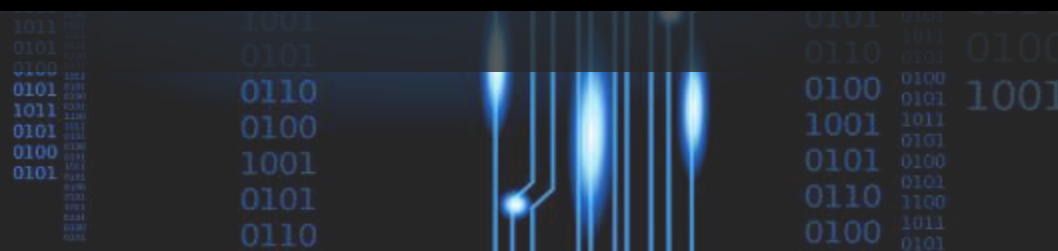




# Introdução à Computação (I.C)

Unidade 03

Prof. Daniel Caixeta



# Conteúdo programático

## 7

### A arquitetura: Estruturas e funcionalidades dos *hardwares* computacionais.

- 7.1. Introdução.
- 7.2. Arquiteturas de computadores.
- 7.3. Os componentes básicos da arquitetura.
  - 7.3.1. Memória.
  - 7.3.2. Memória principal.
  - 7.3.3. Tipos de memórias.
  - 7.3.4. Memória *cache*.
  - 7.3.5. Unidade Central de Processamento (UCP).
  - 7.3.6. Unidades de entrada e saída (E/S ou I/O).
  - 7.3.7. O barramento.
  - 7.3.8. Registradores e *Datapaths*.
  - 7.3.9. *Pipeline*.

### Referências





## 7. A ARQUITETURA.

Estruturas e funcionalidades dos *hardwares* computacionais.

## 7.1. INTRODUÇÃO

- Um sistema computacional é integrado por *hardware* e *software*.
- Neste sistema abriga-se programas fornecidos por fabricantes de *hardware* e *software*, além dos desenvolvidos pelo próprio usuário.

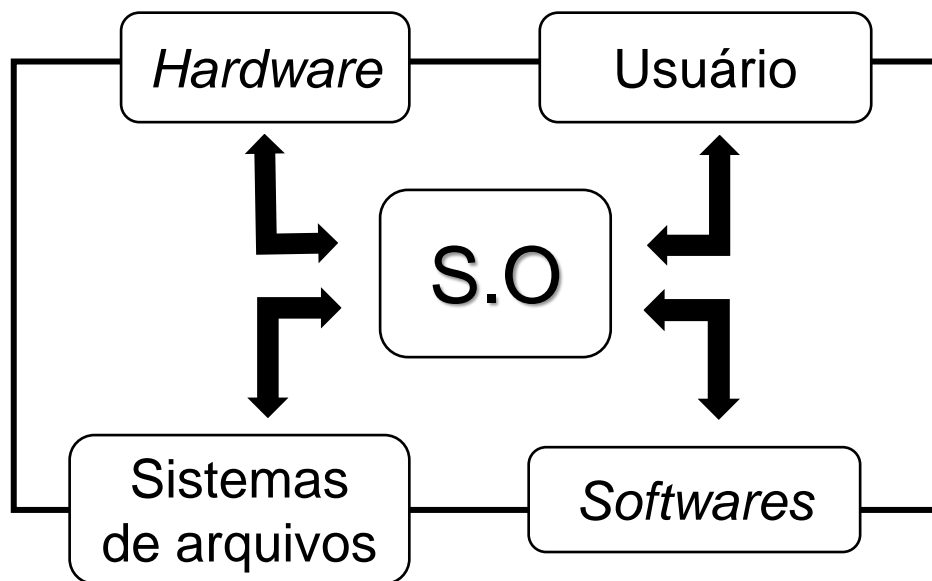


Figura 1. O básico em um sistema computacional.  
(CAIXETA, 2019).

