com um número variável de núcleos, que vão de 2 a 6. O i7 também é compatível para uso de processamento paralelo e threads de hardware e software. O sistema de Hyperthreading, também conhecido como Multithreading simultâneo, permite que latências muito curtas (quantidade de tempo para acesso a outro hardware) sejam utilizadas quando há falta de memória cache.

Além disso, todos os processadores Core i7 têm três níveis de memória cache (L1, L2 e L3), onde cada núcleo possui uma cache de dados L1 (layer 1) com 32 kB (kilobytes) e uma cache L1 de instruções também com 32 kB. Os núcleos dos processadores i7 também possuem sua própria cache L2 com 256 kB, além de compartilharem, entre todos os núcleos, uma cache unificada L3 com tamanhos que variam de 4 a 15 MB (megabytes).

Na figura 50, é possível ver a estrutura interna de um processador i7 e suas subdivisões entre 4 núcleos e uma memória cache L3 compartilhada, além de um coprocessador gráfico.

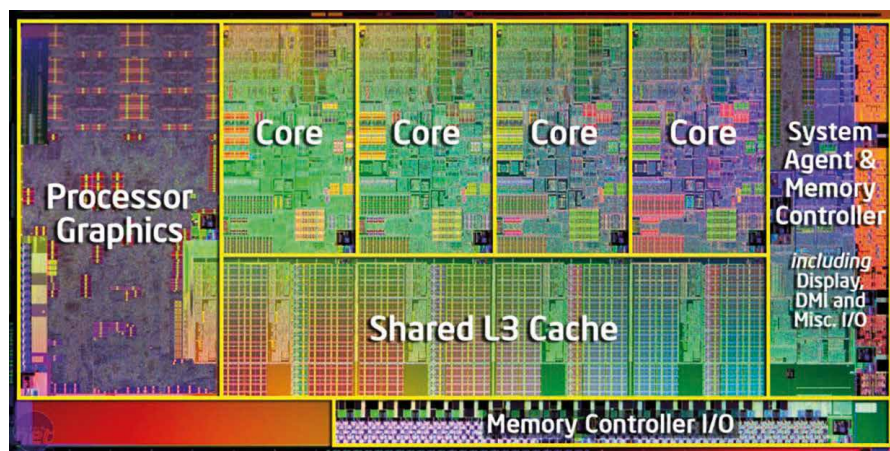


Figura 50 – Estrutura interna subdividida de um processador Intel i7



Como todos os processadores Core i7 possuem múltiplos núcleos com caches específicas para dados, um problema ocorre quando a CPU modifica a palavra na cache privada que também está contida em outro núcleo. Se outro processador tentar ler a mesma palavra da memória, poderá obter um valor ultrapassado, visto que as palavras de cache modificadas não são escritas imediatamente de volta na memória. Assim, para manter uma consistência de informações da memória, cada núcleo em um sistema microprocessado realiza a operação snoops (escuta) no barramento de memória, em busca de referências de palavras contidas na cache. Quando ela acha uma dessas referências, são fornecidos os dados antes que a memória faça a leitura deles.

Outro problema encontrado no i7 refere-se ao consumo de energia, que também ocorre com outras CPUs. Um processador Core i7 consome entre 17 a 150 watts, dependendo de seu modelo ou frequência de operação. A fim de impedir que algum dano ocorra no silício devido ao aquecimento excessivo, em todo projeto de processadores, a Intel utiliza métodos de resfriamento e dissipadores de calor para que este fique afastado do substrato do processador e evita que o silício queime.

O projeto do Core i7 também possui um soquete de conexões Land Grid Array (LGA) quadrado com 37,5 mm (milímetros) de borda e 1155 pinos em sua parte inferior onde 286 são para alimentação e 360 para aterramento. Os pinos estão dispostos como um quadrado de 40 x 40 pinos com 17 x 25 pinos faltando no meio. Outros 20 pinos também estão faltando no perímetro, de forma assimétrica, a fim de impedir que o chip seja inserido de forma incorreta na sua base. Essa disposição física da pinagem é observada na figura 51.

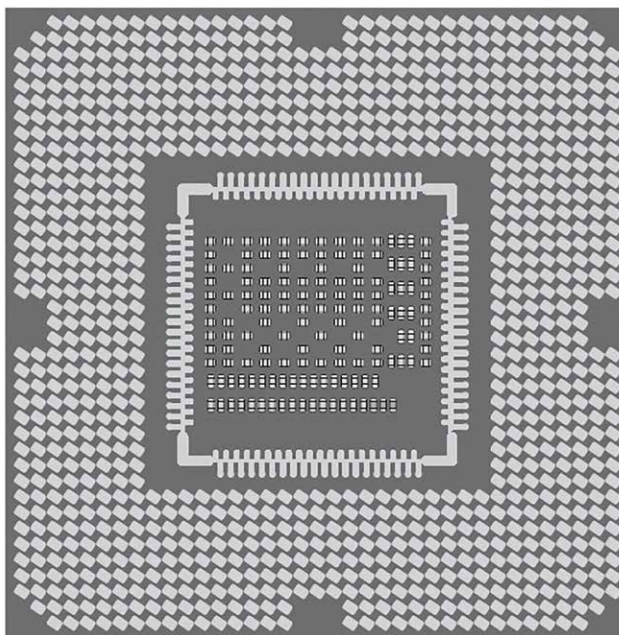


Figura 51 – Disposição física dos pinos do Intel Core i7

Fonte: Tanenbaum e Austin (2013, p. 160).

A pinagem lógica de funcionamento é mostrada na figura 52, onde é possível observar, no seu lado esquerdo, que há cinco grupos principais de sinais de barramento incluindo comunicações com a memória principal através de comunicação Double Data Rate (DDR) e, no seu lado direito, uma diversidade de sinais para diagnósticos, monitoramento térmico, detecção e gerenciamento de energia, entre outros.

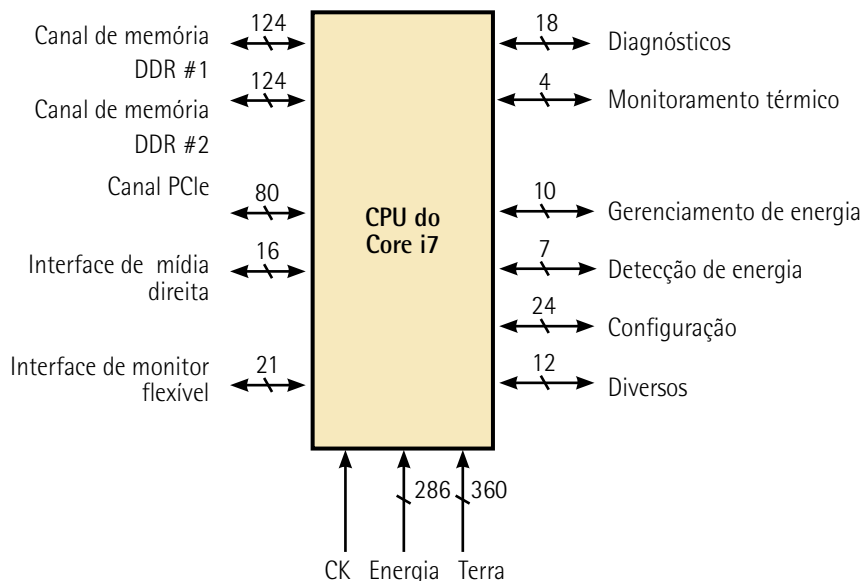


Figura 52 – Pinagem lógica do Intel Core i7

Fonte: Tanenbaum e Austin (2013, p. 161).

A partir da vista externa de um processador com a discussão de sua pinagem lógica e física, é necessária a abordagem das características e do funcionamento interno de um processador, onde ele pode ser organizado de forma funcional em quatro categorias:

- **Buscar instruções:** o processador deve ser capaz de realizar a leitura de uma ou de várias instruções que ocupam endereços específicos na memória (registrador, cache, memória principal).
- **Interpretar instruções:** as instruções são decodificadas a fim de se determinar qual ação é requerida naquele momento.
- **Obter dados:** o ato da execução de uma ou mais instruções pode requerer a leitura de dados da memória ou algum dispositivo de E/S.
- **Processar dados:** a execução de instruções pode requerer a realização de operações lógicas e/ou aritméticas com os dados.
- **Gravar dados:** os resultados obtidos através do processamento de dados podem ser gravados na memória ou dispositivos de E/S.



## 3.7.1 Microarquitetura de processadores

A microarquitetura de um processador é constituída por um conjunto de instruções contendo operações complexas além de três subsistemas principais: unidades de execução, banco de registradores e lógica de controle. Essas unidades também podem ser descritas como o caminho dos dados do processador, pois os dados e instruções fluem regularmente por eles (Carter, 2002).

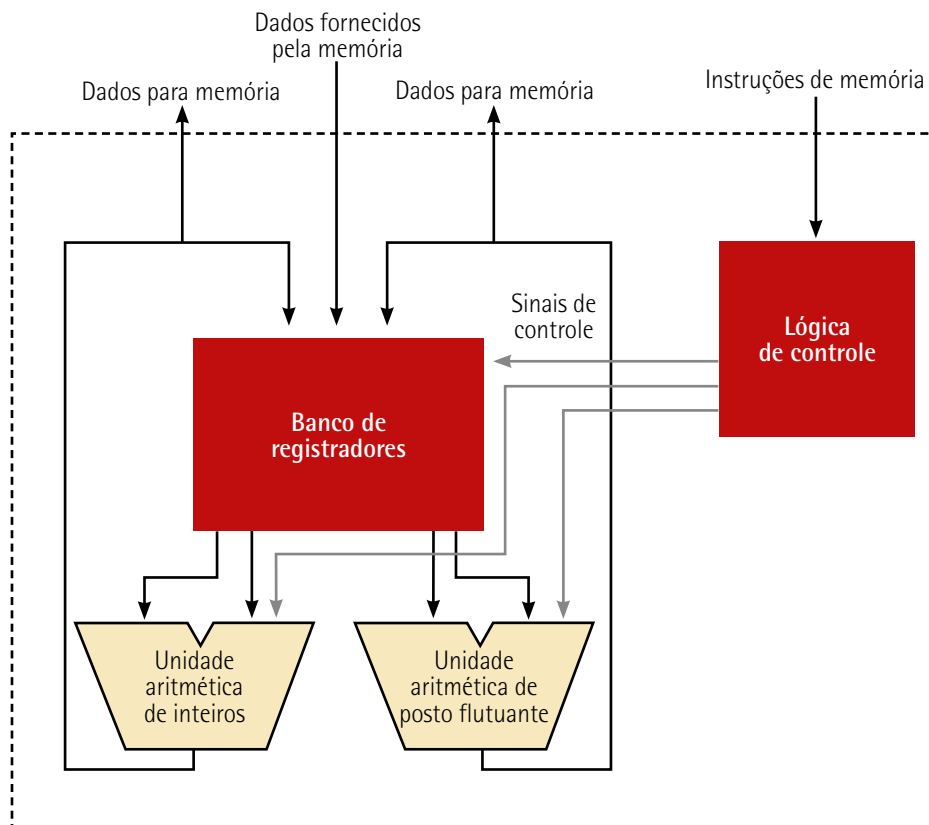


Figura 53 – Diagrama de blocos de um processador moderno

Adaptada de: Carter (2002, p. 49).

O diagrama da figura 53 exemplifica quais são as etapas envolvidas na execução de uma ou mais instruções e como os módulos do processador interagem entre si durante o processamento e decodificação da instrução. Inicialmente, o processador busca a instrução que está contida na memória principal, externa ao processador, e a direciona para ser decodificada pela lógica de controle. Após a decodificação da instrução, esta é representada em um padrão de bits a fim de informar ao hardware como ela deverá ser executada. Esse padrão de bits é, então, enviado para a próxima sessão da unidade de execução da instrução, através de sinais de controle, que lê as entradas da instrução na memória interna do processador (banco de registradores). Assim que a instrução é decodificada e os valores armazenados nos registradores, as instruções são executadas em uma unidade lógica aritmética de inteiros e/ou ponto flutuante a fim de se obter resposta ao processamento desejado. Por fim, os dados obtidos são devolvidos como resposta à memória principal do computador, primeiramente, passando pelo banco de registradores através do barramento interno do processador.

### 3.7.2 Unidade Lógica e Aritmética (ULA)

#### Observação

A Unidade Lógica Aritmética é o principal dispositivo do processador e realiza efetivamente as operações lógicas/matemáticas sobre dados e instruções.

Qualquer ULA é considerada um conjunto de circuitos lógicos e componentes eletrônicos simples que, ao se integrarem, realizam operações lógicas (AND, OR, XOR), operações de deslocamento, incremento, complemento, soma, subtração, multiplicação e divisão (Monteiro, 2019). Uma ULA, geralmente, possui duas entradas de dados conectadas à saída (resultado da operação efetuada): uma entrada para sinais de controle para determinação da operação a ser realizada; e saídas de comunicação com registradores e para sinalização de flags. Em processadores mais antigos, o barramento interno para dados é utilizado para interligar a ULA ao registrador acumulador, accumulator (AC ou ACC), e aos demais registradores, e na sequência à memória principal, conforme observado na figura 54.

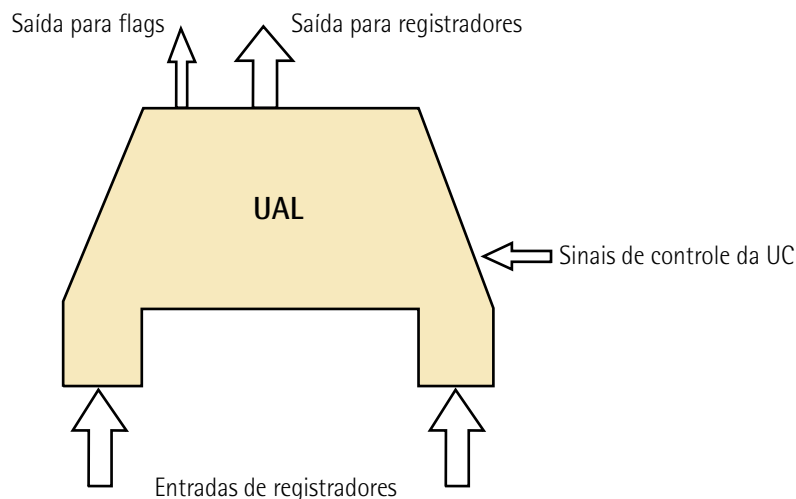


Figura 54 – Unidade Lógica Aritmética e suas conexões com a CPU

Fonte: Monteiro (2019, p. 164).

Alguns fabricantes de processadores atuais, como a Intel, têm substituído o nome ULA por Unidade de Cálculo ou Unidade de Execução, já outros, como a AMD, têm chamado a ULA de UI, Integer Unit (IU). Alguns processadores atuais também são constituídos por unidades responsáveis por cálculos fracionários, representados em ponto flutuante, denominadas de Floating Point Unit (FPU).

## 3.7.3 Unidade de Controle (UC)

A UC é considerada o dispositivo mais complexo da CPU. Ele é responsável pela movimentação de dados e instruções no processador através de sinais de controle sincronizados pelo relógio (clock). A unidade de controle opera sobre microoperações (pequenos passos no ciclo de leitura) onde a cada início de um ciclo de instrução ocorrerá uma busca (fetch) da instrução solicitada, trazendo uma cópia dela para o processador, que será armazenada no Instruction Register (IR). Cada microoperação é inicializada por um pulso oriundo da UC em decorrência de uma programação prévia, realizada diretamente no hardware. A estrutura básica diagramada de funcionamento da unidade de controle é observada na figura 55.

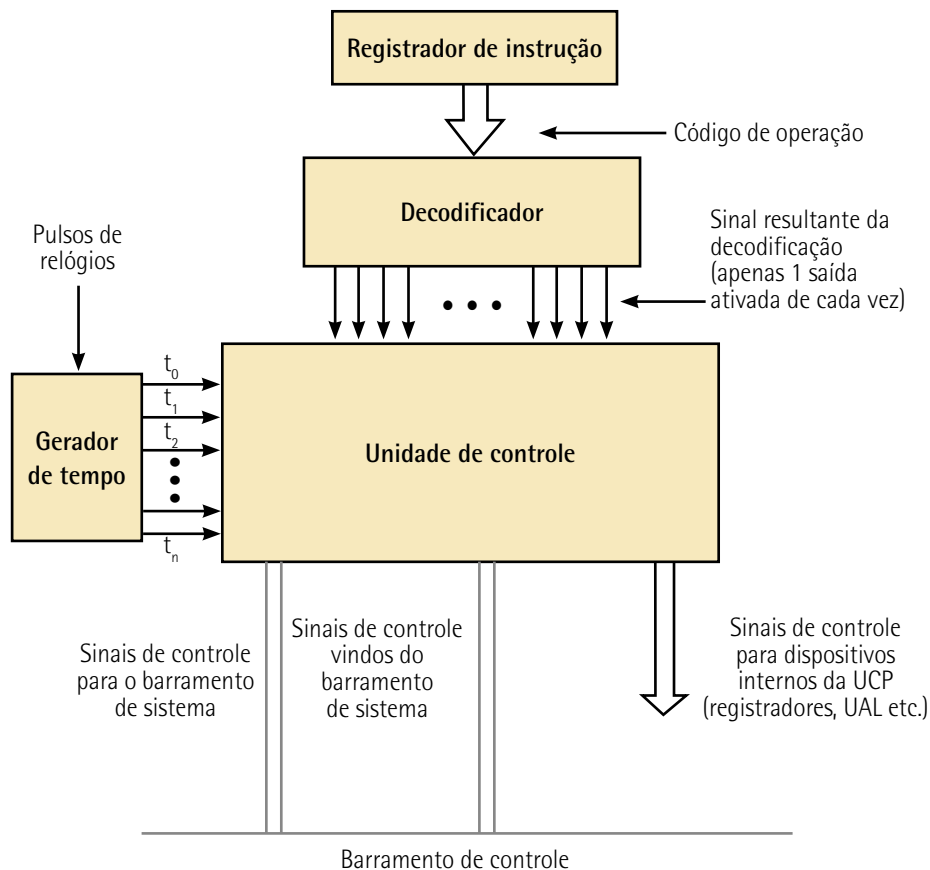


Figura 55 – Diagrama de bloco da função de controle

Fonte: Monteiro (2019, p. 172).

Os opcodes oriundos do registrador IR, primeiramente, são decodificados, ou seja, as instruções são interpretadas para servirem de sinais de execução na UC. Na figura 55, podemos ver um gerador de tempo que está conectado à UC para ditar o ritmo (pulsos) em que as instruções devem ser inseridas no barramento para alguma ação a ser executada pela ULA. A unidade de controle possui também requisitos funcionais, sendo eles: definição dos elementos básicos do processador; descrição das microoperações que o processador executa; determinação das funções que a UC deve realizar para que as microoperações sejam efetuadas.

Além desses requisitos funcionais, a UC também desempenha duas tarefas básicas: de sequenciamento, onde as microoperações devem ser executadas em série, baseadas no programa que está sendo executado; e de execução, onde a UC deve garantir que cada microoperação seja executada.

Para que a UC realize a operação da CPU de forma funcional, ela deve ser constituída por sinais de controle de entrada, como:

- **Registrador de instrução:** utilizado para definir qual opcode deverá ser executado no ciclo de instrução corrente.
- **Flags:** são necessárias para que a UC determine o estado do processador e das saídas das operações anteriores da ULA.
- **Sinais de controle do barramento de controle:** faz parte do barramento do sistema para o fornecimento de sinais para a UC.
- **Sinais de controle dentro do processador:** são sinais que fazem os dados serem movidos de um registrador para outro, além de ativarem funções específicas da ULA.
- **Sinais de controle para barramento de controle:** são constituídos pelos sinais de controle para memória principal e sinais de controle para os módulos de E/S.
- **Clock:** é responsável pelo tempo de ciclo do processador ou tempo de ciclo de clock de operação do processador.

### 3.7.4 Desempenho de operação do processador

Os fabricantes de chips, como a Intel ou AMD, por exemplo, buscam incessantemente o aumento no desempenho em seus processadores, mesmo que a evolução dos processadores esteja atrelada à lei de Moore. Por existirem imitações a fim de aumentar a velocidade do chip, atualmente três técnicas básicas são utilizadas para contornar esse problema (Stallings, 2010):

- A primeira implica aumentar a velocidade de hardware do processador com base na diminuição de tamanho das portas lógicas contidas no processador. Assim, quanto mais portas lógicas reunidas dentro de um mesmo chip, maior será a taxa de clock.
- A segunda implica aumentar a capacidade e velocidade das memórias auxiliares do tipo cache, inseridas entre o processador e a memória principal.
- A terceira técnica implica a realização de mudanças na arquitetura e organização de um processador, de modo a buscar o aumento na velocidade da execução das instruções, o que pode envolver algum processo de paralelismo na execução de tarefas.

Entre os fatores citados, o dominante para que haja ganho no desempenho tem sido o aumento na densidade de transistores no processador, aumentando assim a capacidade de clock. A figura 56 mostra essa tendência nos processadores Intel onde, na medida em que a densidade de transistores aumenta, ocorrem problemas na dissipação de energia além de uma estabilização na capacidade de clock que não evolui conjuntamente com a densidade.