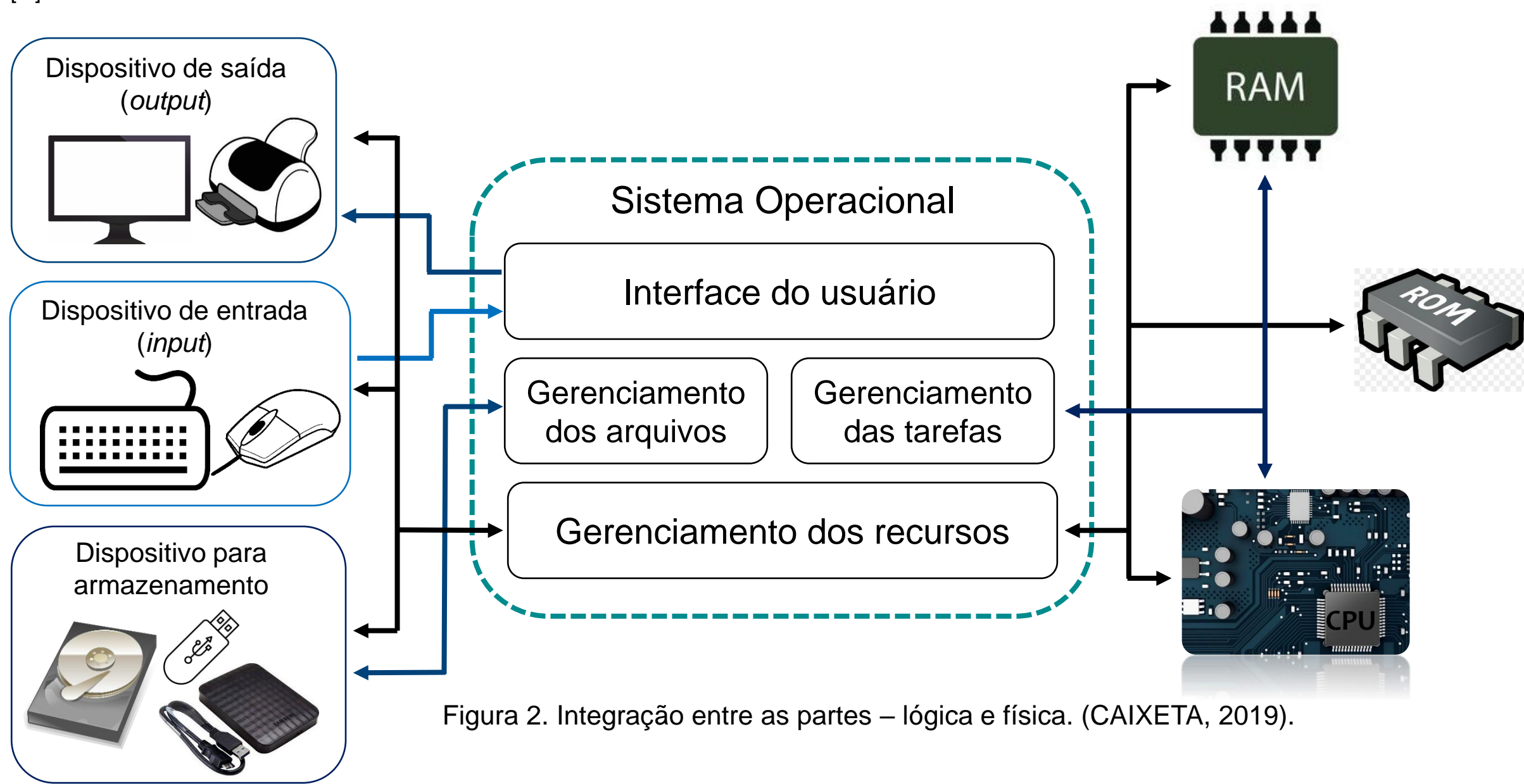


Figura 2. Integração entre as partes – lógica e física. (CAIXETA, 2019).



- Outra forma de identificar os elementos da arquitetura está na orientação de pontes