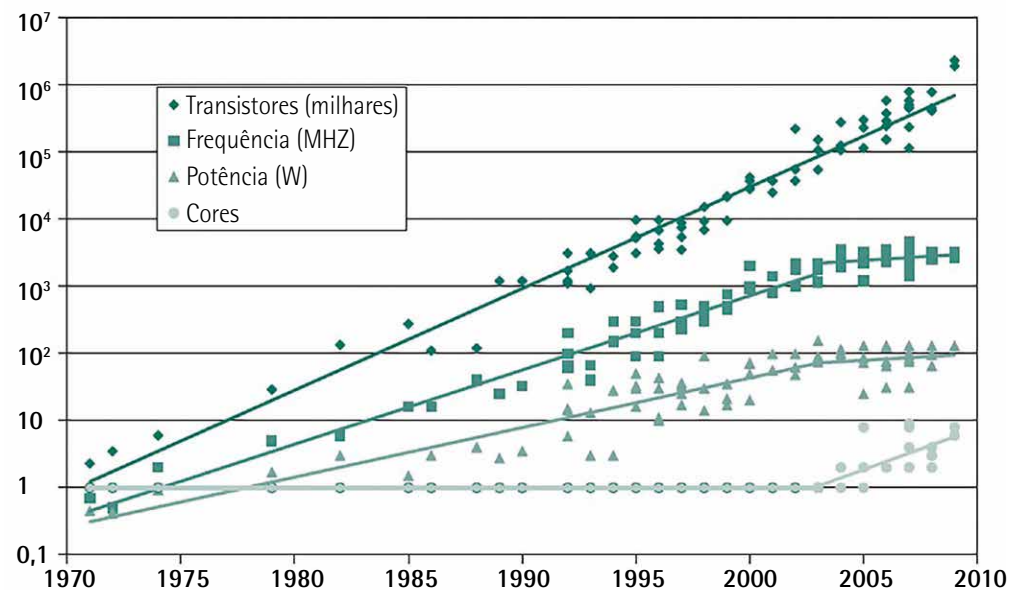


Figura 56 – Tendência de desempenho dos processadores

Fonte: Stallings (2010, p. 42).

Os problemas que podem ser apontados como limitadores no aumento da velocidade de clock e, por consequência, no desempenho do processador também estão relacionados à:

- **Potência:** aumenta a densidade de transistores e, conseqüentemente, a velocidade de clock do processador, aumentando também a densidade de potência dado em Watts/cm<sup>2</sup>. Isso dificultará a dissipação do calor gerado no chip, representando um sério problema a ser considerado no projeto.
- **Atraso de RC (resistência e capacitância):** existe uma limitação física na velocidade em que os elétrons fluem no chip entre os transistores, que é dada pela resistência e capacitância dos fios metálicos que conectam os transistores. Assim, o atraso pode aumentar na medida em que o produto RC aumenta. Conseqüentemente, com a diminuição dos componentes do chip, as interconexões de fios metálicos se tornam mais finas, o que aumenta a resistência elétrica. Ao diminuir o tamanho e a distância entre os transistores, estes também ficam mais próximos uns dos outros, aumentando a capacitância.
- **Latência de memória:** a velocidade de memória baixa limita a velocidade e o desempenho do processador.



### Saiba mais

Para conhecer mais sobre os impactos da Lei de Moore, recomendamos o artigo a seguir:

ROTMAN, D. We're not prepared for the end of Moore's Law. *MIT Technology Review*, 24 fev. 2020. Disponível em: <https://tinyurl.com/4e2pu2zd>. Acesso em: 12 dez. 2024.

## 4 SISTEMAS OPERACIONAIS

Um sistema operacional é um software essencial que gerencia o hardware de um computador, oferecendo uma base para a execução de programas e atuando como intermediário entre o usuário e o hardware. Uma característica interessante dos sistemas operacionais é a variedade de abordagens adotadas para cumprir essas funções.

Sistemas operacionais de mainframes são projetados para otimizar ao máximo o uso dos recursos de hardware. Em contrapartida, sistemas operacionais para computadores pessoais (PCs) são voltados para suportar desde jogos complexos até aplicações comerciais robustas. Já os SO para dispositivos móveis são desenvolvidos para garantir uma interface intuitiva, permitindo ao usuário interagir facilmente com o sistema e executar aplicativos com praticidade.

Dessa forma, alguns sistemas operacionais priorizam conveniência, outros, eficiência, e há aqueles que equilibram ambos os aspectos. Para entender melhor o funcionamento de um sistema operacional, é importante conhecer a estrutura básica de um sistema computacional. Isso inclui processos fundamentais, como inicialização do sistema (boot), operações de entrada e saída (I/O) e gerenciamento de armazenamento.

Devido à sua complexidade, os sistemas operacionais são desenvolvidos de forma modular, com cada módulo desempenhando funções específicas e bem definidas. Neste tópico, apresentaremos uma visão geral dos principais componentes de um sistema computacional moderno, as funções desempenhadas pelo sistema operacional, além de tópicos como as estruturas de dados utilizadas, os diferentes ambientes de computação e a relevância dos sistemas operacionais de código aberto.

### O que os sistemas operacionais fazem?

Iniciamos nossa discussão examinando o papel do sistema operacional no sistema de computação como um todo. Um sistema de computação pode ser grosseiramente dividido em quatro componentes: o hardware, o sistema operacional, os programas aplicativos e os usuários (figura 57).



Figura 57 – Visão abstrata dos componentes de um sistema de computação, imagem gerada em IA Image Generator

O hardware (a unidade central de processamento ou CPU), a memória e os dispositivos de entrada/saída (I/O) fornecem os recursos básicos de computação do sistema.

Os programas aplicativos, como processadores de texto, planilhas, compiladores e navegadores da web, definem as formas pelas quais esses recursos são utilizados para resolver os problemas computacionais dos usuários. O sistema operacional controla o hardware e coordena seu uso através dos diversos programas aplicativos de vários usuários.

Um sistema de computação é composto de hardware, software e dados, assim, o sistema operacional fornece os meios para a utilização apropriada desses recursos durante a operação do sistema de computação.

A fim de exemplificação, podemos pensar que um sistema operacional é semelhante ao governo. Tal como o governo, ele não desempenha funções úteis para si mesmo, proporcionando, simplesmente, um **ambiente** no qual outros programas possam desempenhar tarefas úteis.

Para entender melhor o papel do sistema operacional, examinaremos, a seguir, os sistemas operacionais a partir de dois pontos de vista: do usuário e do sistema.

## O sistema operacional visto pelo usuário

A perspectiva do usuário em relação ao computador varia, a depender da interface que estiver sendo utilizada. A maioria dos usuários de computador senta-se diante de um PC, constituído de um monitor, um teclado, um mouse e a unidade do sistema, onde tal sistema é projetado para que um único usuário



monopolize seus recursos. O objetivo é maximizar o trabalho (ou jogo) que o usuário estiver executando. Nesse caso, o sistema operacional é projetado principalmente para facilitar o uso, com alguma atenção dada ao desempenho e nenhuma à utilização dos recursos (a forma como vários recursos de hardware e software são compartilhados). É claro que o desempenho é importante para o usuário, mas esses sistemas são otimizados para a experiência de um único usuário, e não para atender a vários.

Ainda, em outros casos, o usuário senta-se diante de um terminal conectado a um mainframe ou a um minicomputador. Assim, tais usuários acessam o mesmo computador por intermédio de outros terminais, podendo compartilhar recursos e trocar informações. Em tais casos, o sistema operacional é projetado para maximizar a utilização dos recursos, assegurando que todo o tempo de CPU, memória e I/O disponíveis seja utilizado eficientemente e que nenhum usuário individual ocupe mais do que sua cota.

Há ainda outros casos em que os usuários ocupam estações de trabalho conectadas a redes de outras estações e servidores. Esses usuários possuem recursos dedicados à sua disposição, mas também compartilham recursos, tais como a rede e os servidores, incluindo servidores de arquivo, de computação e de impressão. Logo, seu sistema operacional é projetado para estabelecer um compromisso entre usabilidade individual e utilização dos recursos. Ultimamente, uma variedade de computadores móveis, como smartphones e tablets, passou a ser utilizada, onde a maioria é constituída de unidades autônomas para usuários individuais. Quase sempre, esses computadores moveis são conectados a redes por celulares ou outras tecnologias sem fio, substituindo, cada vez mais, os computadores, desktop e laptop, principalmente entre as pessoas que estão interessadas em usar tais dispositivos para enviar e-mails e navegar na web.

A interface de usuário dos computadores móveis, geralmente, tem uma tela sensível ao toque, fornecendo ao usuário uma interação com o sistema através de um simples toque na tela, não sendo necessário o uso de um teclado ou mouse físico. Ainda, alguns computadores são pouco ou nada visíveis ao usuário, como computadores embutidos em dispositivos domésticos e em automóveis que até podem ter um mostrador numérico, acendendo ou apagando luzes indicativas para mostrar um estado, mas, basicamente, seus sistemas operacionais são projetados para operar sem intervenção do usuário.

### **Do ponto de vista do sistema**

Do ponto de vista do computador, o sistema operacional é o programa mais intimamente envolvido com o hardware. Nesse contexto, podemos considerar um sistema operacional como um alocador de recursos. Uma vez que um sistema de computação tem muitos recursos que podem ser necessários à resolução de um problema (por exemplo: tempo de CPU, espaço de memória, espaço de armazenamento em arquivo, dispositivos de I/O etc.), o SO atua como o gerenciador deles. Ao lidar com solicitações de recursos numerosas e possivelmente conflitantes, o sistema operacional precisa decidir como alocá-los a programas e usuários específicos, de modo a poder operar o sistema de computação de maneira eficiente e justa.

Uma visão ligeiramente diferente de um sistema operacional enfatiza a necessidade de controle dos diversos dispositivos de I/O e programas de usuário. Assim, o SO, por ser um programa de controle, gerencia a execução de programas de usuário para evitar erros e o uso impróprio do computador, se preocupando principalmente com a operação e o controle de dispositivos de I/O.



## 4.1 Conceituação e tipos de sistemas operacionais

A essa altura, você já deve ter notado que o termo **sistema operacional** abrange muitos papéis e funções. Isso ocorre, em parte, devido aos inúmeros designs e usos dos computadores. Os computadores estão presentes dentro de torradeiras, carros, navios, espaçonaves, casas e empresas, mas, também, são a base das máquinas de jogos, reprodutores de música, sintonizadores de TV a cabo e sistemas de controle industrial.

Embora os computadores tenham uma história relativamente curta, eles evoluíram rapidamente. A computação começou como um experimento para determinar possibilidades e migrou rapidamente para sistemas de finalidade específica tanto para uso militar (em decifração de códigos e plotagem de trajetórias), quanto para o uso governamental (como no cálculo do censo). Esses computadores iniciais evoluíram para mainframes multifuncionais de uso geral, e foi então que nasceram os sistemas operacionais. Nos anos 1965, a Lei de Moore previu que o número de transistores de um circuito integrado dobraria a cada 18 meses, e essa previsão se confirmou. Os computadores aumentaram em funcionalidade e diminuíram em tamanho, levando a um vasto número de usos e variedades de sistemas operacionais.

Com essa vastidão de funções e características, como podemos definir o que é um sistema operacional? Em geral, não possuímos uma definição totalmente adequada de um sistema operacional, mas podemos dizer que uma das principais definições de um SO é fornecer uma solução para a dificuldade gerada no processo de criação de um sistema de computação utilizável.

O objetivo fundamental dos sistemas de computação é executar os programas dos usuários e facilitar a resolução dos seus problemas. É com esse objetivo que o hardware do computador é construído e os programas aplicativos são desenvolvidos, tendo em vista que o hardware puro e sozinho não é particularmente fácil de ser utilizado. Esses programas requerem determinadas operações comuns, como as que controlam os dispositivos de I/O. As funções comuns de controle e alocação de recursos são, então, reunidas em um único software: o sistema operacional.

Não há uma definição universalmente aceita sobre o que compõe o sistema operacional. Um ponto de vista simplista é o de que esse sistema inclui tudo que um fornecedor oferece quando você encomenda em SO. Entretanto, os recursos incluídos variam muito entre os sistemas, onde alguns sistemas ocupam menos do que um megabyte de espaço e não têm nem mesmo um editor de tela inteira, enquanto outros requerem gigabytes de espaço e são inteiramente baseados em sistemas de janelas gráficas. Uma definição mais comum, que é a que costumamos seguir, é a de que o sistema operacional é o único programa que permanece em execução no computador durante todo o tempo chamado, em geral, de kernel.



### Observação

Além do kernel, há dois outros tipos de programas: os programas do sistema que estão associados ao sistema operacional, mas não necessariamente fazem parte do kernel; e os programas aplicativos, que incluem todos os programas não associados à operação do sistema.



Para que um computador comece a operar, ele precisa ter um programa inicial para executar. Esse programa inicial, ou programa bootstrap, tende a ser simples, sendo, normalmente, armazenado dentro do hardware do computador em memória somente de leitura (ROM) ou em memória somente de leitura eletricamente apagável e programável, Electrically Erasable Programmable Read-Only Memory (EEPROM), conhecida pelo termo geral firmware. O bootstrap inicializa todos os aspectos do sistema, dos registradores da CPU aos controladores de dispositivos e conteúdo da memória, precisando saber como carregar o sistema operacional e iniciar sua execução. Para alcançar esse objetivo, o programa tem que localizar e carregar na memória o kernel do sistema operacional.

Assim que o kernel é carregado e está em execução, ele pode começar a fornecer serviços para o sistema e seus usuários. Alguns serviços são fornecidos fora do kernel por programas do sistema que são carregados na memória em tempo de inicialização, a fim de se tornarem processos do sistema, ou daemons do sistema, que são executados durante todo o tempo em que o kernel é executado. No UNIX, o primeiro processo do sistema é "init" e ele inicia muitos outros daemons. Quando essa fase é concluída, o sistema está totalmente inicializado e aguarda que algum evento ocorra.

Geralmente, a ocorrência de um evento é indicada por uma interrupção proveniente do hardware ou do software. O hardware pode disparar uma interrupção a qualquer momento enviando um sinal à CPU, normalmente pelo bus do sistema. O software pode disparar uma interrupção executando uma operação especial denominada de sistema (também chamada de monitor).



### **Lembrete**

O programa bootstrap também é conhecido como um framework front-end muito popular que simplifica o desenvolvimento de sites e aplicações web responsivos. Entretanto, são tecnologias totalmente diferentes.

Quando a CPU é interrompida, ela para o que está fazendo e transfere imediatamente a execução para uma localização fixa. Normalmente, essa localização fixa contém o endereço inicial no qual se encontra a rotina de serviço da interrupção. A rotina de serviço da interrupção entra em operação e, ao completar a execução, a CPU retoma a computação interrompida.

Uma linha do tempo dessa operação é mostrada na figura 59.

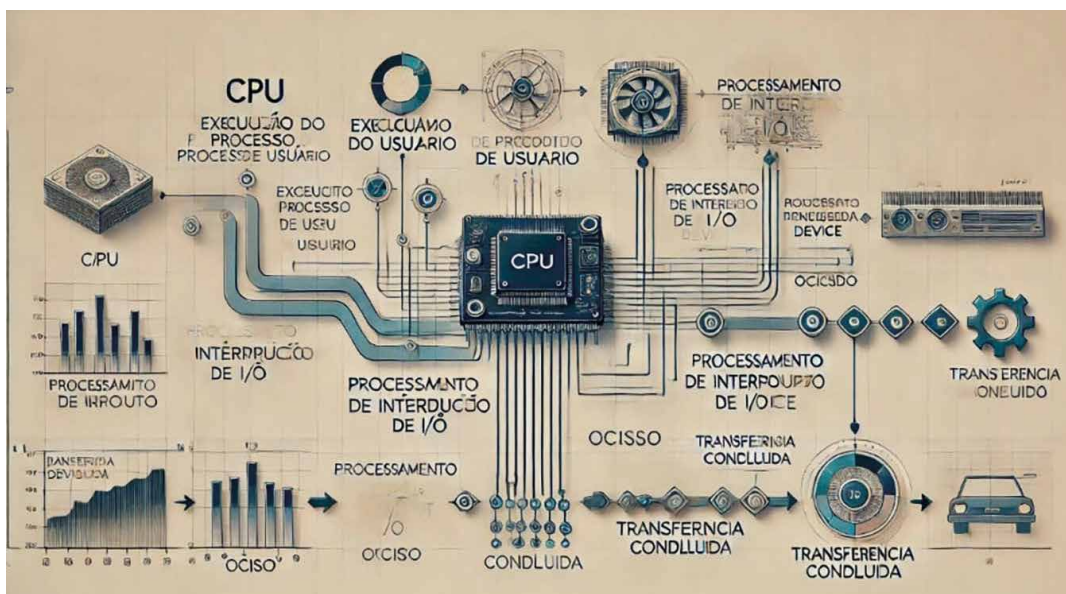


Figura 59 – Linha do tempo de interrupções para um processo individual que gera saídas, imagem gerada em IA Image Generator

As interrupções são uma parte importante da arquitetura do computador, pois cada projeto de computador tem seu próprio mecanismo de interrupção, mas diversas funções são comuns a todos. A interrupção deve transferir o controle para a rotina de serviço de interrupção apropriada; o método mais simples para a manipulação dessa transferência seria invocar uma rotina genérica que examinasse a informação de interrupção. A rotina, por sua vez, chamaria o manipulador de interrupções específico. Entretanto, as interrupções precisam ser manipuladas rapidamente, já que só é permitida uma quantidade predeterminada de interrupções, onde uma tabela de ponteiros para rotinas de interrupção pode ser usada como alternativa para fornecer a velocidade necessária.

A rotina de interrupção é chamada indiretamente pela tabela, sem necessidade de rotina intermediária. Geralmente, a tabela de ponteiros é armazenada na memória baixa (mais ou menos nas 100 primeiras localizações), onde essas localizações mantêm os endereços das rotinas de serviço de interrupção dos diversos dispositivos. Esse array de endereços, ou vetor de interrupções, é então indexado por um único número de dispositivo, fornecido com a solicitação de interrupção, de modo que seja obtido o endereço da rotina de serviço de interrupção do dispositivo que a está causando. Sistemas operacionais tão diferentes, como o Windows e o UNIX, despacham as interrupções dessa maneira.

A arquitetura de interrupções também deve salvar o endereço da instrução interrompida. Muitos projetos antigos simplesmente armazenavam o endereço interrompido em uma localização fixa ou em uma localização indexada pelo número do dispositivo, já arquiteturas mais recentes armazenam o endereço de retorno na pilha do sistema. Se a rotina de interrupção precisar modificar o estado do processador – por exemplo, modificando os valores dos registradores –, ela deve salvar explicitamente o estado corrente e, então, restaurar esse estado antes de retornar. Após a interrupção ser atendida, o endereço de retorno salvo é carregado no contador do programa e a computação interrompida é retomada como se a interrupção não tivesse ocorrido.

## 4.2 Componentes do sistema operacional

### Estrutura de armazenamento

A CPU só pode carregar instruções a partir da memória, portanto, para serem executados, os programas devem estar armazenados na memória. Computadores de uso geral executam a maioria de seus programas a partir de memória regravável, ou seja, na memória principal (também chamada de memória de acesso randômico ou RAM). Normalmente, a memória principal é implementada em uma tecnologia de semicondutor, denominada Dynamic Random-Access Memory (DRAM).

A unidade básica de armazenamento nos computadores é o bit. Um bit pode conter um entre dois valores, 0 e 1. Todos os outros tipos de armazenamento em um computador são baseados em conjuntos de bits. Quando há bits suficientes, é espantoso quantas coisas um computador pode representar: números, letras, imagens, filmes, sons, documentos e programas, para citar apenas alguns. Um byte é composto por 8 bits e, na maioria dos computadores, é o menor bloco de armazenamento conveniente. Por exemplo, a maioria dos computadores não tem uma instrução para mover um bit, e sim para mover um byte.

Um termo menos comum nessa área é **palavra** que é uma unidade de dados nativa de determinada arquitetura de computador. Uma palavra é composta por um ou mais bytes. Por exemplo, um computador que tem registradores de 64 bits e endereçamento de memória de 64 bits tem, normalmente, palavras de 64 bits (8 bytes). Um computador executa muitas operações em seu tamanho de palavra nativo, em vez de usar um byte de cada vez.

O espaço de armazenamento do computador, junto com a maior parte do seu throughput, é geralmente medido e manipulado em bytes e conjuntos de bytes, sendo eles:

- Um kilobyte, ou kB, é igual a 1024 bytes.
- Um megabyte, ou MB, equivale a 1024 kB.
- Um gigabyte, ou GB, é o mesmo que 1024 MB.
- Um terabyte, ou TB, corresponde a 1024 GB.
- Um pentabyte, ou PB, é composto por 1024 TB.

Os fabricantes de computador costumam arredondar esses números e dizem que um megabyte corresponde a 1 milhão de bytes e um gigabyte a 1 bilhão de bytes. As medidas aplicadas às redes são uma exceção a essa regra geral, sendo fornecidas em bits (porque as redes movem os dados bit a bit).



Os computadores também usam outros tipos de memória. Já mencionamos a memória somente de leitura (ROM) e a memória somente de leitura eletricamente apagável e programável (EEPROM). Como a ROM não pode ser alterada, somente programas estáticos podem ser armazenados nela, como o programa bootstrap, o que faz com que a imutabilidade da ROM seja útil em cartuchos de jogos. Quanto à EEPROM, ela pode ser alterada, mas não com frequência, ocasionando a prevalência de programas estáticos. Por exemplo, os smartphones têm EEPROM para armazenar seus programas instalados de fábrica.

Além disso, todos os tipos de memória fornecem um array de bytes e cada byte tem seu próprio endereço. A interação é alcançada por intermédio de uma sequência de instruções load ou store para endereços de memória específicos. A instrução load move um byte ou palavra da memória principal para um registrador interno da CPU, enquanto a instrução store move o conteúdo de um registrador para a memória principal. Além das cargas e armazenamentos explícitos, a CPU carrega automaticamente, para execução, as instruções a partir da memória principal.

Um típico ciclo instrução-execução, conforme realizado em um sistema com arquitetura von Neumann, traz, primeiro, uma instrução da memória e a armazena no registrador de instruções. A instrução é, então, decodificada e pode provocar a busca de operandos na memória e o seu armazenamento em algum registrador interno. Após a execução da instrução sobre os operandos, o resultado pode ser armazenado novamente na memória. Observe que a unidade de memória enxerga apenas um fluxo de endereços de memória. Ela não sabe como eles são gerados (pelo contador de instruções, por indexação, indiretamente, como endereços literais ou por algum outro meio) ou para que servem (instruções ou dados). Da mesma forma, podemos ignorar como um endereço de memória é gerado por um programa. Só estamos interessados na sequência de endereços de memória gerados pelo programa em execução.

Idealmente, gostaríamos que os programas e dados residissem na memória principal de modo permanente. Esse esquema, geralmente, não é possível porque a memória principal não apenas costuma ser muito pequena para armazenar permanentemente todos os programas e dados necessários, como também se enquadra como um dispositivo de armazenamento volátil que perde seus conteúdos quando a energia é desligada ou acaba por outro motivo.

Assim, a maioria dos sistemas de computação fornece memória secundária como uma extensão da memória principal. O principal requisito da memória secundária é que ela seja capaz de armazenar grandes quantidades de dados permanentemente.

O dispositivo de memória secundária mais comum é o disco magnético, que fornece armazenamento tanto para programas quanto para dados. A maioria dos programas (de sistema e de aplicação) é armazenada em um disco até que seja carregada na memória. Por isso, muitos programas utilizam o disco tanto como fonte quanto como destino de seu processamento. Logo, o gerenciamento apropriado do armazenamento em disco é de suma importância para um sistema de computação.

De modo geral, no entanto, a estrutura de armazenamento que descrevemos, constituída de registradores, memória principal e discos magnéticos, é apenas um dos muitos sistemas de armazenamento possíveis. Há ainda a memória cache, o CD-ROM, as fitas magnéticas etc. Cada sistema

de armazenamento fornece as funções básicas de armazenamento e de manutenção de dados até que sejam recuperados mais tarde. As principais diferenças entre os vários sistemas de armazenamento residem na velocidade, no custo, no tamanho e na volatilidade.

A ampla variedade de sistemas de armazenamento pode ser organizada em uma hierarquia (figura 60), de acordo com a velocidade e o custo. Os níveis mais altos são caros, mas, também, rápidos. À medida que descemos na hierarquia, o custo por bit geralmente decresce, enquanto o tempo de acesso, em geral, aumenta. Essa desvantagem é razoável; se determinado sistema de armazenamento fosse, ao mesmo tempo, mais rápido e menos caro do que outro, tendo as demais propriedades idênticas, então não haveria razão para utilizar a memória mais lenta e mais dispendiosa. Na verdade, muitos dispositivos de armazenamento antigos, inclusive fita de papel e memórias de núcleo, foram relegados aos museus depois que a fita magnética e a memória semicondutora tornaram-se mais rápidas e mais baratas. Os quatro níveis mais altos de memória, como vistos na figura 60, podem ser construídos com o uso de memória semicondutora.

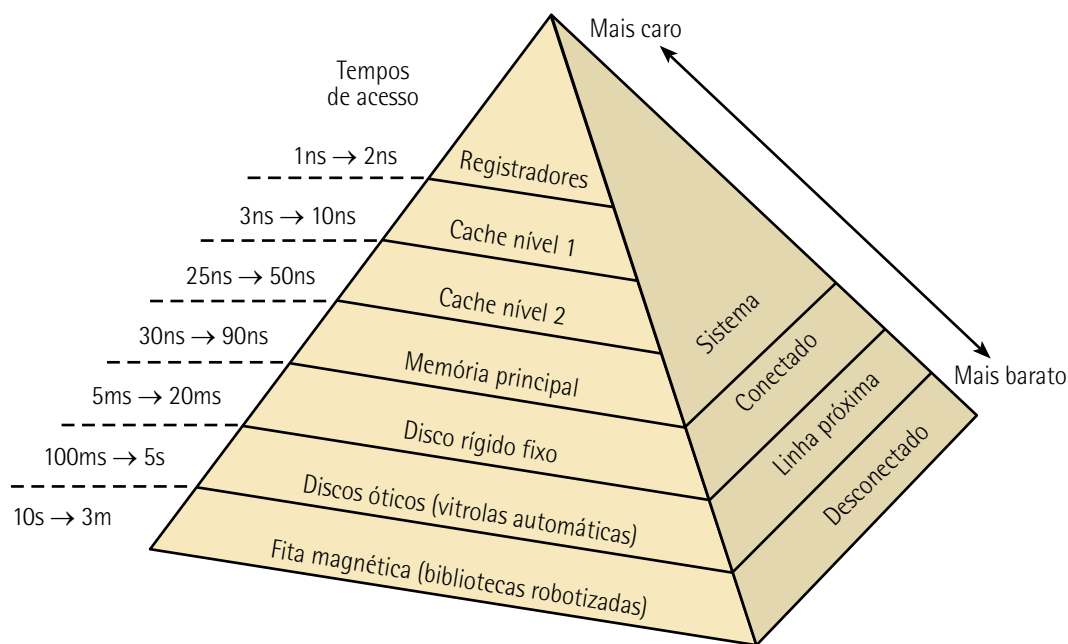


Figura 60 – Hierarquia dos dispositivos de armazenamento

Fonte: Null e Lobur (2010, p. 313).

Além de diferirem na velocidade e no custo, os diversos sistemas de armazenamento podem ser voláteis ou não voláteis. Como mencionado anteriormente, a memória volátil perde seus conteúdos quando a energia para o dispositivo é removida. Na ausência de dispendiosos sistemas de backup por bateria ou gerador, os dados devem ser gravados em uma memória não volátil por segurança. Na hierarquia mostrada na figura 60, os sistemas de armazenamento situados acima do disco de estado sólido são voláteis, enquanto os que o incluem e estão abaixo dele são não voláteis.

Há muitos tipos de discos de estado sólido, mas em geral eles são mais rápidos do que os discos magnéticos e são não voláteis. Um tipo de disco de estado sólido armazena dados em um extenso array DRAM durante a operação normal, mas também contém um disco rígido magnético oculto e uma bateria como energia de backup. Quando há interrupção da energia externa, o controlador do disco de estado sólido copia os dados da RAM no disco magnético. Quando a energia externa é restaurada, o controlador copia os dados novamente na RAM.

Outro tipo de disco de estado sólido é a memória flash, que é popular em câmeras e assistentes digitais pessoais – Personal Digital Assistants (PDAs) –, em robôs e, cada vez mais, em armazenamento de computadores de uso geral. A memória flash é mais lenta do que a DRAM, mas não precisa de energia para reter seus conteúdos. Além desses tipos de disco, há também a NVRAM, que é a DRAM com energia de backup por bateria, podendo ser tão rápida quanto a DRAM e, enquanto a bateria durar, permanece não volátil.

O projeto de um sistema de memória completo deve balancear todos os fatores que acabamos de discutir, devendo usar uma memória cara apenas quando necessário e fornecer o máximo possível de memória barata e não volátil. Caches podem ser instalados para melhorar o desempenho quando existir grande disparidade no tempo de acesso ou na taxa de transferência entre dois componentes. A relação de descendência na hierarquia de memória segue os seguintes critérios:

- Diminuição do custo por bit, quanto menor for o nível na hierarquia.
- Aumento na capacidade de armazenamento, quanto menor for o nível na hierarquia.
- Aumento no tempo para acesso à memória, quanto menor for o nível na hierarquia.
- Aumento ou diminuição da frequência de acesso às memórias, de acordo com o nível de hierarquia.

Assim como vimos na figura 60, no topo da hierarquia de memória estão os registradores do processador (MBR, IR, MAR, PC, AC etc.), que, embora mais rápidos (por volta de 1 a 2 nanosegundos), são os mais caros e possuem menor capacidade de armazenamento. Memórias caches L1, L2 e L3, internas ao processador, possuem tempo de acesso estimados de 3 a 50 nanosegundos, respectivamente. A memória principal do computador (RAM) possui um tempo de acesso médio de 90 nanosegundos e capacidade de armazenamento que pode chegar a 32 gigabytes em um único pente de memória. Descendo mais no nível de hierarquia, os discos rígidos magnéticos, muito utilizados em computadores pessoais, possuem capacidade de armazenamento mediano (1 a 10 terabytes) a um custo de tempo de acesso na faixa dos milissegundos. Discos ópticos e fitas magnéticas, geralmente utilizados para realizar back-up em empresas, estão na base da hierarquia de memória e possuem tempo de acesso que pode levar segundos ou mesmo minutos, com a grande vantagem de serem as memórias com menor custo financeiro atualmente.



## 4.3 Memória cache

A memória cache, em um sistema computacional, tem como objetivo obter velocidade de memória similar à das memórias mais rápidas como os registradores, porém, disponibilizando uma maior capacidade (alguns megabytes) em comparação aos registradores.



### Observação

O computador tem como tendência, durante sua execução, referenciar dados/instruções localizados na memória principal, de forma que essa memória intermediária tenta acessar esses endereços de forma mais ágil e, ao mesmo tempo, armazenar parte desses dados ou endereços para possíveis acessos futuros.

O conjunto de memórias do computador consiste basicamente em uma memória principal (RAM) de maior capacidade, porém mais lenta, em conjunto com a memória cache, de menor capacidade, porém mais rápida, como esquematizado na figura 61.

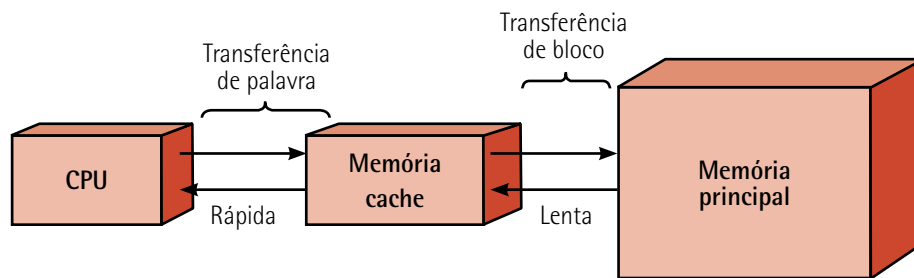


Figura 61 – Relação de comunicação entre a CPU, cache e memória principal

Fonte: Stallings (2010, p. 96).

À medida em que a CPU necessita realizar a leitura de uma palavra contida na memória, realiza-se a verificação para determinar se a palavra está contida na memória cache. Se estiver contida na cache, a palavra será entregue à CPU, caso não esteja, um bloco da memória principal, contendo algum número fixo de palavras, será lido na cache e o resultado será fornecido à CPU.

Essas conexões são conhecidas como localidade de referência e ocorrem quando um bloco de dados é enviado para a memória cache a fim de realizar uma única referência à memória. As memórias cache ainda podem ser subdivididas em diferentes níveis (multiníveis) ou layers (L1, L2 e L3), onde a cache L3 possui maior capacidade de armazenamento entre todas as caches, porém é a mais lenta. Assim, a cache L2 é mais lenta do que a L1 e, por consequência, a L1 é a mais rápida nessa hierarquia.

A figura 62 mostra uma relação de comunicação entre os diferentes níveis de cache, porém, nesse exemplo, todas as caches estão externas à CPU, fato que não ocorre com as memórias cache atuais.

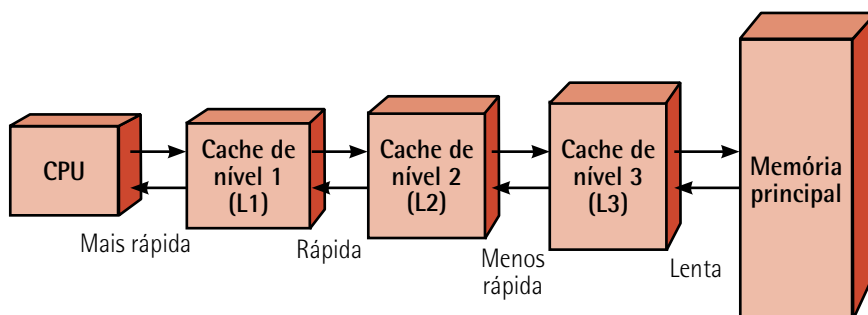


Figura 62 – Organização de memória cache em três níveis

Fonte: Stallings (2010, p. 96).

De forma mais específica, a memória cache possui uma matriz de etiqueta e uma matriz de dados, como mostrado na figura 63. A matriz de etiquetas possui os endereços de todos os dados contidos na memória cache e a matriz de dados contém, conseqüentemente, todos os dados.

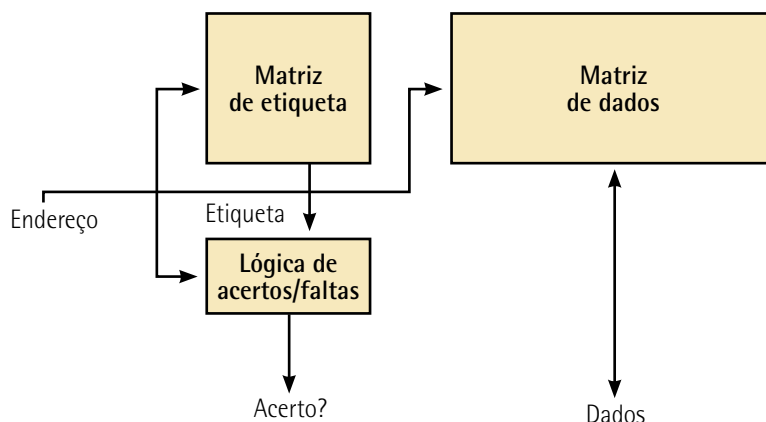


Figura 63 – Diagrama de blocos da memória cache

Fonte: Carter (2002, p. 159).

A proposta da divisão da memória cache em matrizes tem como objetivo reduzir seu tempo de acesso, pois a matriz de etiquetas geralmente possui menos bits se comparada à matriz de dados e, por consequência, poderá ser acessada de forma mais rápida. Quando a matriz de dados é acessada, a saída obtida precisa ser comparada com o endereço da memória de referência para que seja determinado se houve ou não um acerto de memória cache, também conhecido como cache hit. O acerto de cache significa que os buffers de dados e endereços são desativados e haverá uma comunicação apenas entre a CPU e a memória cache, sem algum outro tráfego no barramento interno do sistema. O processo inverso ocorre em caso de falha de cache ou cache miss, quando o endereço desejado é colocado no barramento e os dados são deslocados pelo buffer de dados para a memória cache e a CPU.

## 4.4 Cache de dados e instruções

Em outros tipos de memória como a RAM, as instruções e os dados geralmente compartilham o mesmo espaço dentro de cada nível da hierarquia de memória; entretanto, o mesmo não ocorre na memória cache, em que os dados e instruções são armazenados em caches diferentes. Essa característica ficou conhecida como cache Harvard (figura 64), pois se trata de uma das características da arquitetura de Harvard.

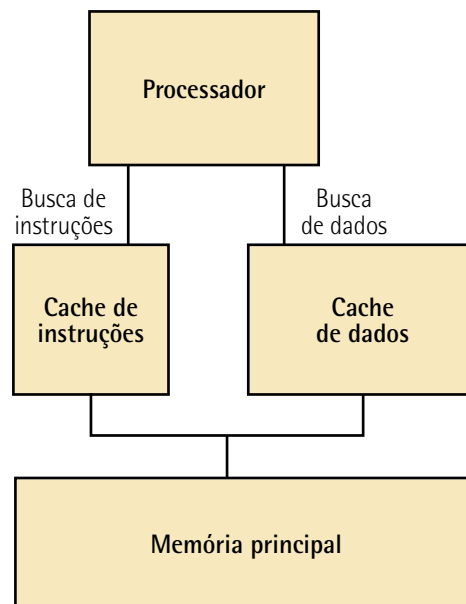


Figura 64 – Esquema de interligação de cache para processadores de arquitetura Harvard

Fonte: Carter (2002, p. 159).

Essa configuração é utilizada por permitir que a CPU busque de forma simultânea as instruções a partir da cache de instruções e dados pela cache de dados. Assim, quando a memória cache possui tanto instruções quanto dados, ela é denominada de cache unificada. Uma desvantagem no uso de cache separadas é que a automodificação de certos programas se torna difícil, ou seja, quando um programa precisa modificar suas próprias instruções, tais instruções serão tratadas como dados e armazenadas na cache de dados. Assim, para executar as instruções modificadas, o sistema operacional precisará realizar operações especiais de descarga da cache. De forma geral, a memória cache de instruções pode ser de duas a quatro vezes menor do que a cache de dados, pois as instruções de um sistema operacional ocupam menor espaço físico na memória se comparado com os dados.

### 4.4.1 Endereço de cache

Um conceito intimamente relacionado com a memória cache diz respeito à memória virtual, nome dado à técnica que utiliza uma memória secundária como o disco rígido ou mesmo a memória cache para o armazenamento temporário. Uma memória virtual, basicamente, permite que programas façam o endereçamento da memória de forma lógica, sem de fato considerar a quantidade de memória disponível fisicamente na RAM. Quando a memória virtual é utilizada, os locais de endereço nas

instruções conterão endereços virtuais, e não físicos, de forma que, para realizar a leitura/escrita da memória RAM, será necessário o uso de uma unidade gerenciadora de memória, Memory Management Unit (MMU), para traduzir os endereços virtuais em endereços físicos na RAM, como mostra a figura 65.

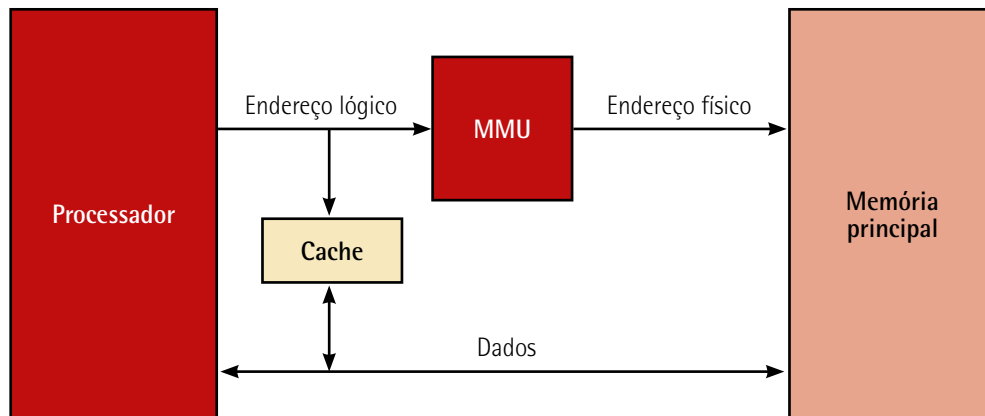


Figura 65 – Organização de memória cache lógica

Fonte: Stallings (2010, p. 99).

Uma cache lógica ou virtual pode armazenar os dados utilizando endereços também virtuais, assim o processador acessará a memória cache de forma direta, sem a necessidade de passar pela MMU, como mostra a organização ilustrada na figura 66.

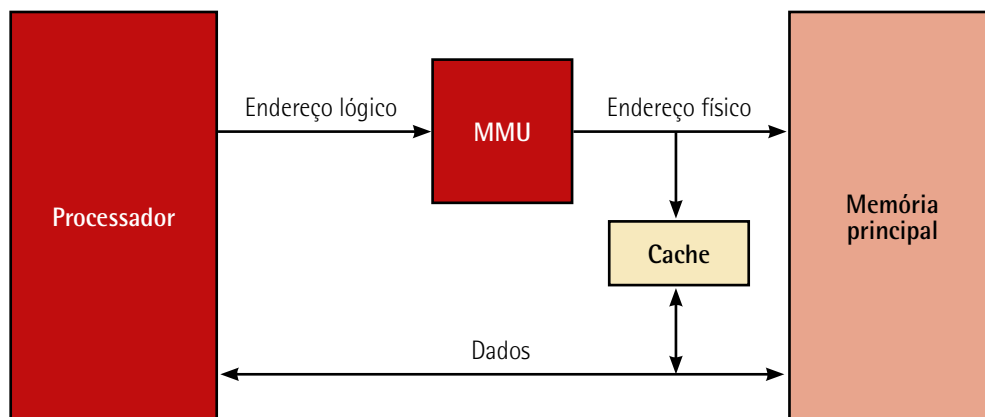


Figura 66 – Organização de memória cache física

Fonte: Stallings (2010, p. 99).

Uma das vantagens nesse tipo de abordagem da cache lógica é que a velocidade de acesso será maior do que para uma cache do tipo física, pois, como os diagramas mostram, a cache pode responder antes que a MMU concretize a tradução do endereçamento.

## 4.4.2 Caches associativas

A associatividade em memória cache se refere a quantas posições contém um endereço de memória nela. As caches associativas permitem que os endereços sejam armazenados em diversas posições na cache, o que reduzirá as penalidades ocasionadas por conflitos no barramento de memória, devido a dados que precisem ser armazenados nas mesmas posições. Memórias cache que possuam uma baixa associatividade podem restringir o número de posições de endereços disponíveis, o que pode ocasionar o aumento nas falhas de cache e que, por consequência, reduzirá o espaço ocupado. As caches podem ter um mapeamento associativo, como mostrado na figura 67.

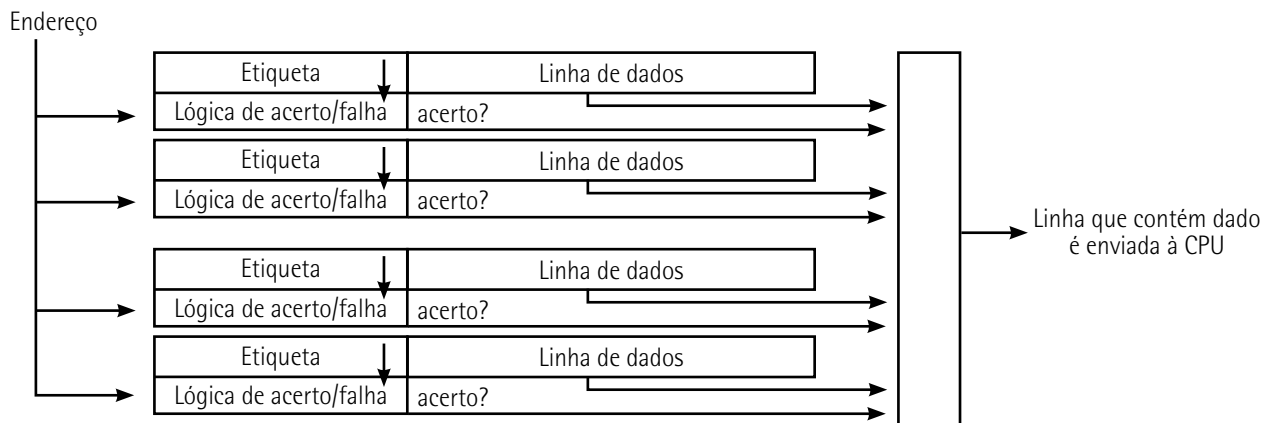


Figura 67 – Esquema de memória cache com mapeamento associativo

Fonte: Carter (2002, p. 162).