norte e sul nas placas mães. Com essa indicação é possível reconhecer a maioria das unidades funcionais do *hardware*.

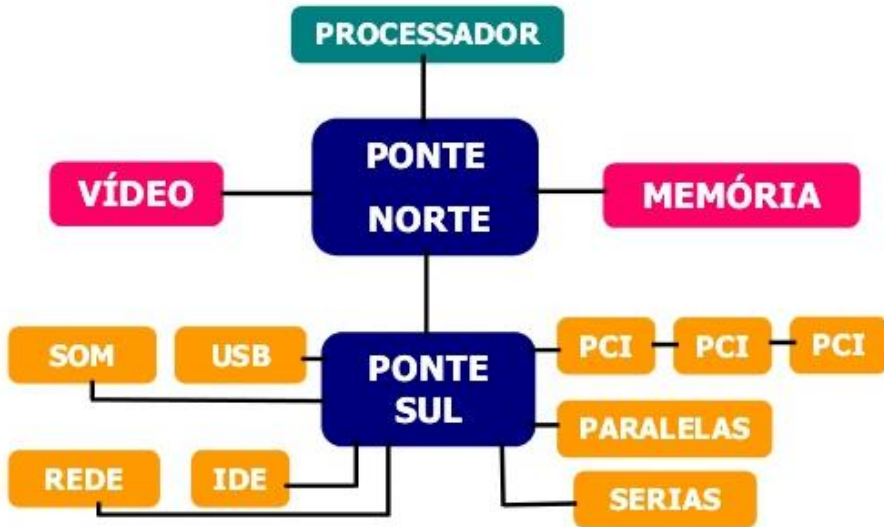
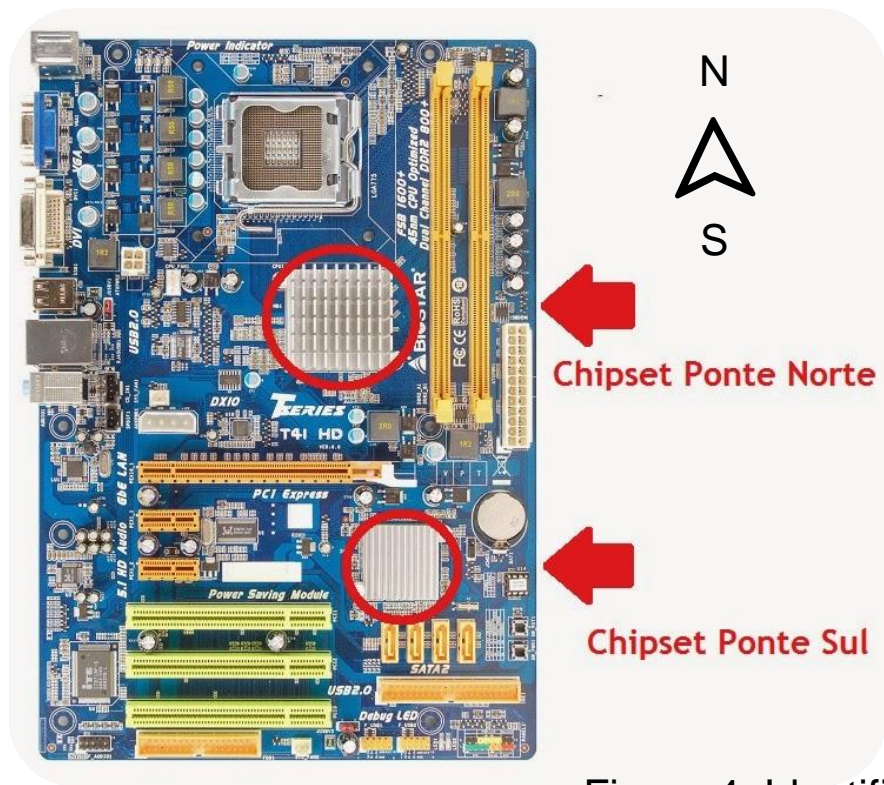