Nesse tipo de endereçamento de cache, é permitido que qualquer endereço seja armazenado em qualquer uma das linhas de cache. Assim, quando uma operação de acesso à memória é solicitada à cache, tal acesso precisa ser comparado a cada uma das entradas na matriz de etiquetas para que seja determinado se os dados referenciados na operação de fato estarão contidos nela. As caches associativas são, no geral, implementadas com matrizes distintas para dados e etiquetas.

## 4.4.3 Caches com mapeamento direto

As caches com mapeamento direto são exatamente o oposto das associativas. No mapeamento direto, cada endereço só poderá ser armazenado em uma posição da memória cache, como exemplificado na figura 68. Dessa forma, quando uma operação de acesso à memória é enviada a uma cache que foi mapeada de forma direta, um subconjunto de bits será utilizado para selecionar o byte que está dentro de uma linha da cache para onde aponta o endereço.

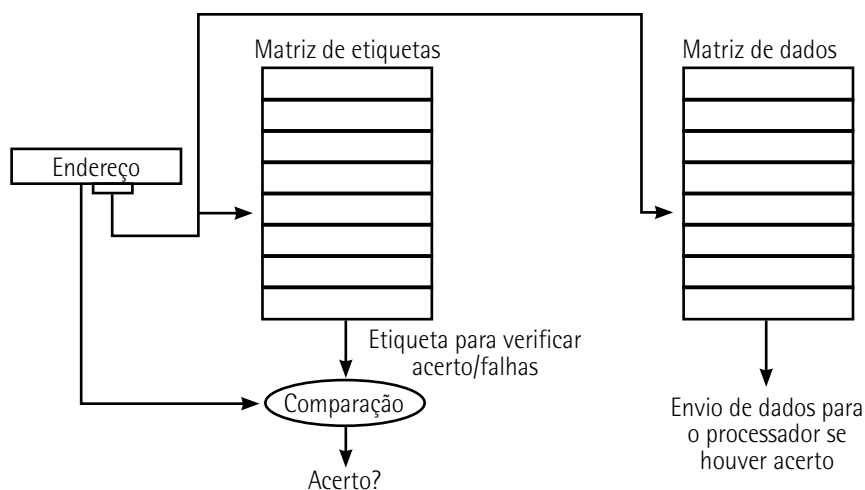


Figura 68 – Esquema de memória cache com mapeamento direto

Fonte: Carter (2002, p. 163).

De forma geral, haverá  $n$  bits menos significativos no endereçamento que serão utilizados para determinar a posição do endereço de dentro da linha de cache, onde  $n$  é dado por  $\log$  na base 2 do número de bytes contidos na linha. Assim, os  $m$  bits mais significativos, dados também por  $\log$  na base 2 do número de linhas na cache, serão utilizados para selecionar a linha que conterá o endereço armazenado.

### Estrutura de I/O

O armazenamento é apenas um dos muitos tipos de dispositivos de I/O de um computador. Grande parte do código do sistema operacional é dedicada ao gerenciamento de I/O, tanto devido a sua importância para a confiabilidade e o desempenho de um sistema quanto em razão da natureza variada dos dispositivos. A seguir, fornecemos uma visão geral do I/O.

Um sistema de computação de uso geral é composto por CPUs e vários controladores de dispositivos conectados por um bus comum. Cada controlador de dispositivos é responsável por um tipo específico de dispositivo; dependendo do controlador, pode haver mais de um conectado. Por exemplo, sete ou mais dispositivos podem ser conectados ao controlador da interface de pequenos sistemas de computação Small Computer-Systems Interface (SCSI). Um controlador de dispositivos mantém algum armazenamento em buffer local e um conjunto de registradores de uso específico. Assim, o controlador é responsável por movimentar os dados entre os dispositivos periféricos que controla e seu buffer de armazenamento local. Normalmente, os sistemas operacionais têm um driver de dispositivo para cada controlador de dispositivos. Esse driver compreende o controlador de dispositivos e fornece uma interface uniforme entre o dispositivo e o resto do sistema operacional.

Para iniciar uma operação de I/O, o driver do dispositivo carrega os registradores apropriados dentro do controlador do dispositivo. Este, por sua vez, examina o conteúdo dos registradores para determinar que ação tomar (tal como "ler um caractere a partir do teclado"). O controlador inicia a transferência de dados do dispositivo para o seu buffer local.

Quando a transferência de dados é concluída, o controlador do dispositivo informa ao driver do dispositivo, por uma interrupção, que terminou sua operação. O driver do dispositivo retorna então o controle para o sistema operacional, possivelmente retornando os dados ou um ponteiro para os dados se a operação foi uma leitura. Para outras operações, o driver do dispositivo retorna informações de status.

Esse tipo de I/O dirigido por interrupção é adequado para a movimentação de pequenas quantidades de dados, mas pode produzir um overhead alto quando usado na movimentação de dados de massa como no I/O de disco. Para resolver esse problema, é utilizado o acesso direto à memória (DMA). Após estabelecer os buffers, ponteiros e contadores para o dispositivo de I/O, o controlador do dispositivo transfere um bloco inteiro de dados diretamente da memória para o seu próprio buffer ou a partir dele para a memória, sem intervenção da CPU. É gerada somente uma interrupção por bloco para informar ao driver do dispositivo que a operação foi concluída, em vez de uma interrupção por byte gerada para dispositivos de baixa velocidade. Enquanto o controlador do dispositivo está executando essas operações, a CPU está disponível para cumprir outras tarefas.

Alguns sistemas de topo de linha usam arquitetura baseada em switch, e não em bus. Nesses sistemas, vários componentes podem conversar com outros componentes concorrentemente, em vez de competirem por ciclos em um bus compartilhado. Nesse caso, o DMA é ainda mais eficaz. A figura 69 mostra a interação de todos os componentes de um sistema de computação.

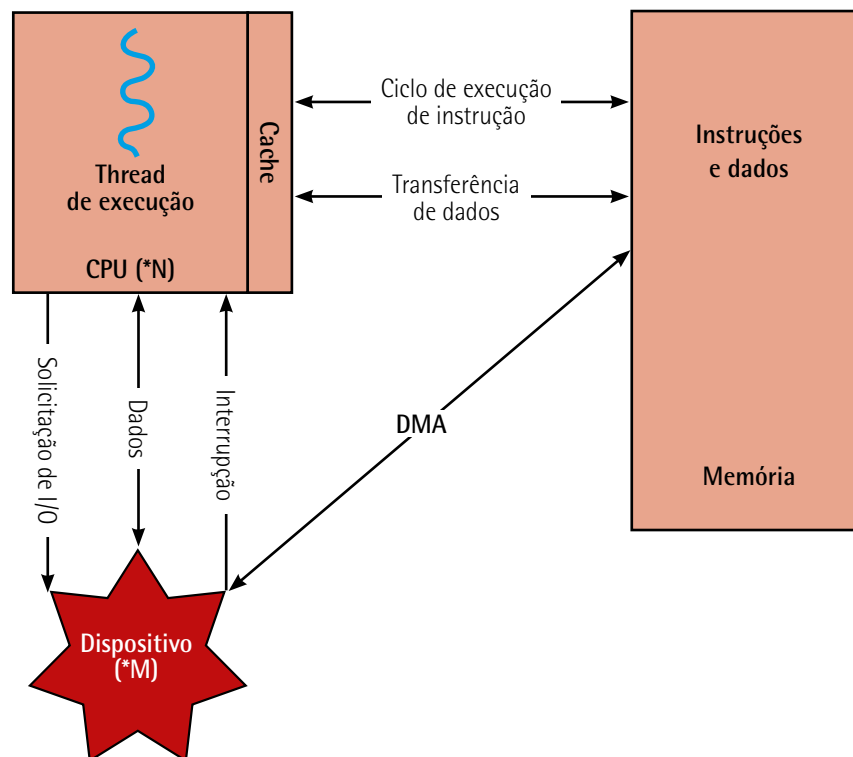


Figura 69 – Como um moderno sistema de computação funciona

Fonte: Silberschartz, Galvin e Gagne (2015, p. 8).

### 4.5 Evolução dos sistemas operacionais

#### 4.5.1 Sistemas uniprocessadores

Até recentemente, a maioria dos sistemas de computação usava um único processador. Em um sistema de processador único, há uma CPU principal capaz de executar um conjunto de instruções de uso geral, incluindo instruções provenientes de processos de usuário. Quase todos os sistemas de processador único também têm outros processadores de uso específico. Eles podem estar na forma de processadores de dispositivos específicos, como controladores de disco, teclado e monitor, ou, ainda, nos mainframes, na forma de processadores de uso mais genérico, como os processadores de I/O que movem dados rapidamente entre os componentes do sistema.

Todos esses processadores de uso específico executam um conjunto limitado de instruções e não executam processos de usuário, sendo, às vezes, gerenciados pelo sistema operacional, caso em que o sistema operacional lhes envia informações sobre sua próxima tarefa e monitora seu status. Por exemplo, o microprocessador de um controlador de disco recebe uma sequência de solicitações da CPU principal e implementa sua própria fila no disco e o algoritmo de scheduling do disco. Esse esquema libera a CPU principal do overhead do scheduling de disco. Os PCs contêm um microprocessador no teclado para converter os toques de teclas em códigos a serem enviados para a CPU. Em outros sistemas ou circunstâncias, os processadores de uso específico são componentes de baixo nível embutidos no hardware.

Dessa forma, o sistema operacional não pode se comunicar com esses processadores; eles executam suas tarefas autonomamente. O uso de microprocessadores de uso específico é comum e não transforma um sistema uniprocessador em um multiprocessador. Assim, quando se existe apenas uma CPU de uso geral, o sistema é uniprocessador.

#### 4.5.2 Sistemas multiprocessadores

Nos últimos anos, os sistemas multiprocessadores (também conhecidos como sistemas paralelos ou sistemas multicore) começaram a dominar o cenário da computação. Tais sistemas possuem dois ou mais processadores em estreita comunicação, compartilhando o bus do computador e algumas vezes o relógio, a memória e os dispositivos periféricos. Os sistemas multiprocessadores apareceram pela primeira vez, principalmente, em servidores e, então, migraram para sistemas desktop e laptop. Recentemente, múltiplos processadores têm aparecido em dispositivos móveis como os smartphones e tablets.

Os sistemas multiprocessadores têm três vantagens principais, a primeira delas é o aumento do throughput, já que, com o aumento do número de processadores, espera-se que mais trabalho seja executado em menos tempo. No entanto, a taxa de aumento de velocidade com  $N$  processadores não é  $N$ : é menor do que  $N$ . Quando vários processadores cooperam em uma tarefa, observa-se algum overhead para manter todas as partes trabalhando corretamente. Esse overhead, somado à concorrência por recursos compartilhados, diminui o ganho esperado dos processadores adicionais. Do mesmo modo,  $N$  programadores trabalhando em cooperação não produzem  $N$  vezes o montante de trabalho que um único programador produziria.



Outra vantagem é a economia de escala, pois tais sistemas multiprocessadores podem custar menos do que múltiplos sistemas equivalentes de processador único, já que eles podem compartilhar periféricos, memória de massa e suprimentos de energia. Quando vários programas trabalham sobre o mesmo conjunto de dados, é mais barato armazenar esses dados em um disco, compartilhando-os com todos os processadores, do que usar muitos computadores com discos locais e muitas cópias dos dados.

A terceira vantagem dos sistemas multiprocessadores é o aumento da confiabilidade. Se as funções puderem ser distribuídas apropriadamente entre vários processadores, a falha de um processador não interromperá o sistema, tornando-o apenas mais lento. Se tivermos dez processadores e um falhar, cada um dos nove processadores remanescentes poderá ficar com uma parcela do trabalho do processador que falhou. Assim, o sistema inteiro operará somente 10% mais devagar, em vez de falhar completamente. O aumento da confiabilidade de um sistema de computação é crucial em muitas aplicações. A capacidade de continuar fornecendo serviço proporcionalmente ao nível do hardware remanescente é chamada de degradação limpa.

Alguns sistemas vão além da degradação limpa e são chamados de tolerantes a falhas, porque podem sofrer falha em qualquer um dos componentes e continuar a operar. A tolerância a falhas requer um mecanismo para permitir que a falha seja detectada, diagnosticada e, se possível, corrigida. O sistema HP NonStop (anteriormente chamado de Tandem) usa a duplicação tanto do hardware como do software para assegurar a operação continuada, apesar das falhas. O sistema consiste em múltiplos pares de CPUs, funcionando em lockstep. Os dois processadores do par executam cada instrução e comparam os resultados. Quando o resultado difere, é porque uma CPU do par está falhando e as duas são interrompidas.

O processo que estava sendo executado é, então, transferido para outro par de CPUs e a instrução que falhou é reiniciada. Essa solução é cara, já que envolve hardware especial e considerável duplicação de hardware.

Hoje em dia, os sistemas multiprocessadores em uso são de dois tipos. Alguns sistemas utilizam multiprocessamento assimétrico, no qual cada processador recebe uma tarefa específica. Um processador mestre controla o sistema; e os demais consultam o mestre para instruções ou possuem tarefas predefinidas. Esse esquema define um relacionamento mestre-escravo, onde o processador mestre organiza e aloca o trabalho para os escravos.

Os sistemas mais comuns utilizam multiprocessamento simétrico, Symmetric Multiprocessing (SMP), no qual cada processador executa todas as tarefas do sistema operacional. A sigla SMP significa que todos os processadores são pares; não existe um relacionamento mestre-escravo entre eles. A figura 70 ilustra uma arquitetura SMP típica.

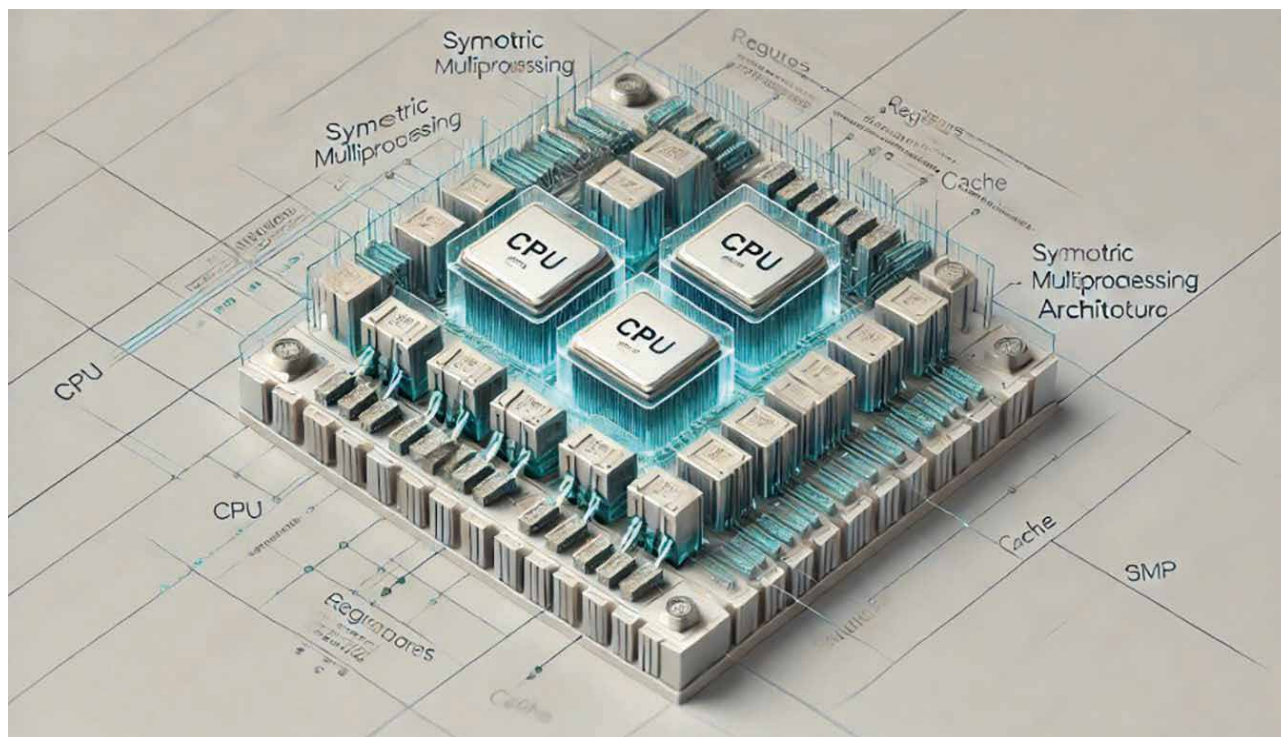


Figura 70 – Arquitetura de multiprocessamento simétrico, imagem gerada em IA Image Generator

Observe que cada processador tem seu próprio conjunto de registradores, assim como um cache privado ou local. No entanto, todos os processadores compartilham memória física. Um exemplo de um sistema SMP é o AIX, uma versão comercial do UNIX projetada pela IBM. Um sistema AIX pode ser configurado de modo a empregar dúzias de processadores. O benefício desse modelo é que muitos processos podem ser executados simultaneamente, onde  $N$  processos podem ser executados se houver  $N$  CPUs, sem causar uma deterioração significativa de desempenho. Entretanto, devemos controlar cuidadosamente o I/O para assegurar que os dados alcancem o processador apropriado. Além disso, como as CPUs são separadas, uma pode ficar ociosa enquanto outra ficará sobrecarregada, resultando em ineficiências. Essas ineficiências podem ser evitadas, se os processadores compartilharem determinadas estruturas de dados. Um sistema multiprocessador desse tipo permitirá que processos e recursos, como a memória, sejam compartilhados dinamicamente entre os vários processadores, podendo diminuir a diferença entre eles. Praticamente, todos os sistemas operacionais modernos – incluindo o Windows, o Mac OS X e o Linux – já suportam o SMP.

A diferença entre multiprocessamento simétrico e assimétrico pode resultar tanto do hardware quanto do software. Um hardware especial pode diferenciar os processadores múltiplos, ou o software pode ser escrito para permitir somente um mestre e múltiplos escravos. Por exemplo, o sistema operacional SunOS Versão 4 da Sun Microsystems fornecia multiprocessamento assimétrico, enquanto a Versão 5 (Solaris) é simétrica no mesmo hardware.

O multiprocessamento adiciona CPUs para aumentar o poder de computação. Se a CPU tem um controlador de memória integrado, o acréscimo de CPUs também pode aumentar o montante de memória endereçável no sistema. De qualquer forma, o multiprocessamento pode fazer com que um sistema altere seu modelo de acesso à memória, do acesso uniforme, Uniform Memory Access (UMA), ao acesso não uniforme, Non-Uniform Memory Access (NUMA). O UMA é definido como a situação em que o acesso a qualquer RAM a partir de qualquer CPU leva o mesmo período de tempo. No NUMA, algumas partes da memória podem demorar mais para serem acessadas do que outras, gerando perda de desempenho. Os sistemas operacionais podem minimizar a perda gerada pelo NUMA por meio do gerenciamento de recursos.

Uma tendência recente no projeto de CPUs é a inclusão de múltiplos núcleos (cores) de computação em um único chip. Tais sistemas multiprocessadores são denominados multicore. Eles podem ser mais eficientes do que vários chips de núcleo único porque a comunicação on-chip (dentro do chip) é mais veloz do que a comunicação between-chip (entre chips). Além disso, um chip com vários núcleos usa bem menos energia do que vários chips de núcleo único.

É importante observar que, enquanto os sistemas multicore são multiprocessadores, nem todos os sistemas multiprocessadores são multicore. Em nossa abordagem dos sistemas multiprocessadores no decorrer deste livro-texto, a menos que mencionado o contrário, usamos o termo mais contemporâneo **multicore**, que exclui alguns sistemas multiprocessadores. Na figura 71, mostramos um projeto dual-core com dois núcleos no mesmo chip. Nesse projeto, cada núcleo tem seu próprio conjunto de registradores, assim como seu próprio cache local. Outros projetos podem usar um cache compartilhado ou uma combinação de caches local e compartilhado. Além das considerações relacionadas com a arquitetura, como a disputa por cache, memória e bus, essas CPUs multicore aparecem para o sistema operacional como  $N$  processadores-padrão. É uma característica que pressiona os projetistas de sistemas operacionais – e programadores de aplicações – a fazer uso desses núcleos de processamento.

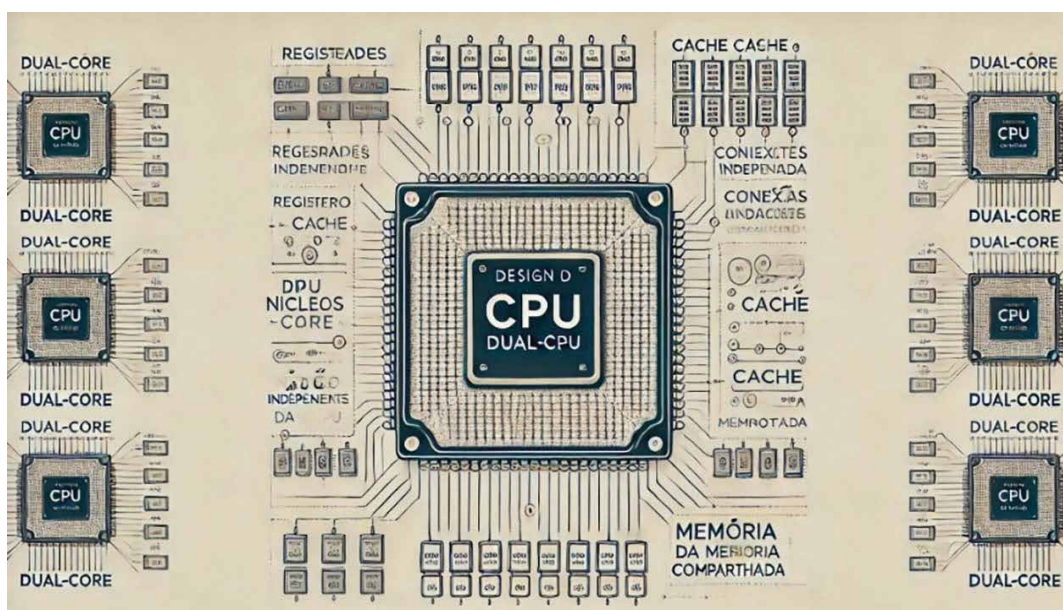


Figura 71 – Projeto dual-core com dois núcleos inseridos no mesmo chip, imagem gerada em IA Image Generator

Para concluir, os servidores blade são um desenvolvimento relativamente recente em que várias placas de processador, placas de I/O e placas de rede são inseridas no mesmo chassi. A diferença entre esses sistemas e os sistemas multiprocessadores tradicionais é que cada placa de processador blade é inicializada independentemente e executa seu próprio sistema operacional. Algumas placas de servidor blade também são multiprocessadoras, o que torna difícil diferenciar tipos de computadores. Em essência, esses servidores são compostos por vários sistemas multiprocessadores independentes.

### 4.5.3 Sistemas Agrupados (Clusters)

Outro tipo de sistema multiprocessador é o sistema em cluster que reúne várias CPUs. Esses sistemas diferem dos sistemas multiprocessadores por serem compostos de dois ou mais sistemas individuais (ou nós) acoplados. Tais sistemas são considerados fracamente acoplados, onde cada nó pode ser um sistema uniprocessador ou um sistema multicore. É preciso ressaltar que a definição de cluster não é unânime; muitos pacotes comerciais divergem quanto à definição de um sistema em cluster e à justificativa de uma forma ser melhor do que a outra. A definição geralmente aceita é a de que computadores em cluster compartilham memória e são estreitamente conectados por uma rede local LAN ou de uma interconexão mais veloz, como a InfiniBand.

O cluster costuma ser usado para fornecer serviço de alta disponibilidade, isto é, o serviço continuará mesmo se um ou mais sistemas do cluster falhar. Geralmente, a alta disponibilidade é obtida pela adição de um nível de redundância ao sistema. Uma camada de software de cluster opera sobre os nós do cluster e cada nó pode monitorar um ou mais outros nós (por meio da LAN). Se a máquina monitorada falhar, a máquina de monitoramento pode apropriar-se da sua memória e reiniciar as aplicações que estavam sendo executadas na máquina que falhou. Os usuários e clientes das aplicações percebem somente uma breve interrupção do serviço.

O cluster pode ser estruturado assimétrica ou simetricamente. No cluster assimétrico, uma das máquinas permanece em modalidade de alerta máximo, enquanto a outra executa as aplicações. A máquina hospedeira em alerta nada faz além de monitorar o servidor ativo. Se esse servidor falhar, o hospedeiro em alerta torna-se o servidor ativo. No cluster simétrico, dois ou mais hospedeiros executam aplicações e se monitoram uns aos outros. É claro que essa estrutura é mais eficiente, já que usa todo o hardware disponível. No entanto, isso requer que mais de uma aplicação esteja disponível para execução.

Já que um cluster é composto por vários sistemas de computação conectados por uma rede, os clusters também podem ser usados para fornecer ambientes de computação de alto desempenho. Esses sistemas podem fornecer um poder de computação significativamente maior do que sistemas de um único processador, ou até mesmo sistemas SMP, porque podem executar uma aplicação concorrentemente em todos os computadores do cluster. No entanto, a aplicação deve ter sido escrita especificamente para se beneficiar do cluster. Isso envolve uma técnica conhecida como paralelização, que divide um programa em componentes separados executados em paralelo em computadores individuais do cluster.



Normalmente, essas aplicações são projetadas para que, após cada nó de computação do cluster ter resolvido sua parte do problema, os resultados de todos os nós sejam combinados em uma solução final.

### 4.5.4 Estrutura do sistema operacional

Agora que já discutimos a organização e a arquitetura básica do sistema de computação, estamos prontos para conversar sobre sistemas operacionais. Um sistema operacional fornece o ambiente dentro do qual os programas são executados.

Internamente, os sistemas operacionais variam bastante em sua composição, já que estão organizados em muitas linhas diferentes. No entanto, há muitas semelhanças, que consideramos nesta seção.

Um dos aspectos mais importantes dos sistemas operacionais é sua capacidade de multiprogramar. Geralmente, um único programa não pode manter a CPU ou os dispositivos de I/O ocupados o tempo todo. Usuários individuais costumam ter vários programas em execução. A multiprogramação aumenta a utilização da CPU organizando os jobs (código e dados) de modo que a CPU tenha sempre um para executar.

Assim, o sistema operacional mantém vários jobs na memória simultaneamente, como podemos ver na figura 72. Já que a memória principal costuma ser muito pequena para acomodar todos os jobs, inicialmente, eles são mantidos em disco no pool de jobs. Esse pool é composto de todos os processos residentes em disco aguardando alocação da memória principal.

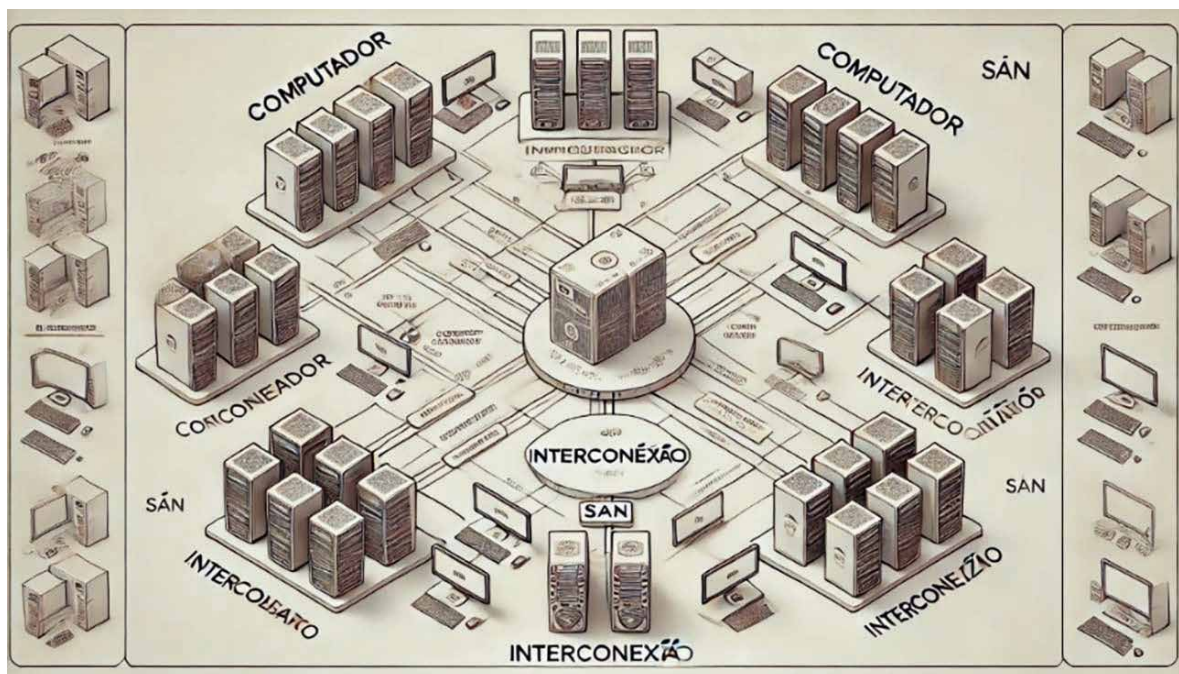


Figura 72 – Estrutura geral de um sistema em cluster, imagem gerada em IA Image Generator

O conjunto de jobs na memória pode ser um subconjunto dos jobs mantidos no pool de jobs. O sistema operacional seleciona e começa a executar um dos jobs da memória. Em dado momento, o job pode ter de aguardar que alguma tarefa, como uma operação de I/O, seja concluída. Em um sistema sem multiprogramação, a CPU permaneceria ociosa. Em um sistema multiprogramado, o sistema operacional simplesmente passa para um novo job e o executa.

Quando **este** job tem de aguardar, a CPU é redirecionada para **outro** job, e assim por diante. Por fim, o primeiro job sai da espera e retoma a CPU. Contanto que pelo menos um job tenha de ser executado, a CPU nunca fica ociosa.

Essa ideia é comum em outras situações da vida. Um advogado não trabalha para apenas um cliente de cada vez, por exemplo, enquanto um caso está aguardando julgamento ou esperando a preparação de documentos, ele pode trabalhar em outro. Se tiver clientes suficientes, nunca ficará ocioso por falta de trabalho.

Os sistemas multiprogramados fornecem um ambiente em que os diversos recursos do sistema (por exemplo: CPU, memória e dispositivos periféricos) são utilizados eficientemente, mas não possibilitam a interação do usuário com o sistema de computação. O tempo compartilhado (ou multitarefa) é uma extensão lógica da multiprogramação. Em sistemas de tempo compartilhado, a CPU executa vários jobs alternando-se entre eles, mas as mudanças de job ocorrem com tanta frequência que os usuários podem interagir com cada programa enquanto ele está em execução.

O tempo compartilhado requer um sistema de computação interativo que forneça comunicação direta entre o usuário e o sistema. O usuário fornece instruções ao sistema operacional ou a um programa, diretamente, usando um dispositivo de entrada, como teclado, mouse, touch pad ou touch screen, e espera os resultados imediatos em um dispositivo de saída. Em contrapartida, o tempo de resposta deve ser curto, normalmente, menos de um segundo.

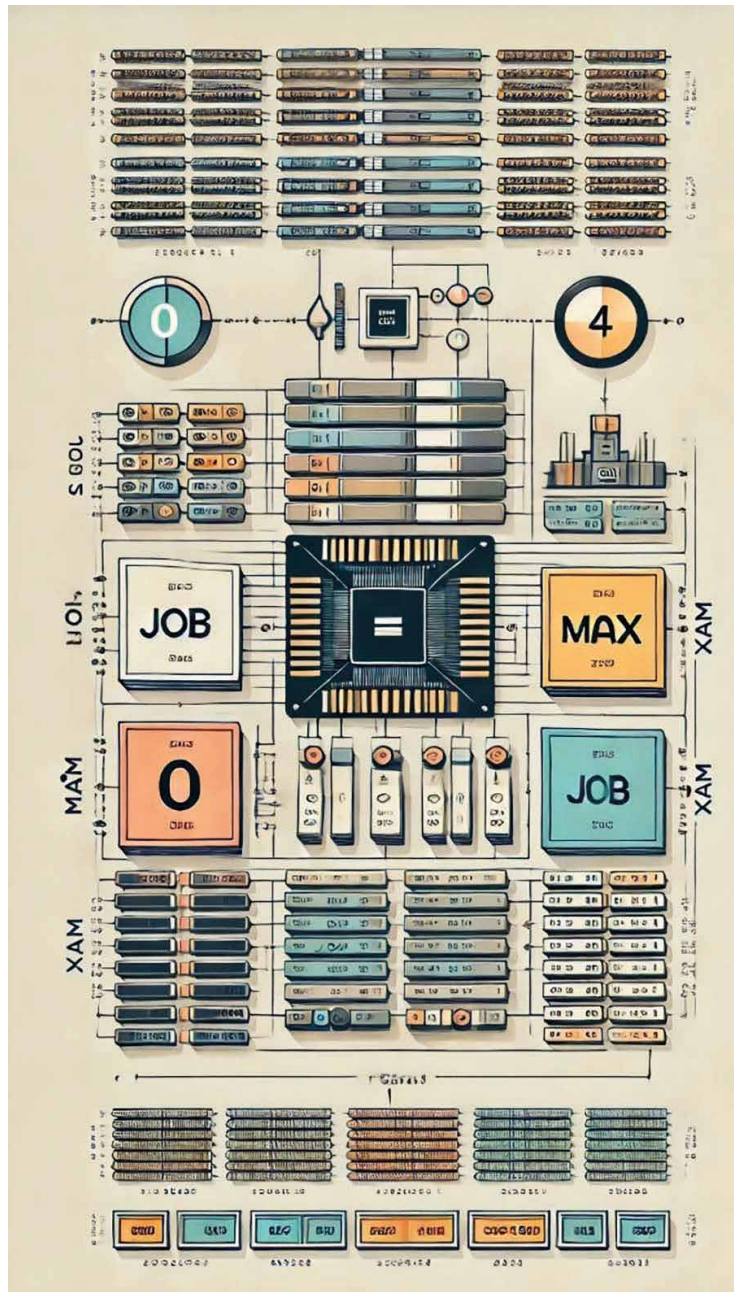


Figura 73 – Layout da memória de um sistema de multiprogramação, imagem gerada em IA Image Generator

Um sistema operacional de tempo compartilhado permite que muitos usuários compartilhem o computador simultaneamente. Como cada ação ou comando em um sistema de tempo compartilhado tende a ser breve, apenas um pequeno tempo de CPU é necessário para cada usuário. Já que o sistema muda rapidamente de um usuário para outro, cada usuário tem a impressão de que o sistema de computação inteiro é dedicado ao seu uso, mesmo que compartilhado entre muitos usuários.

O sistema operacional de tempo compartilhado usa a multiprogramação e o scheduling da CPU para disponibilizar, a cada usuário, uma pequena porção de um computador de tempo compartilhado. Cada usuário tem, pelo menos, um programa separado na memória, sendo que cada programa carregado na memória e em execução é chamado de processo. Quando um processo entra em execução, normalmente, opera apenas por um breve período antes que termine ou tenha de executar I/O, este que pode ser interativo, isto é, a saída é exibida para o usuário e a entrada vem do teclado, do mouse ou de outro dispositivo do usuário. Considerando que o I/O interativo costuma ser executado na "velocidade do usuário", pode levar um longo tempo para ser concluído. A entrada, por exemplo, pode ser limitada pela velocidade de digitação do usuário; sete caracteres por segundo é rápido para as pessoas, mas incrivelmente lento para os computadores. Em vez de deixar a CPU ociosa enquanto essa entrada interativa acontece, o sistema operacional direciona rapidamente a CPU ao programa de algum outro usuário.

O tempo compartilhado e a multiprogramação requerem que vários jobs sejam mantidos simultaneamente na memória. Se vários jobs estão prontos para serem conduzidos à memória e se não houver espaço suficiente para todos, o sistema deve selecionar um entre eles. Essa tomada de decisão envolve o scheduling de jobs.

Quando o sistema operacional seleciona um job no pool de jobs, ele carrega esse job na memória para execução. A existência de vários programas na memória, ao mesmo tempo, requer algum tipo de gerenciamento da memória. Além disso, se vários jobs estão prontos para execução ao mesmo tempo, o sistema deve selecionar que job será executado primeiro. Essa tomada de decisão é o scheduling da CPU. Por fim, a execução de múltiplos jobs concorrentemente requer que suas capacidades de afetar uns aos outros sejam limitadas em todas as fases do sistema operacional, incluindo o scheduling de processos, o armazenamento em disco e o gerenciamento da memória.

Em um sistema de tempo compartilhado, o sistema operacional deve assegurar um tempo de resposta razoável. Às vezes, esse objetivo é alcançado pelo swapping, em que os processos são alternados entre a memória principal e o disco.

Um método comum para garantir um tempo de resposta razoável é a memória virtual, uma técnica que permite a execução de um processo que não se encontra totalmente na memória. A principal vantagem do esquema de memória virtual é que ele permite que os usuários executem programas maiores do que a memória física real. Além disso, ele resume a memória principal a um grande e uniforme array de armazenamento, separando a memória física da memória lógica visualizada pelo usuário. Esse esquema libera os programadores da preocupação com limitações de armazenamento na memória.

Um sistema de tempo compartilhado também deve fornecer um sistema de arquivos. O sistema de arquivos reside em um conjunto de discos; portanto, o gerenciamento de discos deve estar disponível.

Além disso, um sistema de tempo compartilhado fornece um mecanismo para a proteção dos recursos contra uso inapropriado. Para garantir execução ordenada, o sistema deve fornecer mecanismos de sincronização e comunicação entre jobs e deve assegurar que os jobs não fiquem presos em um deadlock, esperando eternamente uns pelos outros.



## 4.5.5 Operação em modalidade dual e multimodalidade

Para garantir a execução apropriada do sistema operacional, temos que ser capazes de distinguir entre a execução de código do sistema operacional e de código definido pelo usuário. A abordagem adotada pela maioria dos sistemas de computação é o fornecimento de suporte de hardware que permite diferenciar as diversas modalidades de execução.

Precisamos de, pelo menos, duas modalidades de operação separadas: a modalidade de usuário e a modalidade de kernel (também chamada de modalidade de supervisor, modalidade de sistema ou modalidade privilegiada). Um bit, chamado bit de modalidade, é adicionado ao hardware do computador para indicar a modalidade corrente, kernel (0) ou usuário (1). Com o bit de modalidade, podemos distinguir entre uma tarefa que é executada em nome do sistema operacional e a que é executada em nome do usuário. Quando o sistema de computação está operando em nome de uma aplicação de usuário, ele está em modalidade de usuário. Mas, quando a aplicação de usuário solicita um serviço do sistema operacional (por meio de uma chamada de sistema), o sistema deve passar da modalidade de usuário para a de kernel de modo a atender à solicitação, como mostrado na figura 74, sendo que essa melhoria de arquitetura também é útil em muitos outros aspectos de operação do sistema.

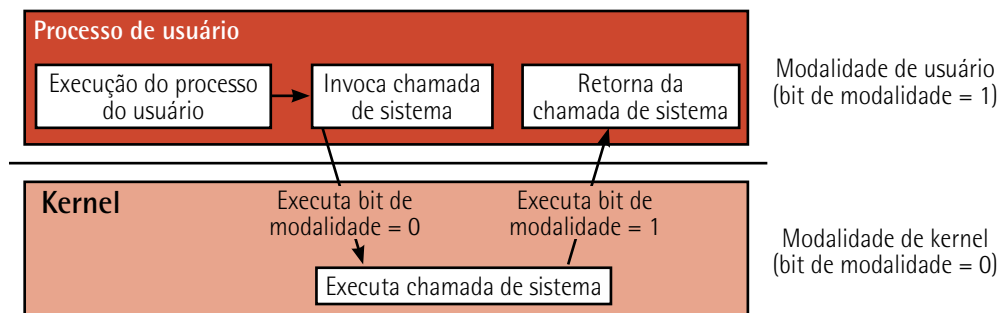


Figura 74 – Transição da modalidade de usuário para a modalidade de kernel

Fonte: Silberschartz (2015, p. 13).

Em tempo de inicialização do sistema, o hardware começa a operar em modalidade de kernel. O sistema operacional é, então, carregado e inicia aplicações de usuário em modalidade de usuário. Sempre que uma exceção ou interrupção ocorre, o hardware passa da modalidade de usuário para a modalidade de kernel (isto é, muda para 0 o estado do bit de modalidade). Portanto, sempre que o sistema operacional obtém o controle do computador, ele está em modalidade de kernel. O sistema sempre passa para a modalidade de usuário (posicionando o bit de modalidade como 1) antes de passar o controle para um programa de usuário.

A modalidade dual de operação fornece os meios para a proteção do sistema operacional contra usuários errantes, e a proteção desses contra eles mesmos. Obtemos essa proteção designando como instruções privilegiadas algumas das instruções de máquina que podem causar erro. O hardware permite que as instruções privilegiadas sejam executadas somente em modalidade de kernel. Se é feita alguma tentativa de executar uma instrução privilegiada em modalidade de usuário, o hardware não executa a instrução, tratando-a como ilegal e interceptando-a (por uma exceção) para o sistema operacional.

A instrução de passagem para a modalidade de kernel é um exemplo de instrução privilegiada. Alguns outros exemplos incluem o controle de I/O, o gerenciamento do timer e o gerenciamento de interrupções.

O conceito de modalidades pode ser estendido para além de duas modalidades (caso em que a CPU utiliza mais de um bit para estabelecer e testar a modalidade). Com frequência, CPUs que suportam virtualização têm uma modalidade separada para indicar quando o gerenciador de máquinas virtuais, Virtual Machine Manager (VMM), e o software de gerenciamento de virtualização estão no controle do sistema. Nessa modalidade, o VMM tem mais privilégios do que os processos de usuário e menos do que o kernel. Ele precisa desse nível de privilégio para poder criar e gerenciar máquinas virtuais, alterando o estado da CPU para fazer isso. Às vezes, também, modalidades diferentes são usadas por vários componentes do kernel. Observemos que, como uma alternativa às modalidades, o projetista da CPU pode usar outros métodos para diferenciar privilégios operacionais. A família de CPUs Intel 64 suporta quatro níveis de privilégios, por exemplo, e suporta a virtualização, mas não tem uma modalidade separada para ela.

Podemos ver, agora, o ciclo de vida da execução de instruções em um sistema de computação. O controle inicial reside no sistema operacional, no qual instruções são executadas em modalidade de kernel. Quando o controle é passado para uma aplicação de usuário, a modalidade é posicionada como modalidade de usuário. O controle acaba sendo devolvido ao sistema operacional por uma interrupção, uma exceção ou uma chamada de sistema.

As chamadas de sistema fornecem o meio para que um programa de usuário solicite ao sistema operacional a execução de tarefas reservadas a ele em nome do programa de usuário. Uma chamada de sistema pode ser invocada de várias maneiras, dependendo da funcionalidade fornecida pelo processador subjacente. De qualquer forma, esse é o método usado por um processo para solicitar uma ação ao sistema operacional. Geralmente, uma chamada de sistema assume a forma de uma exceção para uma localização específica no vetor de interrupções. Essa exceção pode ser executada por uma instrução trap genérica, embora alguns sistemas (como o MIPS) tenham uma instrução syscall específica para invocar uma chamada de sistema.

Quando uma chamada de sistema é executada, ela é tipicamente tratada pelo hardware como uma interrupção de software. O controle passa, por intermédio do vetor de interrupções, para uma rotina de serviço no sistema operacional, e o bit de modalidade é posicionado como modalidade de kernel. A rotina de serviço da chamada de sistema faz parte do sistema operacional, fazendo com que o kernel examine a instrução de interrupção para determinar qual chamada de sistema ocorreu, e um parâmetro indica que tipo de serviço o programa do usuário está solicitando. Informações adicionais exigidas pela solicitação podem ser passadas em registradores, na pilha ou na memória (com ponteiros para as locações na memória passados em registradores). O kernel verifica se os parâmetros estão corretos e legais, executa a solicitação e retorna o controle para a instrução seguinte à chamada de sistema.