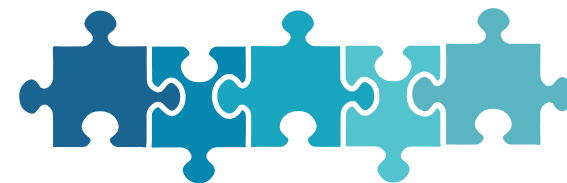
Figura 3. Exemplo de um organograma com a distribuição dos periféricos.

Figura 4. Identificação da ponte norte e sul da placa mãe.

---

## 7.2. ARQUITETURA DE COMPUTADORES

- Segundo Monteiro (2014), a organização de um computador, também conhecida como implementação é a parte da ciência da computação que estuda o comportamento dos componentes e periféricos. (grifo meu).
- Trata-se de um sistema complexo, pois se cada desenvolvedor de *software* desconhecesse as funcionalidades do *hardware*, nenhum código seria possível de ser escrito.
- Portanto, compreende-se que as conexões/ligações entre o *software* e o *hardware* são realizadas por códigos que compilados executam funções e tarefas que gerenciam os dispositivos físicos.
- É função do S.O gerenciar todas essas funcionalidades.



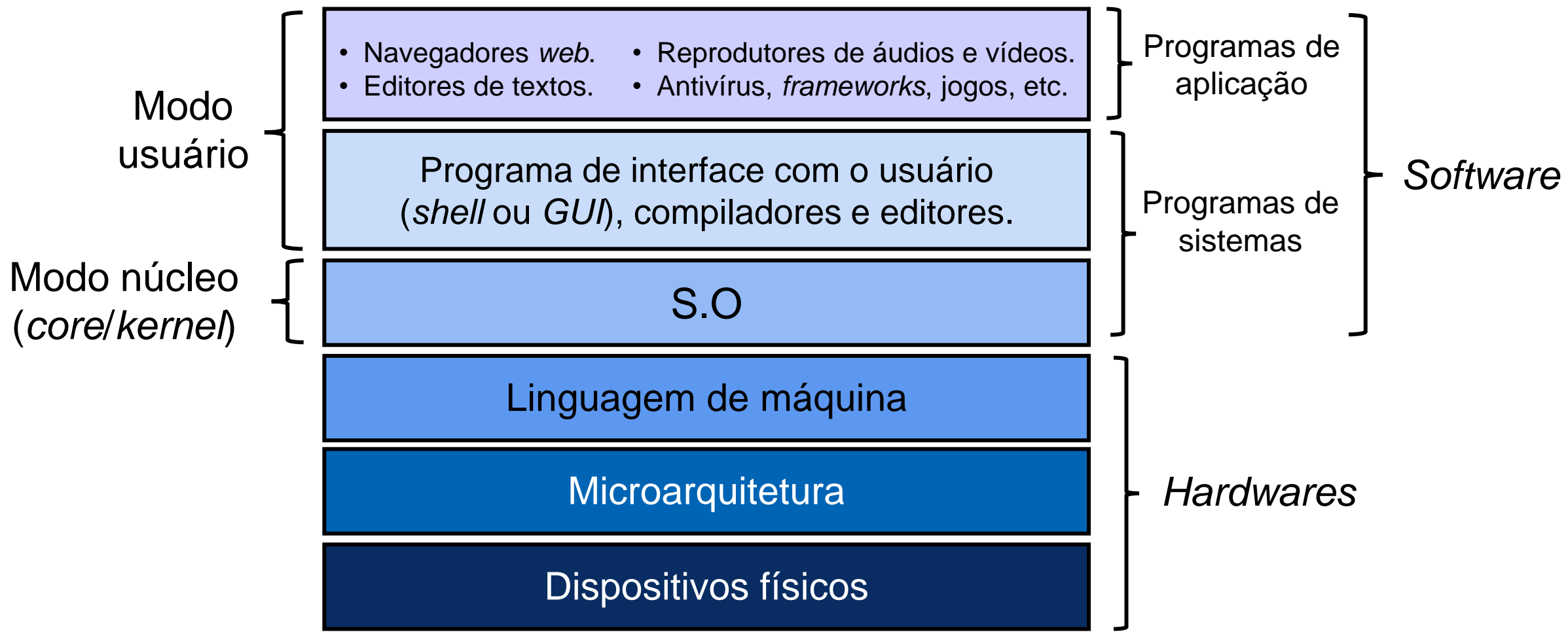


Figura 5. Arquitetura básica de um sistema computacional. (TANENBAUM, 2010 adaptado CAIXETA, 2019).



- Tomando a Figura 5, temos as seguintes definições para as camadas da arquitetura.

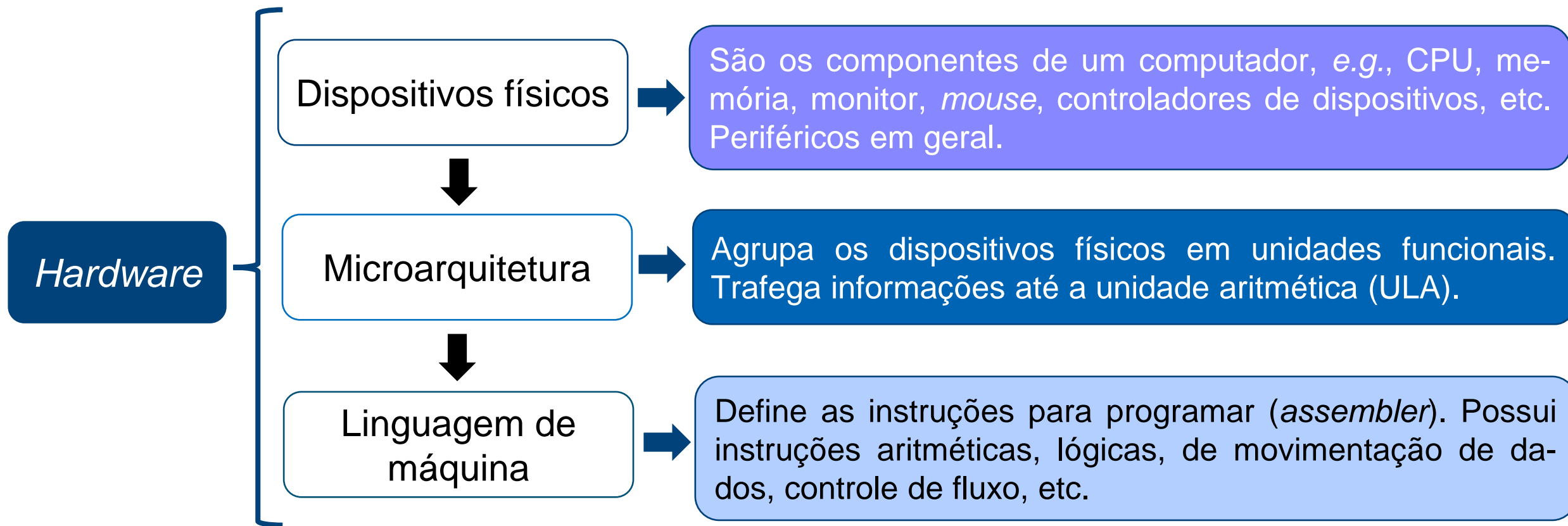


Figura 6. Estrutura física - *Hardware* de uma arquitetura básica.