A ausência de uma modalidade dual suportada por hardware pode causar falhas graves em um sistema operacional. Por exemplo, o MS-DOS foi escrito para a arquitetura Intel 8088 como não possuindo bit de modalidade e, portanto, não tendo modalidade dual. Um programa de usuário operando incorretamente pode tirar do ar o sistema operacional gravando dados sobre ele; e vários programas poderiam gravar ao mesmo tempo em um dispositivo, com resultados potencialmente desastrosos. Versões modernas da CPU Intel fornecem operação em modalidade dual. Como resultado, a maioria dos sistemas operacionais contemporâneos, como o Microsoft Windows 7 e, também, o Unix e o Linux, se beneficia desse recurso e fornece maior proteção ao sistema operacional.

Uma vez que a proteção de hardware esteja ativa, ela detecta erros que violam as modalidades. Normalmente, esses erros são manipulados pelo sistema operacional. Se um programa de usuário falha de alguma forma, como ao tentar executar uma instrução ilegal ou acessar memória que não faça parte do espaço de endereçamento do usuário, o hardware gera uma exceção para o sistema operacional. A exceção transfere o controle, por meio do vetor de interrupções, para o sistema operacional, da mesma forma que a interrupção faz. Quando ocorre um erro no programa, o sistema operacional deve encerrá-lo anormalmente. Essa situação é manipulada pelo mesmo código de um encerramento anormal solicitado pelo usuário, onde uma mensagem de erro apropriada é fornecida e a memória do programa pode ser despejada. Dessa forma, o despejo da memória é gravado em um arquivo para que o usuário ou o programador possa examiná-lo, possivelmente corrigi-lo e reiniciar o programa.

### 4.5.6 Timer

O timer é utilizado pelo sistema operacional com a finalidade de impedir que um programa de usuário fique travado em um loop infinito. Ele pode ser configurado para interromper o computador após um período especificado. O período pode ser fixo (por exemplo, 1/60 segundos) ou variável (por exemplo, de 1 milissegundo a 1 segundo). Geralmente, um timer variável é implementado por um relógio de marcação fixa e um contador. Assim, o sistema operacional posiciona o contador e cada vez que o relógio marca, o contador é decrementado. Quando o contador atinge 0, ocorre uma interrupção. Por exemplo, um contador de 10 bits com um relógio de 1 milissegundo permite interrupções a intervalos de 1 a 1024 milissegundos em passos de 1 milissegundo.

Antes de retornar o controle ao usuário, o sistema operacional assegura que o timer seja configurado para causar uma interrupção. Quando o timer causa a interrupção, o controle é transferido automaticamente para o sistema operacional, que pode tratar a interrupção como um erro fatal ou dar mais tempo ao programa. É claro que as instruções que modificam o conteúdo do timer são privilegiadas.

Podemos usar o timer para impedir que um programa de usuário seja executado por muito tempo. Uma técnica simples é a inicialização de um contador com o período de tempo em que um programa pode ser executado. Um programa com um limite de tempo de 7 minutos, por exemplo, teria seu contador inicializado em 420. A cada segundo, o timer causaria uma interrupção e o contador seria decrementado de uma unidade. Enquanto o contador é positivo, o controle é retornado ao programa do usuário; quando o contador se torna negativo, o sistema operacional encerra o programa por exceder o limite de tempo designado.

### 4.6 Conceitos básicos sobre processos, memória e arquivos

Um **processo** é a unidade básica de execução em um sistema operacional. Ele representa um programa em execução, incluindo seu código, dados e estado. Em essência, um processo é a abstração de um programa em execução, fornecendo ao sistema operacional um meio de gerenciar e controlar a execução do programa. Um sistema batch executa jobs, enquanto um sistema de tempo compartilhado tem programas de usuário ou tarefas. Até mesmo em um sistema monousuário, o usuário pode executar vários programas ao mesmo tempo: um processador de texto, um navegador web, um pacote de e-mail etc. Mesmo que o usuário possa executar apenas um programa de cada vez, como em um dispositivo embutido que não suporte multitarefa, o sistema operacional deve dar suporte às suas próprias atividades programadas, como o gerenciamento da memória. Em muitos aspectos, essas atividades são semelhantes e, assim, todas são chamadas de processos.

Os termos job e processo são usados de maneira quase intercambiável neste livro-texto. Embora o termo processo seja de nossa preferência, grande parte da teoria e terminologia dos sistemas operacionais foi desenvolvida durante uma época em que a principal atividade dos sistemas operacionais era o processamento de jobs. Seria enganoso evitar o uso de termos comumente aceitos que incluem a palavra job (como scheduling de jobs), simplesmente porque processo substituiu job.

Informalmente, como já mencionado, um processo é um programa em execução. Um processo é mais do que o código do programa, que também é conhecido como seção de texto, ele também inclui a atividade corrente, conforme representada pelo valor do contador do programa e o conteúdo dos registradores do processador. Geralmente, um processo também inclui a pilha do processo que contém dados temporários (como parâmetros de funções, endereços de retorno e variáveis locais), e uma seção de dados, que contém variáveis globais. Um processo também pode incluir um heap, que é a memória dinamicamente alocada durante o tempo de execução do processo (figura 75).

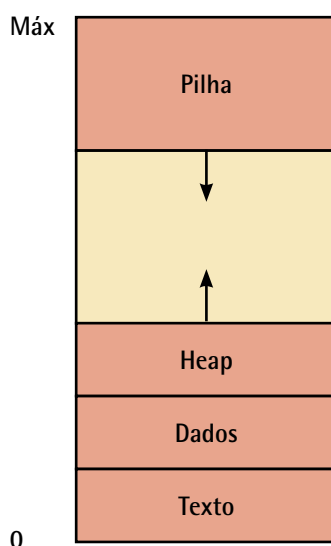


Figura 75 – Processo na memória

No entanto, precisamos enfatizar que um programa por si só não é um processo. Um programa é uma entidade passiva, como um arquivo contendo uma lista de instruções armazenadas em disco (geralmente chamado de arquivo executável). Por outro lado, um processo é uma entidade ativa, com um contador de programa especificando a próxima instrução a ser executada e um conjunto de recursos associados. Um programa torna-se um processo quando um arquivo executável é carregado na memória. Há duas técnicas comuns para a carga de arquivos executáveis: clicar duas vezes em um ícone que representa o arquivo executável ou dar entrada no nome do arquivo executável na linha de comando (como em `prog.exe` ou `a.out`).

Embora dois processos possam estar associados ao mesmo programa, ainda assim, eles são considerados duas sequências de execução separadas. Por exemplo, vários usuários podem estar executando diferentes cópias do programa de e-mail ou o mesmo usuário pode invocar muitas cópias do programa de navegação na web. Cada uma delas é um processo separado e, embora as seções de texto sejam equivalentes, os dados, o heap e as seções de pilha variam. Também é comum haver um processo que gera muitos processos ao ser executado.

Observe que o próprio processo pode ser um ambiente de execução para outros códigos. O ambiente de programação Java fornece um bom exemplo, já que, na maioria dos casos, um programa executável Java é operado dentro da máquina virtual Java (JVM). A JVM executa um processo que interpreta o código Java carregado e realiza ações (por meio de instruções nativas da máquina) em nome desse código. Por exemplo, para executar o programa Java compilado `Program.class`, daríamos entrada em: `java Program`.



### Lembrete

O comando `java` executa a JVM como um processo comum que, por sua vez, executa o programa Java `Program` na máquina virtual. O conceito é o mesmo da simulação, exceto pelo fato de o código ser escrito na linguagem Java, em vez de ser escrito a partir de um conjunto de instruções diferentes.

### Estados de um processo

Um processo pode se encontrar em diferentes estados durante seu ciclo de vida. Os estados mais comuns são:

- **Novo:** o processo está sendo criado, mas ainda não está pronto para execução.
- **Pronto:** o processo está pronto para ser executado, mas está esperando sua vez na fila de processos prontos.
- **Em execução:** o processo está sendo executado pelo processador.
- **Bloqueado:** o processo está aguardando a conclusão de algum evento, como a conclusão de uma operação de E/S.
- **Terminado:** o processo finalizou sua execução.

Esses nomes são arbitrários e variam entre os sistemas operacionais, mas os estados que eles representam são encontrados em todos os sistemas. Certos sistemas operacionais também descrevem, mais apuradamente, os estados do processo. É importante saber que apenas um processo pode estar **em execução** em algum processador a cada instante, mas, concomitantemente, muitos processos podem estar **prontos** e **em espera**. O diagrama de estado correspondente a esses estados é apresentado na figura 76.

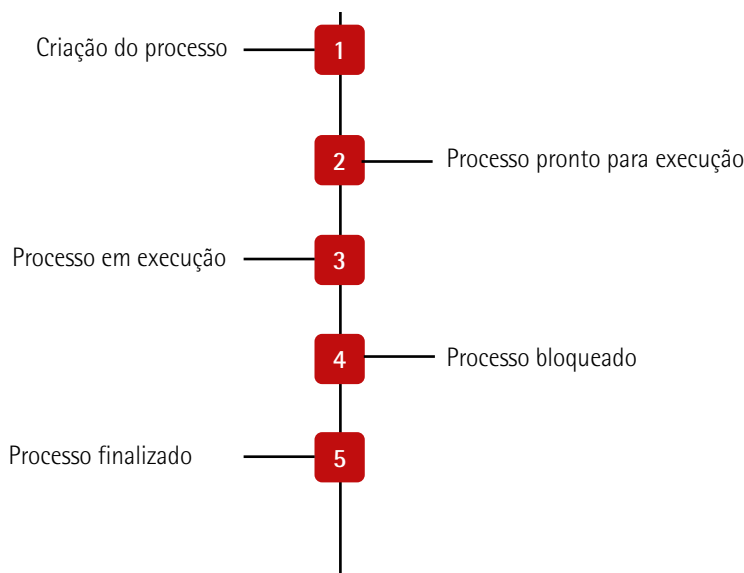


Figura 76 – Ciclo de vida de um processo, imagem gerada em IA Image Generator

### Blocos de processos

Cada processo é representado, no sistema operacional, por um bloco de controle de processo Process Control Block (PCB), também chamado bloco de controle de tarefa (figura 77).

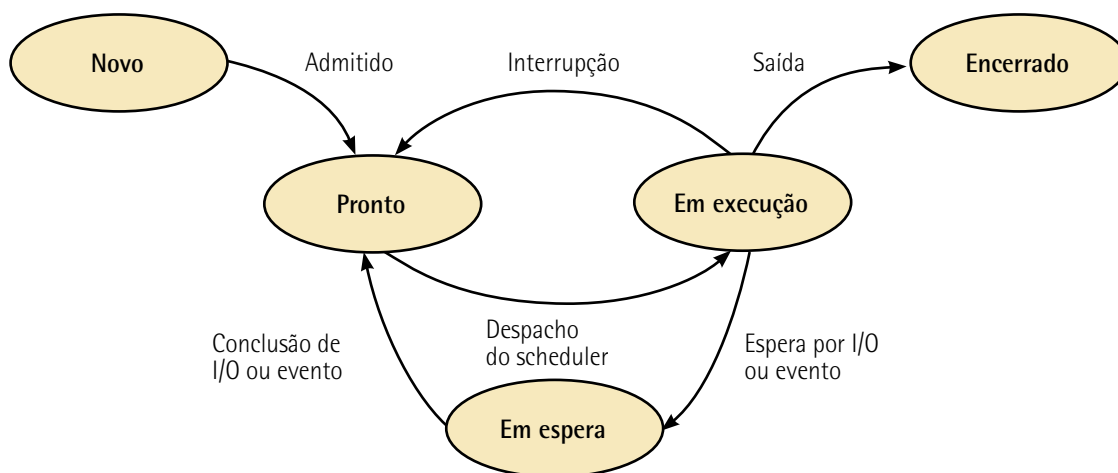


Figura 77 – Diagrama de estado do processo

O bloco de controle de processo contém muitas informações associadas a um processo específico, incluindo:

- **Estado do processo:** o estado pode ser novo, pronto, em execução, em espera, parado e assim por diante.
- **Contador do programa:** o contador indica o endereço da próxima instrução a ser executada para esse processo.
- **Registradores da CPU:** os registradores variam em número e tipo, dependendo da arquitetura do computador. Eles incluem acumuladores, registradores índice, ponteiros de pilhas e registradores de uso geral, além de qualquer informação do código de condição. Junto com o contador do programa, essas informações de estado devem ser salvas quando ocorre uma interrupção, para permitir que o processo seja retomado corretamente mais tarde (figura 78).

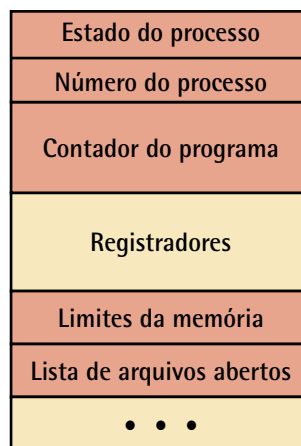


Figura 78 – Bloco de controle de processo (PCB)

Fonte: Silberschartz, Galvin e Gagne (2015, p. 100).

- **Informações de scheduling da CPU:** essas informações incluem a prioridade de um processo, ponteiros para filas de scheduling e quaisquer outros parâmetros de scheduling.
- **Informações de gerenciamento da memória:** essas informações podem incluir itens como o valor dos registradores base e limite e as tabelas de páginas ou as tabelas de segmentos, dependendo do sistema de memória usado pelo sistema operacional.
- **Informações de contabilização:** essas informações incluem o montante de tempo real e de CPU usados, limites de tempo, números de conta, números de jobs ou processos e assim por diante.
- **Informações de status de I/O:** essas informações incluem a lista de dispositivos de I/O alocados ao processo, uma lista de arquivos abertos e assim por diante.

Em suma, o PCB serve como o repositório de quaisquer informações que possam variar de um processo para outro.

### Threads

O modelo de processo discutido até agora sugere que um processo é um programa que executa apenas um thread. Por exemplo, quando um processo está executando um programa de processamento de texto, um único thread de instruções está sendo executado. Esse thread de controle único permite que o processo execute apenas uma tarefa de cada vez. O usuário não pode digitar caracteres e executar, simultaneamente, o corretor ortográfico dentro do mesmo processo, por exemplo. A maioria dos sistemas operacionais modernos estendeu o conceito de processo para permitir que um processo tenha múltiplos threads de execução e, assim, possa desempenhar mais de uma tarefa de cada vez. Esse recurso é particularmente benéfico em sistemas multicore, em que múltiplos threads podem ser executados em paralelo. Em um sistema que suporte threads, o PCB é expandido de modo a incluir informações para cada thread, sendo também necessárias outras alterações no sistema como um todo para que ele suporte threads.

### Representação de processos no Linux

O bloco de controle de processo é representado no sistema operacional Linux pela estrutura em C `task_struct`, que é encontrada no arquivo de inclusão `<linux/sched.h>` no diretório de código-fonte do kernel. Essa estrutura contém todas as informações necessárias à representação de um processo, incluindo o estado do processo, informações de scheduling e de gerenciamento da memória, a lista de arquivos abertos, ponteiros para o pai do processo e uma lista de seus filhos e irmãos (o pai de um processo é o processo que o criou; seus filhos são quaisquer processos que ele tenha criado; seus irmãos são os filhos que têm o mesmo processo-pai). Alguns desses campos incluem:

- `long state; /* estado do processo */`
- `struct sched_entity se; /* informações de scheduling */`
- `struct task_struct *parent; /* pai desse processo */`
- `struct list_head children; /* filhos desse processo */`
- `struct files_struct *files; /* lista de arquivos abertos */`
- `struct mm_struct *mm; /* espaço de endereçamento desse processo */`



Por exemplo, o estado de um processo é representado pelo campo long state nessa estrutura. Dentro do kernel do Linux, todos os processos ativos são representados com o uso de uma lista duplamente encadeada de task\_struct. O kernel mantém um ponteiro (current) para o processo em execução corrente no sistema, como mostrado a seguir:

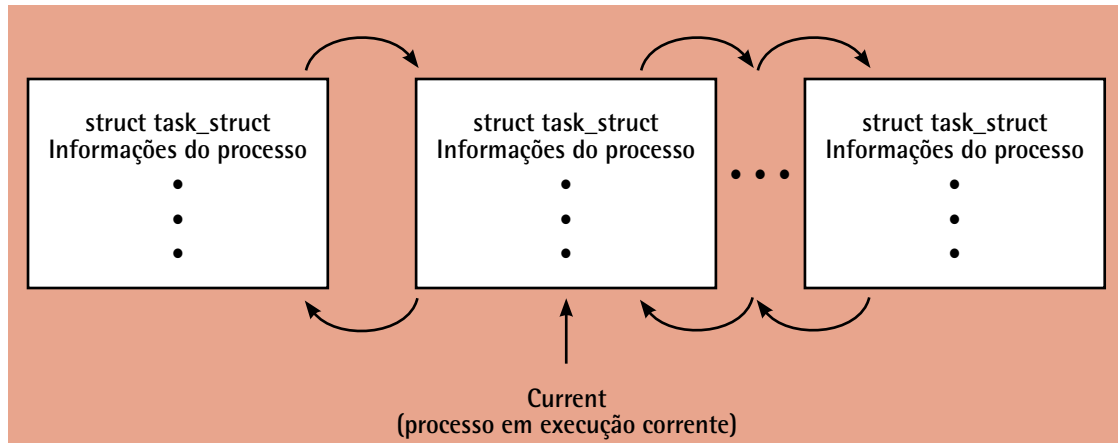


Figura 79 – Processo em execução corrente

Fonte: Silberschartz, Galvin e Gagne (2015, p. 102).

Como mostrado na figura 79, o kernel pode manipular um dos campos da `task_struct` de um processo especificado. Vamos assumir que o sistema queira alterar o estado do processo em execução corrente para o valor `new_state`, se `current` for um ponteiro para o processo em execução corrente, seu estado será alterado com a linha a seguir:

```
current->state = new_state;
```

O sistema também inclui outras filas. Quando a CPU é alocada a um processo, ele é executado por algum tempo e, eventualmente, para, sendo interrompido ou colocado em espera pela ocorrência de um evento específico, como a conclusão de uma solicitação de I/O. Suponha que o processo faça uma solicitação de I/O para um dispositivo compartilhado, como um disco. Já que há muitos processos no sistema, o disco pode estar ocupado com a solicitação de I/O de algum outro processo, o que faz com que o processo possa ter que esperar pelo disco.

A lista de processos em espera por um dispositivo de I/O específico é chamada fila do dispositivo, onde cada dispositivo tem sua própria fila (figura 80).

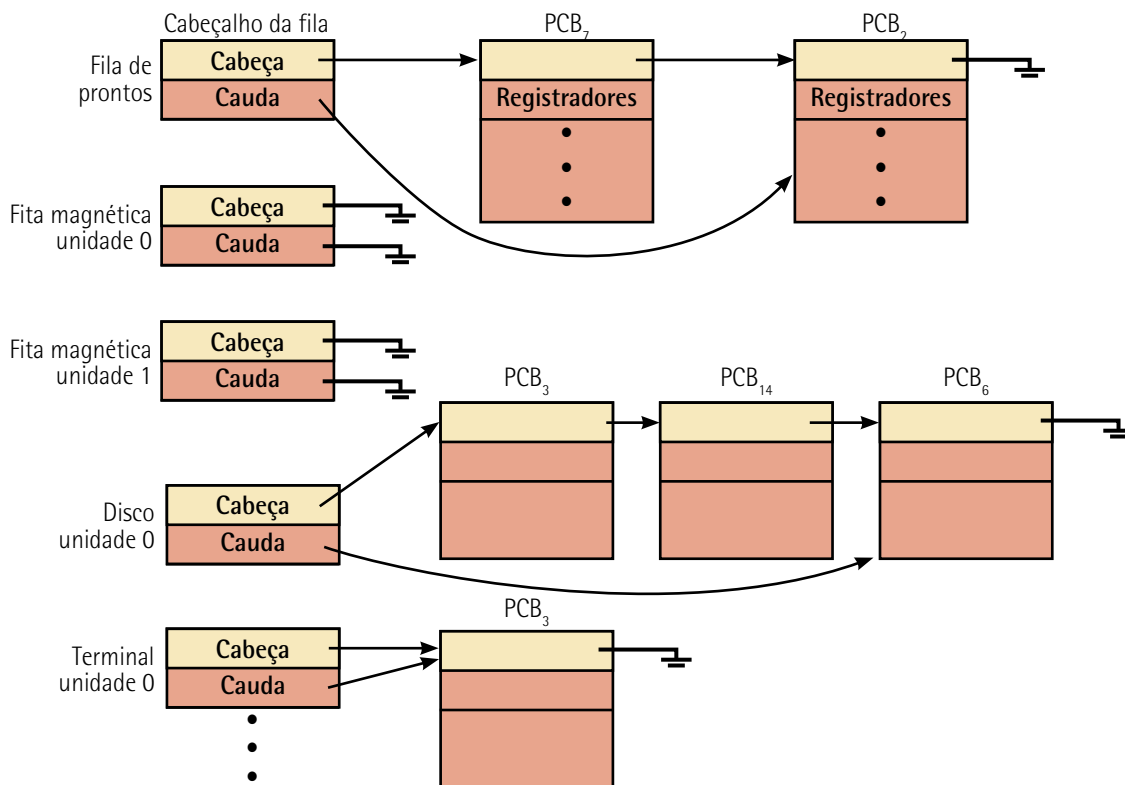


Figura 80 – A fila de prontos e várias filas de dispositivo de I/O

Fonte: Silberschartz, Galvin e Gagne (2015, p. 103).

Uma representação comum do scheduling de processos é o diagrama de enfileiramento, como o da figura 81. Cada caixa retangular representa uma fila, contendo dois tipos de filas: a fila de prontos e um conjunto de filas de dispositivos. Os círculos representam os recursos que servem às filas e as setas indicam o fluxo de processos no sistema.