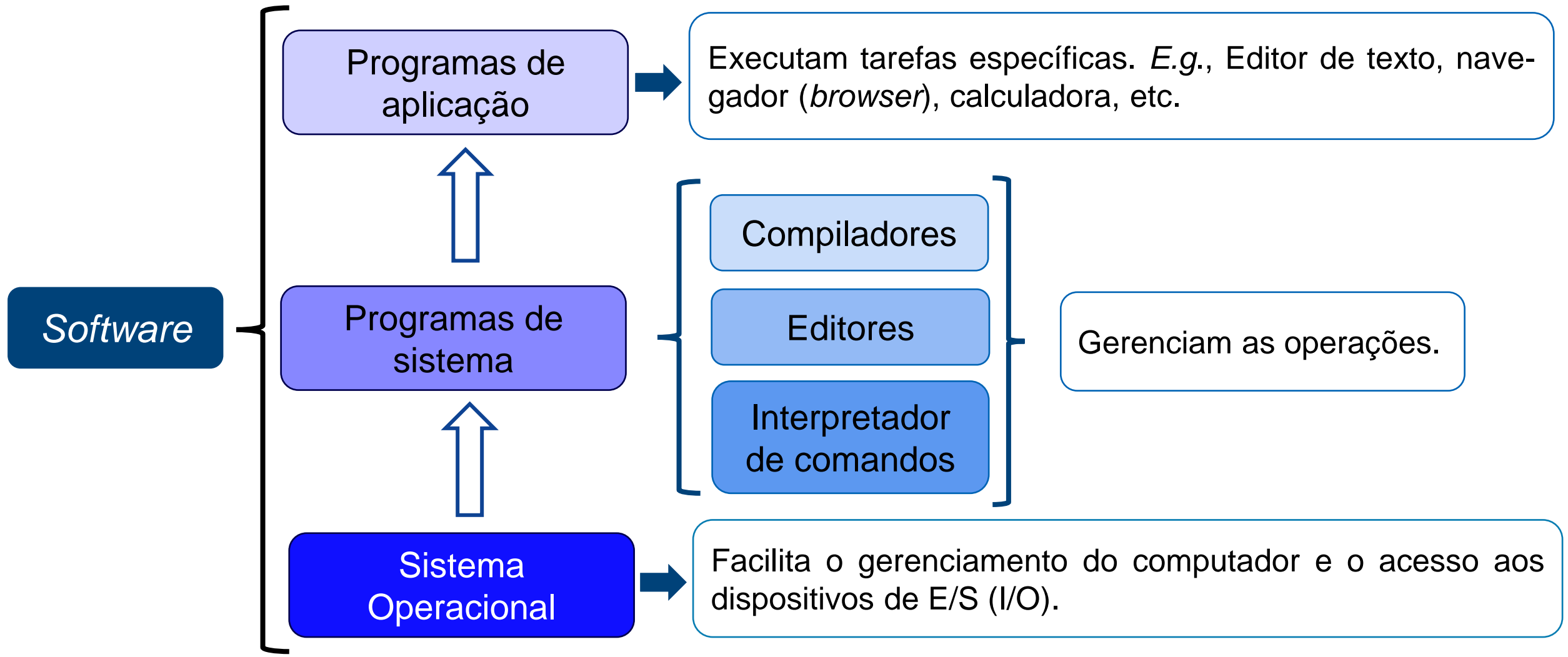
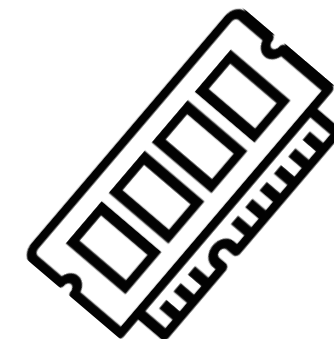
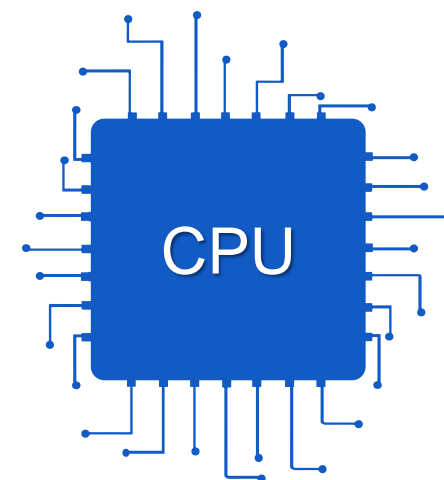


Figura 7. Estrutura lógica - *Software* que gerencia os elementos básicos da arquitetura dos computadores.

- Segundo Farias (2013), os circuitos de um computador que executam operações, tais como adição e subtração, são isolados em uma região chamada Unidade Central de Processamento - UCP (CPU - *Central Processing Unit*), ou processador.
- Os dados armazenados na memória principal do computador são transferidos através de barramentos que interligam todos os componentes/periféricos de *hardwares*.
- Já a comunicação com o mundo externo (usuários e sistema) se dá por meio dos dispositivos de entrada (*input*) e saída (*output*) (E/S - I/O).



- A comunicação interna - entre os dispositivos - se dá através dos controladores, que são microprogramas que enviam comandos através dos circuitos lógicos inseridos na placa principal do computador (placa mãe).

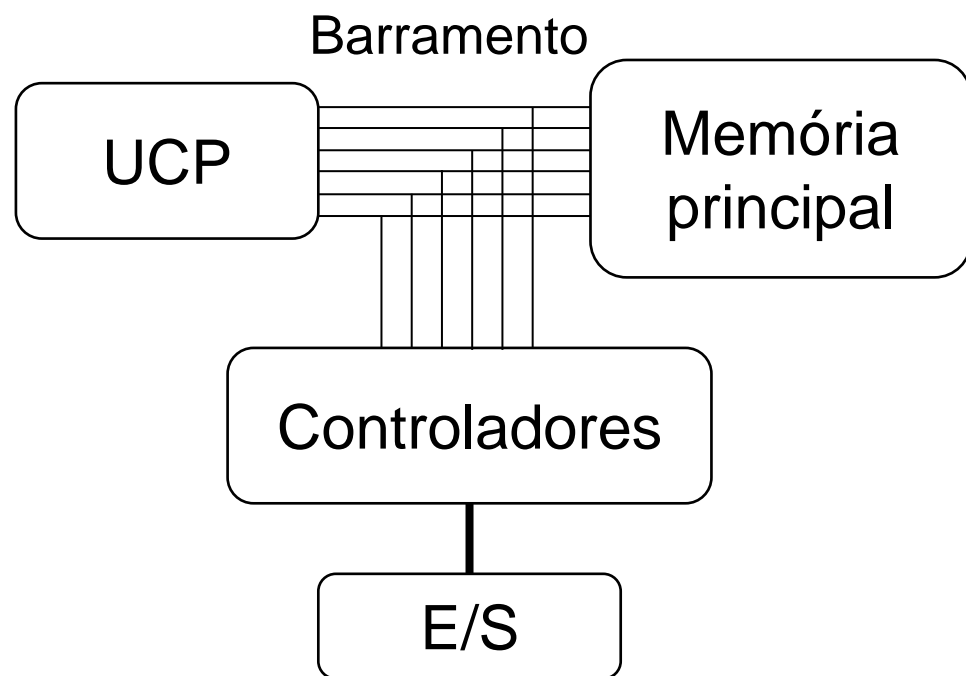


Figura 8. Conexão entre os principais sistemas/componentes de um computador.



- Comparando-se a evolução das arquiteturas, von Neumann em 1946 já trazia elementos para a criação de um sistema/*software* (operacional ou não) que gerenciavam as trocas de informações e a trafegabilidade de dados em nível de *hardware*.