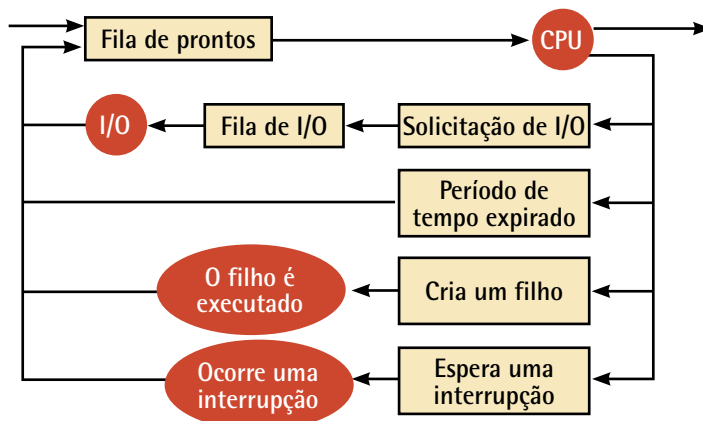


Figura 81 – Representação do scheduling de processos em diagrama de enfileiramento

Fonte: Silberschartz, Galvin e Gagne (2015, p. 104).

Inicialmente, um novo processo é inserido na fila de prontos. Ele aguarda até ser selecionado para execução ou ser despachado. Uma vez que a CPU seja alocada ao processo e este entre em execução, um entre vários eventos pode ocorrer:

- O processo pode emitir uma solicitação de I/O e, então, ser inserido em uma fila de I/O.
- O processo pode criar um novo processo-filho e esperá-lo terminar.
- O processo pode ser removido à força da CPU, como resultado de uma interrupção, e ser devolvido à fila de prontos.

Nos dois primeiros casos, o processo eventualmente passa do estado de espera para o estado de pronto e é, então, devolvido à fila de prontos. Um processo continua esse ciclo até terminar, momento em que é removido de todas as filas e tem seu PCB e recursos desalocados.

Com frequência, em um sistema batch, são submetidos mais processos do que é possível executar imediatamente. Esses processos são reservados em um dispositivo de armazenamento de massa (tipicamente, um disco), no qual são mantidos para execução posterior. O scheduler de longo prazo, ou scheduler de jobs, seleciona processos nesse pool e os carrega na memória para execução. O scheduler de curto prazo, ou scheduler da CPU, seleciona entre os processos que estão prontos para execução e aloca a CPU a um deles.

A principal diferença entre esses dois schedulers está na frequência de execução. O scheduler de curto prazo deve selecionar, com frequência, um novo processo para a CPU. O processo pode ser executado por apenas alguns milissegundos antes de entrar em espera por uma solicitação de I/O. Quase sempre, o scheduler de curto prazo é executado pelo menos uma vez a cada 100 milissegundos. Em razão do curto tempo entre as execuções, o scheduler de curto prazo deve ser rápido. Se ele levar 10 milissegundos para decidir executar um processo por 100 milissegundos, então,  $10/(100 + 10) = 9\%$  da CPU serão usados (desperdiçados) simplesmente no scheduling do trabalho.

O scheduler de longo prazo é executado com muito menos frequência; minutos podem separar a criação de um novo processo e o processo seguinte. O scheduler de longo prazo controla o grau de multiprogramação (o número de processos na memória). Se o grau de multiprogramação for estável, a taxa média de criação de processos deve ser igual à taxa média de processos que estão deixando o sistema. Portanto, o scheduler de longo prazo pode ter que ser invocado apenas quando um processo deixa o sistema. Em razão do intervalo mais longo entre as execuções, o scheduler de longo prazo pode dispor de mais tempo para decidir que processo deve ser selecionado para execução.

É importante que o scheduler de longo prazo faça uma seleção cuidadosa. Em geral, a maioria dos processos pode ser descrita como limitada por I/O ou limitada por CPU. Um processo limitado por I/O é aquele que gasta mais do seu tempo fazendo I/O do que executando computação. Um processo limitado por CPU, de outro lado, gera solicitações de I/O com menos frequência, usando a maior parte de seu tempo executando computação. É importante que o scheduler de longo prazo selecione um bom mix de processos, formado de processos limitados por I/O e limitados por CPU. Se todos os processos forem

limitados por I/O, a fila de prontos ficará, quase sempre, vazia e o scheduler de curto prazo terá pouco a fazer. Se todos os processos forem limitados por CPU, a fila de espera por I/O ficará, quase sempre, vazia, os dispositivos não serão usados e, novamente, o sistema ficará desbalanceado. Assim, o sistema com melhor desempenho terá uma combinação de processos limitados por CPU e processos limitados por I/O.

Em alguns sistemas, o scheduler de longo prazo pode estar ausente ou ser mínimo. Por exemplo, sistemas de tempo compartilhado, como os sistemas UNIX e Microsoft Windows, quase sempre, não têm scheduler de longo prazo e, simplesmente, inserem cada novo processo na memória para o scheduler de curto prazo. A estabilidade desses sistemas depende tanto de uma limitação física (como a quantidade de terminais disponíveis) quanto da natureza de autoajuste dos usuários humanos. Quando o desempenho desce a níveis inaceitáveis em um sistema multiusuário, alguns usuários simplesmente desistem.

### 4.7 Sistemas monotarefa, multitarefa, multiusuário

#### 4.7.1 Sistemas Monotarefa

Em sistemas monotarefa, o processador executa apenas uma tarefa de cada vez. Quando uma tarefa é iniciada, o processador dedica todos os seus recursos a ela, completando-a antes de iniciar a próxima. O foco é na execução sequencial, garantindo a conclusão completa de cada tarefa antes de iniciar outra. Algumas características dos sistemas monotarefa são:

- **Simplicidade:** os sistemas monotarefa são relativamente simples de implementar, pois não requerem mecanismos complexos de gerenciamento de tarefas.
- **Eficiência:** em cenários com poucas tarefas, os sistemas monotarefa podem ser eficientes, utilizando todos os recursos do processador para cada tarefa.
- **Determinismo:** a ordem de execução das tarefas é previsível, tornando-as adequadas para aplicações onde o tempo de resposta é crítico.
- **Limitado:** o processamento de apenas uma tarefa por vez limita a capacidade de executar múltiplas tarefas em paralelo, o que pode afetar a produtividade em cenários mais complexos.

Os sistemas monotarefa são mais adequados para aplicações simples e com um fluxo de trabalho sequencial. Eles são comumente encontrados em dispositivos embarcados, como calculadoras, impressoras e alguns dispositivos IoT.

O sistema operacional MS-DOS é um exemplo de sistema monotarefa. Ele tem um interpretador de comandos que é invocado quando o computador é iniciado (figura 82a). Como o MS-DOS é monotarefa, ele usa um método simples para executar um programa e não cria um novo processo. Ele carrega o programa na memória, gravando sobre grande parte de si próprio, para dar ao programa o máximo de memória possível (figura 82b). Em seguida, posiciona o ponteiro de instruções para a primeira instrução do programa. Assim, o programa é executado e, então, ou um erro causa uma exceção, ou o programa executa uma chamada de sistema para ser encerrado. De uma forma ou de outra, o código de erro é

salvo na memória do sistema para uso posterior. Após essa ação, a pequena parte do interpretador de comandos que não foi sobreposta, retoma a execução. Sua primeira tarefa é recarregar o resto do interpretador de comandos a partir do disco. Em seguida, o interpretador de comandos torna o código de erro anterior disponível para o usuário ou para o próximo programa.

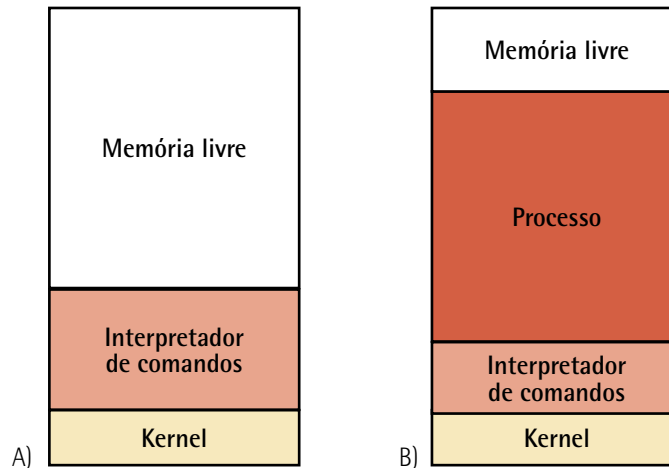


Figura 82 – Execução do MS-DOS: a) na inicialização do sistema; b) executando um programa

Fonte: Silberschartz, Galvin e Gagne (2015, p. 104).

### 4.7.2 Sistemas multitarefa

Sistemas multitarefa permitem que o processador execute múltiplas tarefas ao mesmo tempo. Essa capacidade é alcançada através de mecanismos de gerenciamento de tempo, que dividem o tempo do processador entre as tarefas em execução. O processador alterna rapidamente entre as tarefas, dando a impressão de que todas estão sendo executadas simultaneamente.

Sistemas multitarefa são amplamente utilizados em computadores modernos, smartphones e servidores. Eles permitem a execução simultânea de várias tarefas, melhorando a capacidade de resposta e a produtividade dos usuários. Algumas características dos sistemas multitarefa são:

- **Multiprocessamento:** sistemas multitarefa possibilitam a execução de múltiplas tarefas em paralelo, aumentando a produtividade e a capacidade de resposta.
- **Gerenciamento de recursos:** o gerenciamento de tempo garante que todos os recursos do processador sejam utilizados de forma eficiente, compartilhados entre as tarefas em execução.
- **Interrupções:** o sistema é capaz de lidar com interrupções, permitindo que tarefas de alta prioridade sejam interrompidas e retomadas posteriormente.
- **Complexidade:** o gerenciamento de múltiplas tarefas requer mecanismos mais complexos para garantir a sincronização e a comunicação entre as tarefas, aumentando a complexidade do sistema.

FreeBSD (derivado do UNIX de Berkeley) é um exemplo de sistema multitarefa. Quando um usuário faz login no sistema, o shell que ele escolheu é executado. Esse shell é semelhante ao shell do MS-DOS, já que aceita os comandos e executa os programas que o usuário solicita. No entanto, como o FreeBSD é um sistema multitarefa, o interpretador de comandos pode continuar em operação enquanto outro programa é executado. Para iniciar um novo processo, o shell executa uma chamada de sistema `fork()`. Em seguida, o programa selecionado é carregado na memória por meio de uma chamada de sistema `exec()` e é executado. Dependendo da maneira como o comando foi emitido, o shell espera que o processo termine ou o executa "em background". No último caso, o shell solicita imediatamente outro comando. Quando um processo está sendo executado em background, ele não pode receber entradas diretamente do teclado, porque o shell está usando esse recurso. Portanto, o I/O é executado por meio de arquivos ou de uma GUI. Enquanto isso, o usuário pode solicitar ao shell que execute outros programas, monitore o progresso do processo em execução, altere a prioridade desse programa e assim por diante. Quando o processo é concluído, ele executa uma chamada de sistema `exit()` para ser encerrado, retornando ao processo que o invocou, o código de status 0 ou um código de erro diferente de zero. Esse código de status ou de erro fica, então, disponível para o shell ou para outros programas.

### 4.7.3 Sistemas multiusuário

Sistemas multiusuário permitem que múltiplos usuários acessem e compartilhem recursos computacionais simultaneamente. Cada usuário tem um espaço de trabalho isolado, com seus próprios arquivos, programas e configurações, garantindo a segurança e a privacidade dos dados. Algumas características dos sistemas multiusuário são:

- **Compartilhamento de recursos:** os sistemas multiusuário otimizam o uso de recursos computacionais, permitindo que vários usuários compartilhem o hardware e os softwares.
- **Segurança:** mecanismos de segurança, como autenticação de usuários e controle de acesso, protegem os dados e a privacidade de cada usuário.
- **Gerenciamento de usuários:** administradores do sistema podem gerenciar as contas dos usuários, definir permissões de acesso e controlar o uso dos recursos.
- **Escalabilidade:** sistemas multiusuário podem ser escalados para acomodar um número crescente de usuários, aumentando a capacidade do sistema.

Em sistemas multiusuários, devemos adicionalmente proteger os processos de usuário uns dos outros. Essa proteção deve ser fornecida pelo hardware porque o sistema operacional não costuma intervir entre a CPU e seus acessos à memória (em razão do comprometimento do desempenho). O hardware implementa essa proteção de várias maneiras diferentes, como mostramos no decorrer do tópico.

Os sistemas multiusuários são a base de servidores web, bancos de dados e sistemas operacionais como Linux e Windows. Eles facilitam o compartilhamento de recursos computacionais entre vários usuários, permitindo a colaboração e a comunicação.

O Windows 11 é um sistema operacional multiusuário que suporta acesso simultâneo por meio de serviços distribuídos ou por meio de múltiplas instâncias da GUI via servidor de terminais Windows. As edições de servidor do Windows 11 suportam sessões de servidor de terminais simultâneos a partir de sistemas Windows para desktop. As edições para desktop do servidor de terminais multiplexam o teclado, o mouse e o monitor entre sessões de terminais virtuais para cada usuário conectado. Esse recurso, chamado permuta rápida de usuário, permite que os usuários se revezem no console de um PC sem ter que fazer logoff e login.

Mencionamos anteriormente que parte da implementação da GUI migrou para a modalidade de kernel no Windows NT 4.0. Ela começou a migrar novamente para a modalidade de usuário no Windows Vista, que incluiu o gerenciador de janelas de desktop Desktop Window Manager (DWM) como um processo de modalidade de usuário. O DWM implementa a composição do desktop do Windows, fornecendo a aparência da interface Aero do Windows no topo do software gráfico Windows DirectX. O DirectX continua sendo executado no kernel, assim como o código que implementa os modelos anteriores de geração de janelas e de elementos gráficos (Win64k e GDI) do Windows.

### 4.8 Sistemas distribuídos

Um sistema distribuído é um conjunto de sistemas de computação fisicamente separados e possivelmente heterogêneos que são conectados em rede para conceder aos usuários acesso aos vários recursos que o sistema mantém. O acesso a um recurso compartilhado aumenta a velocidade de computação, a funcionalidade, a disponibilidade dos dados e a confiabilidade. Alguns sistemas operacionais generalizam o acesso à rede como um tipo de acesso ao arquivo, com os detalhes da conexão de rede contidos no driver de dispositivo da interface de rede; outros fazem os usuários invocarem funções da rede especificamente. Geralmente, os sistemas contêm uma combinação das duas modalidades, como o FTP e o NFS. Os protocolos que criam um sistema distribuído podem afetar bastante a utilidade e a popularidade desse sistema.

Uma rede, em uma descrição simples, é uma via de comunicação entre dois ou mais sistemas. Os sistemas distribuídos dependem da conexão de rede para oferecer sua funcionalidade. As redes variam de acordo com os protocolos usados, as distâncias entre os nós e a mídia de transporte. O TCP/IP é o protocolo de rede mais comum e fornece a arquitetura básica da Internet. A maioria dos sistemas operacionais dá suporte ao TCP/IP, inclusive os de uso geral. Alguns sistemas suportam protocolos proprietários para satisfazer suas necessidades. Para um sistema operacional, um protocolo de rede precisa apenas de um dispositivo de interface — um adaptador de rede, por exemplo — com um driver de dispositivo para gerenciá-lo, assim como um software para manipular os dados.

As redes são caracterizadas de acordo com as distâncias entre seus nós, podendo operar das seguintes formas: uma rede local (LAN) conecta computadores dentro de uma sala, um prédio ou um campus; já uma rede de longa distância (WAN) geralmente conecta prédios, cidades ou países — por exemplo, uma empresa global pode ter uma WAN para conectar seus escritórios mundialmente.



Ainda, as redes LAN e WAN podem executar um protocolo ou vários, mas, com o decorrer do tempo, o contínuo surgimento de novas tecnologias trouxe novos tipos de rede. É o caso da rede metropolitana (MAN), que pode conectar prédios dentro de uma cidade, e dos dispositivos Bluetooth e 802.11, que usam tecnologia sem fio para se comunicar por uma distância de vários quilômetros, criando uma rede de área pessoal (PAN) entre um telefone e um fone de ouvido, ou um smartphone e um computador desktop.

As mídias que suportam as redes são igualmente variadas, incluindo fios de cobre, fibras trançadas, transmissões sem fio entre satélites, parabólicas de ondas curtas e rádios. Quando dispositivos de computação são conectados a telefones celulares, se cria uma rede. Até mesmo a comunicação em infravermelho de muito pouco alcance pode ser usada na conexão de rede. Em um nível rudimentar, sempre que os computadores se comunicam, eles usam ou criam uma rede. Essas redes também variam em seu desempenho e confiabilidade.

Alguns sistemas operacionais levaram o conceito de redes e sistemas distribuídos para além da noção de fornecimento de conectividade de rede. Um sistema operacional de rede é um sistema operacional que fornece recursos como o compartilhamento de arquivos pela rede e um esquema de comunicação que permite que diferentes processos em diferentes computadores troquem mensagens. Um computador executando um sistema operacional de rede, atua independentemente de todos os outros computadores da rede, embora tenha conhecimento da rede e seja capaz de se comunicar com outros computadores conectados. Um sistema operacional distribuído fornece um ambiente menos autônomo, fazendo com que os diferentes computadores se comuniquem com proximidade suficiente para dar a impressão de que apenas um único sistema operacional controla a rede.

Os sistemas distribuídos permitem que os usuários compartilhem recursos em hospedeiros geograficamente dispersos conectados por uma rede de computadores. Tais recursos podem ser fornecidos pelo modelo cliente-servidor ou do modelo entre pares, onde a virtualização envolve a abstração de um hardware de computador em vários ambientes de execução diferentes. Isso ocorre, pois a computação em nuvem usa um sistema distribuído para incluir serviços em uma "nuvem", em que os usuários podem acessá-los de locais remotos.

Em contrapartida, sistemas operacionais de tempo real são projetados para ambientes embutidos, como dispositivos de consumidores, automóveis e robótica. A partir disso, o movimento do software livre criou milhares de projetos de código-fonte aberto, inclusive sistemas operacionais. É devido a esse movimento que estudantes podem usar códigos-fontes como ferramenta de aprendizado, podendo modificar programas e testá-los, ajudar a encontrar e corrigir bugs, explorar sistemas operacionais maduros e completos, compiladores, ferramentas, interfaces de usuário e outros tipos de programas.

O GNU/Linux e o BSD UNIX são sistemas operacionais de código-fonte aberto. As vantagens do software livre e do código-fonte aberto possibilitam o aumento da quantidade e da qualidade de projetos de código-fonte aberto, levando a um acréscimo no número de indivíduos e empresas que usam esses projetos.

Um sistema distribuído é um conjunto de nós fracamente acoplados interconectados por uma rede de comunicação. Do ponto de vista de um nó específico de um sistema distribuído, o resto dos nós e seus respectivos recursos são remotos, enquanto seus próprios recursos são locais.

Os nós de um sistema distribuído podem variar em tamanho e função, podendo incluir pequenos microprocessadores, computadores pessoais e grandes sistemas de computação de uso geral. Esses processadores recebem vários nomes, como processadores, sítios, máquinas e hospedeiros, a depender do contexto em que são mencionados. Aqui, usaremos, principalmente, sítio para indicar a localização de uma máquina, e nó para nos referir a um sistema específico de um sítio. Geralmente, um nó em um sítio, o servidor, tem um recurso que outro nó em outro sítio, o cliente (ou usuário), gostaria de usar.

Há quatro razões principais para a construção de sistemas distribuídos: compartilhamento de recursos, aceleração do processamento, confiabilidade e comunicação.

### **Compartilhamento de recursos**

Se vários sítios diferentes (com diferentes recursos) estiverem conectados uns aos outros, o usuário de um sítio poderá usar os recursos disponíveis em outro sítio. Por exemplo, um usuário do sítio A poderia usar uma impressora a laser localizada no sítio B. Enquanto isso, um usuário do sítio B poderia acessar um arquivo residente em A. Em geral, o compartilhamento de recursos em um sistema distribuído fornece mecanismos para o compartilhamento de arquivos de sítios remotos, o processamento de informações em um banco de dados distribuído, a impressão de arquivos em sítios remotos, o uso de dispositivos de hardware remotos especializados (como um supercomputador) e a execução de outras operações.

### **Aceleração do processamento**

Quando um processamento específico pode ser dividido em subprocessamentos que possam ser executados concorrentemente, um sistema distribuído nos permite distribuir os subprocessamentos entre os diversos sítios. Por poderem ser executados concorrentemente, os subprocessamentos fornecem aceleração do processamento. Além disso, se um sítio específico estiver sobrecarregado com jobs, alguns deles podem ser transferidos para outros sítios menos carregados. Essa transferência de jobs é denominada compartilhamento de carga ou migração de jobs. O compartilhamento de carga automatizado, em que o sistema operacional distribuído transfere jobs automaticamente, ainda não é comum em sistemas comerciais.

### **Confiabilidade**

Quando um sítio falha em um sistema distribuído, os sítios restantes podem continuar operando, dando ao sistema melhor confiabilidade. Se o sistema for composto por várias instalações autônomas grandes (isto é, computadores de uso geral), uma falha em uma delas não deve afetar as outras. Se, no entanto, o sistema for composto por máquinas pequenas, cada uma responsável por alguma função crucial do sistema (como o servidor web ou o sistema de arquivos), uma única falha pode interromper a operação do sistema inteiro. Em geral, com redundância suficiente (tanto de hardware quanto de dados), o sistema pode continuar operando, mesmo se algum de seus sítios tiver falhado.

A falha em um sítio deve ser detectada pelo sistema, e uma ação apropriada pode ser necessária para a recuperação. O sistema não deve mais usar os serviços desse sítio. Além disso, se a função do sítio defeituoso puder ser assumida por outro sítio, o sistema deve assegurar que a transferência da função ocorra corretamente. Logo, quando o sítio defeituoso for recuperado ou reparado, mecanismos devem estar disponíveis para integrá-lo novamente ao sistema sem demais problemas.