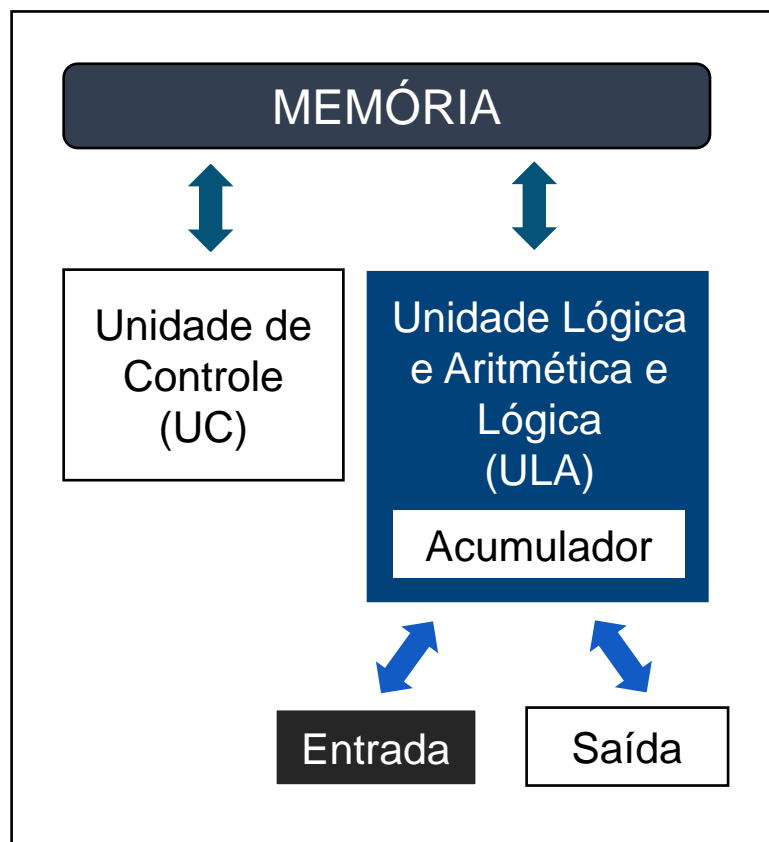
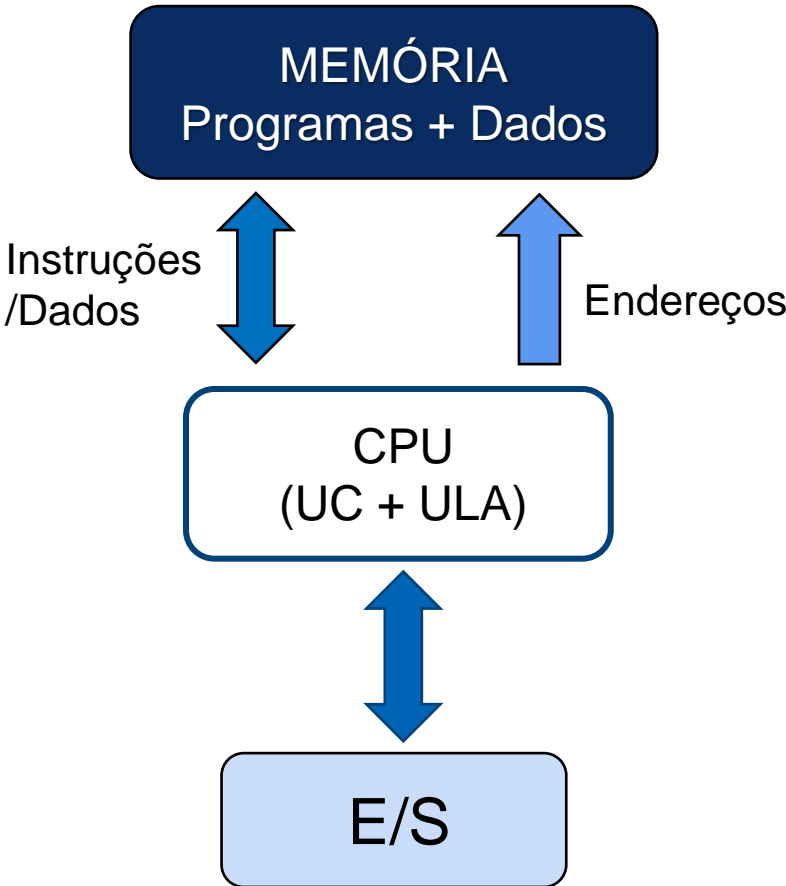
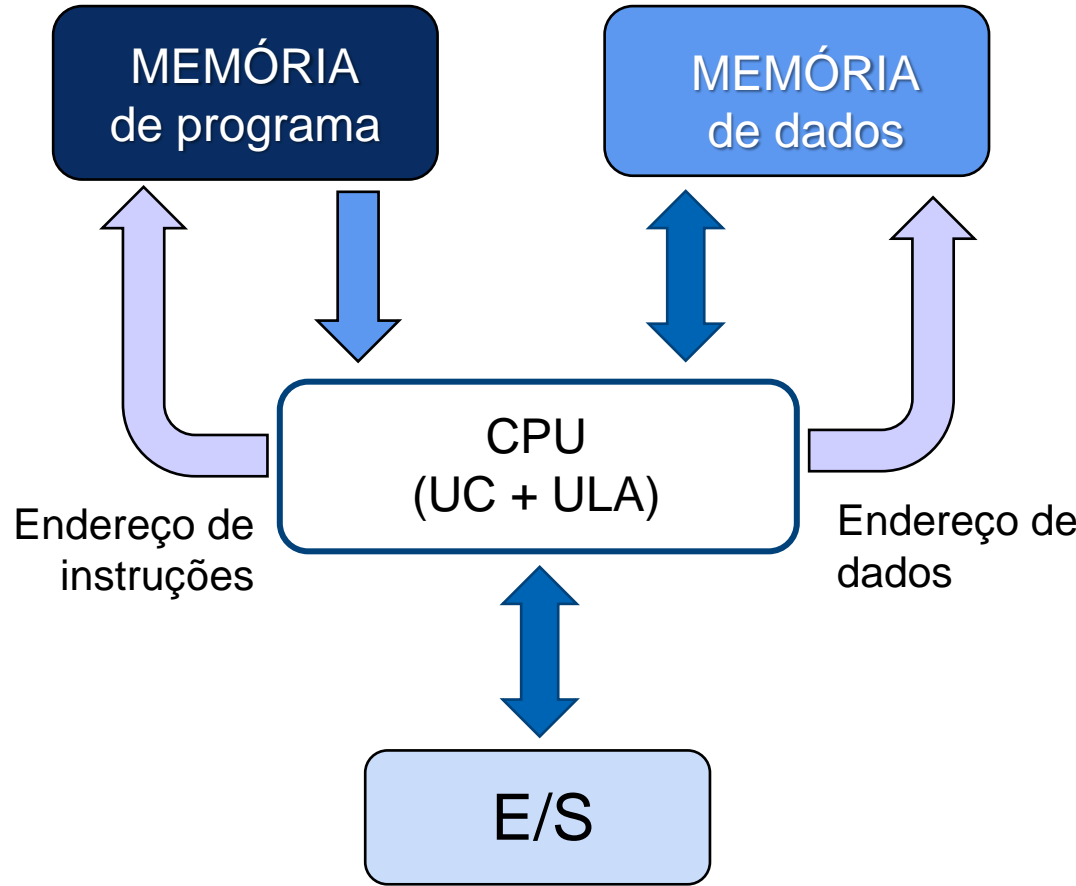


Figura 9. John von Neumann (1903-1957).

Figura 10. Proposta de arquitetura de computador de von Neumann (1946).



Arquitetura von Neumann  
(1946)



Arquitetura Harvard  
(1945)

Figura 11. Arquitetura de von Neumann *versus* Arquitetura de Harvard.