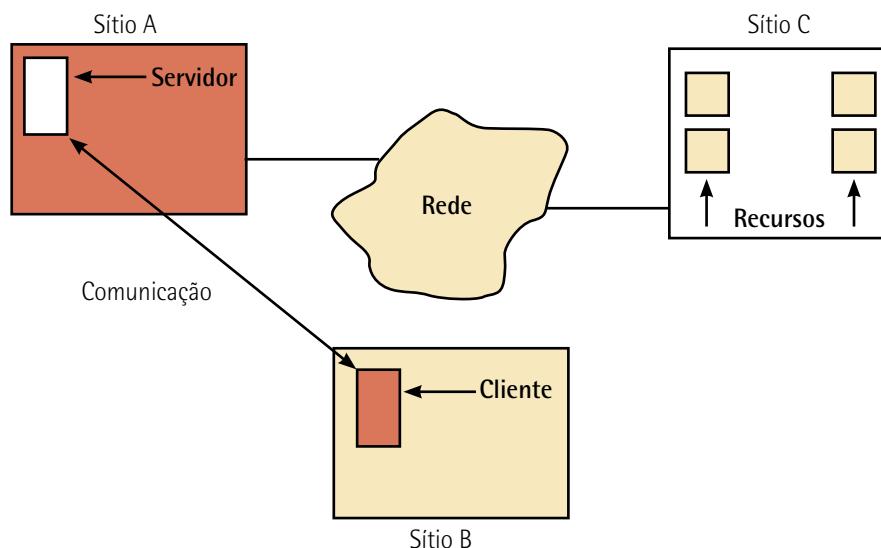


Figura 83 – Um sistema distribuído

Fonte: Silberschartz, Galvin e Gagne (2015, p. 579).

### Comunicação

Quando vários sítios estão conectados por uma rede de comunicação, seus usuários têm a oportunidade de trocar informações. Em baixo nível, mensagens são passadas entre os sistemas, do mesmo modo como mensagens são passadas entre processos no sistema de mensagens de um único computador. Graças à transmissão de mensagens, toda a funcionalidade de alto nível encontrada nos sistemas autônomos pode ser expandida para englobar o sistema distribuído. Essas funções incluem transferência de arquivos, login, correio e chamadas de procedimento remotas (RPCs).

A vantagem de um sistema distribuído é que essas funções podem ser executadas a grandes distâncias. Duas pessoas em sítios geograficamente distantes podem colaborar em um projeto, por exemplo, transferindo os arquivos do projeto, fazendo login nos sistemas remotos uns dos outros para executar programas e trocando e-mails para coordenar o trabalho, os usuários reduzem as limitações inerentes ao trabalho a longas distâncias.

As vantagens dos sistemas distribuídos têm resultado em uma tendência que abrange toda a indústria e ruma para o downsizing. Muitas empresas estão substituindo seus mainframes por redes de estações de trabalho ou computadores pessoais. As empresas obtêm melhor relação custo-benefício (isto é, melhor funcionalidade pelo custo), mais flexibilidade na localização de recursos e expansão de instalações, melhores interfaces de usuário e manutenção mais fácil.

## 4.9 Sistemas de tempo real

O termo sistema de tempo real (real-time system) foi usado ao longo da história da computação para denotar diferentes classes de sistemas. Originalmente, na época em que o acesso principal ao computador ocorria através de cartões de papel e listagens impressas, a interação do usuário com a sua aplicação através de um terminal alfanumérico caracterizava aquela aplicação como sendo de tempo real. Quando o emprego de terminais se tornou corrente, tempo real passou a ser usado como sinônimo de alto desempenho. A partir da década de 1980, o termo foi consolidado como não relacionado com desempenho, mas, sim, com a capacidade de atender requisitos temporais explícitos.

Na verdade, a expressão "sistemas de tempo real" é uma forma reduzida para o termo completo "sistemas computacionais com requisitos de tempo real". Os sistemas computacionais com requisitos de tempo real são aqueles submetidos a requisitos de natureza temporal não triviais. Tais requisitos podem aparecer de várias formas: um prazo máximo para a execução de uma dada tarefa, um período no qual dada tarefa deve ser sempre executada, um intervalo máximo ou mínimo de tempo entre duas ações, um prazo máximo para a validade dos dados etc.

Nos sistemas de tempo real, os resultados e as ações do sistema devem ser corretos não apenas logicamente e aritmeticamente, mas também precisam ser corretos temporalmente. Em outras palavras, não basta realizar as ações corretas e calcular os valores corretos, é necessário também realizar as ações e apresentar os valores no tempo correto.

Em geral, sistemas computacionais são desenvolvidos seguindo a filosofia "fazer o trabalho usando o tempo necessário". Para sistemas de tempo real, a filosofia passa a ser "fazer o trabalho usando o tempo disponível". Em outras palavras, os aspectos temporais do sistema não estão limitados a uma questão de maior ou menor desempenho, mas estão diretamente associados com a funcionalidade do sistema. Se o compilador C demora um segundo a mais ou a menos para compilar o programa, ele continua funcionando, desde que gere o código executável corretamente. Assim, o tempo que o compilador demora está associado com o seu desempenho. Entretanto, por exemplo, um videogame de corrida de carros onde o carro demora um segundo para reagir ao acionar o freio, é um videogame que não funciona. Um sistema de tempo real "não funciona" se ele não cumprir os requisitos temporais de sua especificação.

Na maioria das vezes, é fácil identificar uma dada aplicação computacional como sendo ou não sendo de tempo real. No entanto, nem sempre é assim. Existem sistemas onde atrasos são tolerados, desde que dentro de certos padrões, o que dificulta uma simples classificação como tempo real ou não tempo real. A existência de diferentes tipos de sistemas de tempo real leva, naturalmente, a diferentes abordagens para implementar e depois para verificar a corretude destes sistemas.

Aplicações com requisitos de tempo real surgem em muitos mercados. A seguir, listamos alguns exemplos que podem ser considerados de tempo real. A lista não é exaustiva, mas mostra claramente o amplo espectro de aplicações que apresentam requisitos temporais. Ao longo dos próximos tópicos, diferentes abordagens serão apresentadas para lidar com diferentes tipos de aplicações.

- **Automação industrial:** na indústria, há centenas de máquinas e equipamentos controlados por computador que apresentam requisitos temporais. O rigor dos requisitos temporais varia bastante de indústria para indústria (manufatura, petroquímica, setor elétrico etc.) e de equipamento para equipamento (controle de processos químicos, braços robóticos, tornos automatizados, máquinas de corte a laser etc.), mas sempre estão presentes. Além disso, atualmente, sistemas industriais podem ser muito grandes, abrangendo inúmeros computadores e várias redes interconectadas, além de operarem em diferentes escalas de tempo, conforme a tarefa em questão (controle, supervisão, gestão da produção etc.).
- **Veículos:** nas últimas décadas, a utilização de sistemas computacionais dentro de veículos cresceu muito. Isso vale para qualquer tipo de veículo, incluindo aviões, automóveis e satélites. No contexto da aviação, os sistemas aviônicos (avionic systems) já controlam dezenas de funções a bordo das grandes aeronaves. O piloto automático é um exemplo óbvio de aplicação com requisitos de tempo real. Automóveis também seguem essa tendência, através do uso de dezenas de unidades de controle eletrônico, Electronic Control Units (ECU), para controlar diversas partes do veículo. O correto comportamento temporal é imprescindível para aplicações como injeção eletrônica, transmissão automática e freios Anti-Blocking System (ABS). Atualmente, a tendência está aumentando devido ao crescente interesse em carros autônomos.
- **Defesa:** os sistemas militares modernos são fortemente baseados em tecnologia computacional. Por exemplo, considere o sistema de defesa de um navio de guerra. Nele, sinais de radar são recolhidos e processados digitalmente em um banco de dados usado para identificar o objeto em questão, onde a decisão de disparar ou não disparar contra ele deve ser tomada, ocorrendo dentro de um prazo rigoroso estabelecido pelo fato de o objeto poder ser um míssil inimigo. Muitos cenários desse tipo podem ser encontrados em sistemas militares.
- **Telecomunicações:** as centrais telefônicas eletromecânicas do passado foram substituídas por computadores. Ao chegar à central, a voz é digitalizada e toda a operação é controlada por software. Existem regulamentações específicas para operações telefônicas, principalmente no caso de concessões públicas. Uma central telefônica passa a ser uma aplicação de tempo real quando são incluídos requisitos temporais na sua especificação, tais como tempo máximo até o tom de discar, tempo máximo para completar a chamada, tarifação etc. Além, é claro, da própria qualidade do áudio.
- **Videogames:** na maioria dos jogos, a jogabilidade (qualidade do jogo) está associada com os tempos apresentados pelo jogo em resposta às ações do jogador. Uma exceção são os jogos baseados em turnos, mas em jogos com perseguições, tiroteios ou simulação de veículos, os tempos das reações que o computador apresenta ao jogador são fundamentais para a percepção de qualidade. Por exemplo, retomando um exemplo prévio, se o jogador vira o volante do carro para a esquerda e o computador atrasa em refletir isto no comportamento do carro simulado, o jogo não funciona.

- **Mercado financeiro:** cada vez mais as bolsas de valores e mercadorias em todo o mundo são computadorizadas. O tradicional pregão onde os corretores anunciam suas ofertas foi substituído por um conjunto de computadores que recebem as ordens de compra e venda eletronicamente. Sistemas em software (chamados de robôs pelos operadores do mercado) são construídos para monitorar as cotações dos mercados e disparar ordens de compra e venda aproveitando janelas de oportunidade temporárias, ou seja, negociação automática. Esses sistemas são de tempo real, de forma que de nada adianta uma excelente estratégia de negócios que é concebida após o fechamento da janela de oportunidade. Os requisitos temporais são por vezes tão exigentes que as ordens dos investidores são geradas por software. Os sistemas de negociação automática (automatic trading systems) podem ser instalados em um computador hospedado em espaço físico disponibilizado pela própria bolsa de valores e mercadorias (direct market access via conexão direta – co-location).

Todo sistema de tempo real apresenta forte acoplamento do sistema computacional com o seu ambiente. Existe uma forte relação com o mundo físico e o processamento é ativado por estímulos do ambiente. No caso de um radar, a detecção de um objeto requer sua análise; no caso de um vídeo, a passagem do tempo requer a exibição do próximo quadro (frame).



### Saiba mais

Para conhecer um pouco mais sobre sistemas de tempo real e critérios de escalonamento, leia o capítulo 5 do livro abaixo:

MACHADO, F. B.; MAIA, L. P. *Arquitetura de sistemas operacionais*. 5. ed. Rio de Janeiro: LTC, 2013.



### Resumo

A unidade II teve como objetivo introduzir os conceitos fundamentais da lógica digital, que servem como base para o funcionamento de qualquer sistema computacional moderno. Assim, exploramos os elementos básicos da lógica digital, como portas lógicas, clocks e memória, a fim de promover uma maior compreensão de como os dados são processados e armazenados em computadores.

As portas lógicas são os blocos de construção básicos da lógica digital, que recebem um ou mais sinais de entrada e produzem um sinal de saída, baseado em uma função lógica específica. Existem diversos tipos de portas lógicas, cada uma com sua função específica, como AND, OR, NOT, XOR e NAND. Essas portas são combinadas para criar circuitos mais complexos, capazes de realizar operações lógicas e matemáticas.

Além disso, as portas lógicas desempenham um papel crucial na execução de operações lógicas e matemáticas, como adição, subtração, multiplicação e divisão. Elas também são usadas para controlar o fluxo de dados em um sistema computacional, como ativar ou desativar dispositivos, filtrar dados e determinar o caminho de um sinal.

A combinação de portas lógicas permite criar circuitos digitais mais complexos, como registradores, contadores, decodificadores, multiplexadores, decodificadores e muito mais. Esses circuitos podem ser utilizados para realizar tarefas específicas em sistemas computacionais.

Ao longo desta unidade, também abordamos o conceito de clocks, que é fundamental para o funcionamento de sistemas digitais, regulando o ritmo de processamento dos dados. Os clocks são sinais periódicos que controlam a sincronização das operações dentro de um sistema digital. Eles garantem que todas as operações ocorram de forma sincronizada e organizada, evitando erros e instabilidades no sistema.

Estudamos sobre a memória Random Access Memory (RAM), que é um tipo de memória volátil, ou seja, os dados são apagados quando o sistema é desligado. É usada para armazenar os dados utilizados pelo sistema no momento, como programas em execução e arquivos abertos. A memória RAM é rápida, permitindo que o sistema acesse os dados rapidamente, o que contribui para um bom desempenho.



Além da memória RAM, também vimos a memória Read-Only Memory (ROM), que é um tipo de memória não volátil, ou seja, os dados são armazenados permanentemente, mesmo quando o sistema é desligado. A memória ROM é utilizada para armazenar instruções de inicialização do sistema, dados de configuração e software básico. Ela é um tipo de memória de leitura, ou seja, os dados podem ser lidos, mas não modificados.

A partir do conhecimento das portas lógicas, passamos a estudar sobre construção de circuitos e como construir circuitos digitais mais complexos. Esses circuitos podem ser usados para realizar diversas tarefas, como somar números, controlar o fluxo de dados em um sistema computacional ou realizar operações lógicas.

Já os circuitos combinacionais são aqueles que produzem uma saída que depende apenas dos seus sinais de entrada, sem memória. Eles são usados para realizar operações lógicas simples, como somar, subtrair, multiplicar e comparar dados.

Quanto aos circuitos sequenciais, esses são aqueles que possuem memória, o que significa que a saída depende dos sinais de entrada e do estado anterior do circuito. Eles são usados para realizar operações mais complexas, como contar, armazenar dados ou controlar o fluxo de informações em um sistema computacional.

Ao adentrarmos o tópico sobre a lógica digital, vimos como ela é a base fundamental para a construção de qualquer sistema computacional moderno. A lógica digital é utilizada em diversos componentes, desde os chips de memória e processadores até os dispositivos periféricos. Através dela, podemos criar circuitos que processam informações, controlam o fluxo de dados, armazenam informações e realizam operações complexas.

Por fim, ao vermos processadores e periféricos, estudamos sobre o processador, também conhecido como CPU, que é o componente central de um sistema computacional. Ele é responsável por executar instruções e processar dados. O processador é construído com base em circuitos digitais complexos, utilizando portas lógicas para realizar operações lógicas e matemáticas. Os dispositivos periféricos, como teclados, mouses, monitores, impressoras e unidades de armazenamento, também utilizam a lógica digital, sendo projetados para se comunicarem com o processador, transmitindo e recebendo informações e realizando funções específicas de acordo com seu propósito.



### Exercícios

**Questão 1.** Ao longo dos anos, é possível observar a evolução dos circuitos de lógica digital desde as primeiras máquinas de calcular mecânicas criadas por Charles Babbage até a integração de milhões de transistores em chips modernos. Nesse percurso, surgiu a adoção de arquiteturas como a arquitetura de von Neumann, baseada em um conjunto de princípios que definem como a memória, a CPU e os dispositivos de entrada/saída se interconectam por meio de barramentos de dados e de controle. Essa evolução chegou a incluir conceitos como a computação quântica, a lógica programável em FPGAs e o desenvolvimento de portas lógicas mais avançadas (XOR, XNOR, NAND e NOR).

Também é necessário enfatizar a importância dos clocks, que sincronizam as operações dentro dos processadores e o uso de memórias de diferentes tipos (ROM, RAM, cache etc.) para equilibrar custo e desempenho. A questão, então, envolve compreender como esses elementos históricos e técnicos culminaram na criação de CPUs poderosas, frequentemente multicore, que realizam milhões de instruções por segundo, usando técnicas de paralelismo e microarquiteturas complexas. Diante disso, avalie as alternativas a seguir e assinale a que melhor se enquadra na forma como essa evolução impacta a lógica digital e a forma como projetamos e otimizamos sistemas computacionais.

A) A lógica digital manteve-se, basicamente, inalterada desde a época de Charles Babbage, pois o uso de álgebras booleanas e de portas lógicas pouco mudou na prática. Mesmo com o surgimento de transistores e circuitos integrados, a concepção fundamental de hardware continua a mesma, sem incrementar a eficiência de processamento nem reduzir custos.

B) A adoção da arquitetura de von Neumann e o surgimento de dispositivos de lógica programável, como FPGAs, suprimiram por completo a importância de instruções pré-compiladas nos programas. Em vez disso, todo software tornou-se reconfigurável e executado diretamente em portas lógicas, o que dispensou o papel de um sistema operacional ou de linguagens de alto nível.

C) As portas lógicas modernas, incluindo XOR, XNOR, NAND, NOR e combinações mais avançadas, representam a base de cálculos binários e oferecem suporte à computação quântica em seu estágio atual. Na prática, cada FPGA e cada CPU já contam com qubits análogos e podem atingir velocidade exponencial para tarefas de alta complexidade, sem a necessidade de hardware quântico dedicado.

D) A miniaturização de transistores, a integração em larga escala (LSI e VLSI) e as inovações na microarquitetura de processadores transformaram completamente o cenário da lógica digital. Esse cenário possibilitou o surgimento de CPUs capazes de processar bilhões de instruções por segundo, com múltiplos núcleos, caches de diversos níveis e paralelismo no nível de instruções. Além disso, as portas lógicas evoluíram para suportar projetos reconfiguráveis (FPGAs) e aplicações emergentes em IA e computação quântica, sem abandonar o fundamento booleano que sustenta todo o sistema binário.

E) O uso de clocks para sincronizar operações deixou de ser relevante no momento em que se alcançou a escala de gigahertz. Agora, todos os processadores funcionam de maneira assíncrona, baseados em conceitos de computação quântica, o que garante velocidades inatingíveis pelos antigos mecanismos de bordas de subida e de descida de pulso.

Resposta correta: alternativa D.

### Análise da questão

A alternativa D menciona a passagem histórica desde as máquinas de calcular primitivas até a era dos chips densamente integrados, o que indica que o cerne booleano da lógica digital permanece, mas ganhou aplicações mais versáteis e um nível de desempenho muito maior. Ressalta, ainda, a expansão para IA, FPGA e possíveis avanços em computação quântica, sem contradizer as bases clássicas.

**Questão 2.** Os sistemas operacionais exercem papel fundamental na definição de como recursos de hardware e de software são gerenciados, oferecendo serviços tanto para aplicativos quanto para usuários. Nesse contexto, um dos grandes desafios está em equilibrar as demandas de conveniência e de eficiência. Por exemplo, em estações de trabalho interativas, priorizam-se a facilidade de uso e a capacidade de tempo de resposta; já em ambientes de servidores de grande porte, busca-se otimizar o uso dos recursos, a fim de garantir que nenhum usuário individual exceda sua cota de processamento ou de memória. Em contrapartida, há também sistemas operacionais distribuídos que se concentram em aumentar a disponibilidade, a escalabilidade e a confiabilidade ao compartilhar processos e dados entre diversos nós interconectados. Considere todas essas abordagens e assinale a alternativa que descreve de maneira mais abrangente como os sistemas operacionais equilibram conveniência, eficiência e necessidades de ambientes distintos, levando em conta fatores como multiprogramação, compartilhamento de recursos, modularização e controle de interrupções.

A) Esta abordagem enfatiza exclusivamente a conveniência para o usuário final e ignora fatores essenciais, como o controle rígido da alocação de recursos e a implementação de processos de segurança e de isolamento. Os sistemas operacionais que se limitam a essa visão costumam apresentar interfaces extremamente amigáveis, mas sofrem com problemas de estabilidade e com a impossibilidade de lidar com requisitos que exijam maior robustez, como a operação de aplicações críticas em tempo real ou a execução simultânea de dezenas de processos em servidores de larga escala. Dessa forma, ao negligenciar aspectos de eficiência, esses sistemas não conseguem oferecer um equilíbrio adequado entre a experiência do usuário e a otimização do hardware disponível.

B) Este modo de organização gira em torno de uma lógica em que todos os processos do usuário são executados simultaneamente, sem qualquer distinção de prioridade nem de controle por parte do sistema operacional. Embora pareça oferecer alta disponibilidade ao usuário, tal método acaba saturando o processador, a memória e os dispositivos de entrada/saída, o que torna inviável qualquer forma de modularização ou de escalonamento. Além disso, a ausência de mecanismos de supervisão e de troca de contexto faz com que o sistema não tenha meios de proteção contra conflitos entre processos nem suporte a interrupções adequadas, o que inviabiliza soluções que equilibrem conveniência e eficiência.

C) Nesta concepção, o sistema operacional seria estruturado unicamente a partir da perspectiva de sistemas de tempo real rigorosos, em que todo o modelo de execução se baseia no cumprimento estrito de prazos (deadlines). Essa visão concentra a maior parte dos esforços na previsibilidade das operações de entrada e de saída e na sincronização determinística entre processos. Porém, ao assumir que prazos estritos devem ser aplicados em todos os cenários, o sistema deixa de contemplar necessidades de uso interativo em PCs, de alta escalabilidade em servidores, ou até mesmo de modularização típica em arquiteturas distribuídas. Assim, não cobre a ampla diversidade de demandas de sistemas operacionais modernos.

D) Esta proposta combina o princípio da multiprogramação, que visa a manter o processador ocupado, executando diversos processos ao mesmo tempo com estratégias de escalonamento que consideram prioridades e tempos de resposta. Paralelamente, adota-se a modularização em diferentes camadas ou serviços do kernel para tornar o sistema mais flexível e seguro, o que permite a troca de contexto e a sincronização de processos por meio de interrupções controladas. Esse modelo pode ser adaptado tanto para sistemas de grande porte, que priorizam a eficiência de recursos, quanto para sistemas pessoais, que focam na conveniência. Além disso, mecanismos de compartilhamento e proteção de memória, associando processos e usuários distintos, promovem equilíbrio entre simplicidade, desempenho e segurança.

E) Esta alternativa enfatiza exclusivamente a arquitetura de sistemas de arquivos distribuídos, em que cada nó da rede atua quase que de forma independente e só se conecta eventualmente para obter recursos remotos. Embora seja interessante para reduzir pontos de falha e melhorar a disponibilidade de arquivos, a ausência de supervisão centralizada compromete a padronização de serviços básicos, como o escalonamento de CPU ou de memória, além de dificultar a implementação de segurança unificada. Em última análise, essa abordagem não consegue atender à necessidade de conveniência em máquinas pessoais nem de robustez em servidores, pois não há controle eficaz sobre o compartilhamento de dispositivos I/O e processos.

Resposta correta: alternativa D.

### Análise da questão

A alternativa D descreve como um sistema operacional pode combinar técnicas de multiprogramação (mantendo a CPU ocupada), de modularização (organizando funções do kernel em camadas ou módulos), de controle de interrupções (gerenciamento de eventos internos e externos) e de escalonamento (definindo prioridades e limites de uso). Essa composição torna o sistema adaptável a cenários diversos, desde desktops focados na experiência do usuário até grandes servidores orientados para eficiência e disponibilidade, mantendo segurança e confiabilidade.

---

---

---

---

---

---