- Considerando ainda as referências apresentadas na Figura 5, a grande maioria dos computadores tem dois níveis operacionais, sendo eles modo núcleo e modo usuário. São nestes níveis que ocorrem todas as funcionalidades computacionais.

Tabela 1. Principais características dos modos operacionais.

Modo núcleo (ou supervisor)	Modo usuário
<ul style="list-style-type: none"><li>• Acesso completo ao <i>hardware</i>.</li><li>• Executa qualquer instrução de máquina.</li></ul>	<ul style="list-style-type: none"><li>• Operam as instruções de E/S (I/O), além de um subconjunto de instruções de máquinas disponíveis.</li></ul>

- É importante destacar a complexidade envolvida no gerenciamento dos diferentes níveis operacionais, pois, durante a execução dos processos, ocorrem simultaneamente trocas de informações e alocação de recursos.
- Nesse contexto, os sistemas operacionais desempenham um papel fundamental, sendo responsáveis pelo gerenciamento dessas abstrações em nível de processos.
- Por sua vez, os usuários finais interagem com o sistema por meio das abstrações fornecidas pelas interfaces, que podem ser *shell* (linha de comando e interpretador de comandos) ou GUI (*Graphical User Interface*/Interface Gráfica do Usuário).

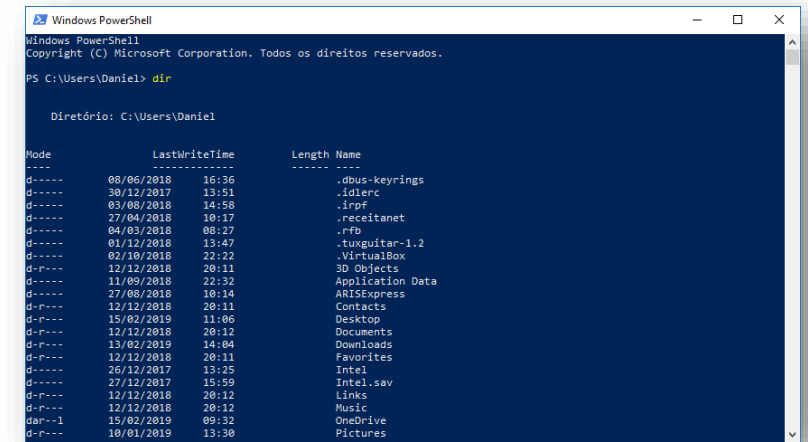
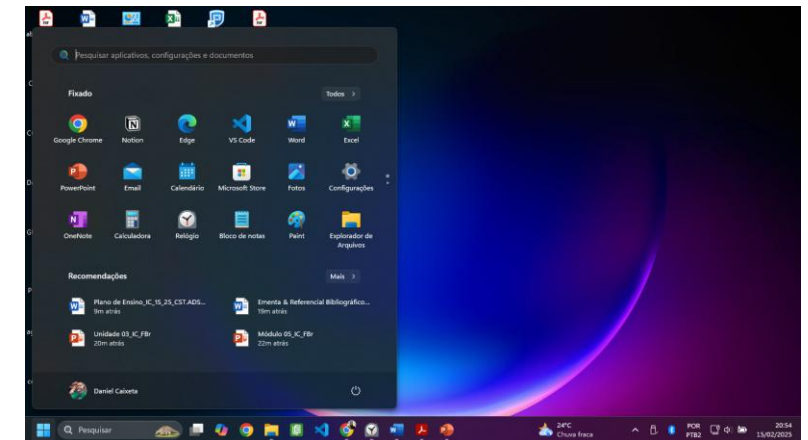
Figura 12. *Power Shell* (Windows 11).

Figura 13. GUI no Windows 11.



---

## 7.3. OS COMPONENTES BÁSICOS DA ARQUITETURA

### 7.3.1. MEMÓRIA

- É um componente de computador, dispositivos ou mídia de gravação que retêm os dados digitais usados pelo computador durante um intervalo de tempo. É um dos componentes fundamentais de qualquer computador moderno. (FERNANDEZ, 2015).
- Geralmente nos referimos a um dispositivo semicondutor conhecido como Memória de Acesso Aleatório (RAM - *Random Access Memory*), com alta capacidade, rapidez, mas provisório. Mas, também podemos chamar de dispositivo de armazenamento de alta capacidade, tais como discos óticos, discos rígidos magnéticos, etc.

## Algumas características das memórias

- Segundo Fernandez (2015) as memórias possuem algumas características:
  1. Quanto à volatilidade, podem ser:
    - Temporária:
      - Precisa de energia constante para manter as informações armazenadas.
      - São de rápido acesso.
      - Acesso randômico.
      - *E.g.*, Registradores, *cache*, RAM (armazenamento primário).
    - Permanente:
      - Retém as informações armazenadas, mesmo sem energia elétrica.
      - Armazena informações por um longo período de tempo.
      - *E.g.*, Disco rígido, memória *flash*, *pendrive*, SSD, etc. (armazenamento secundário).

## 2. Método de acesso, podem ser:

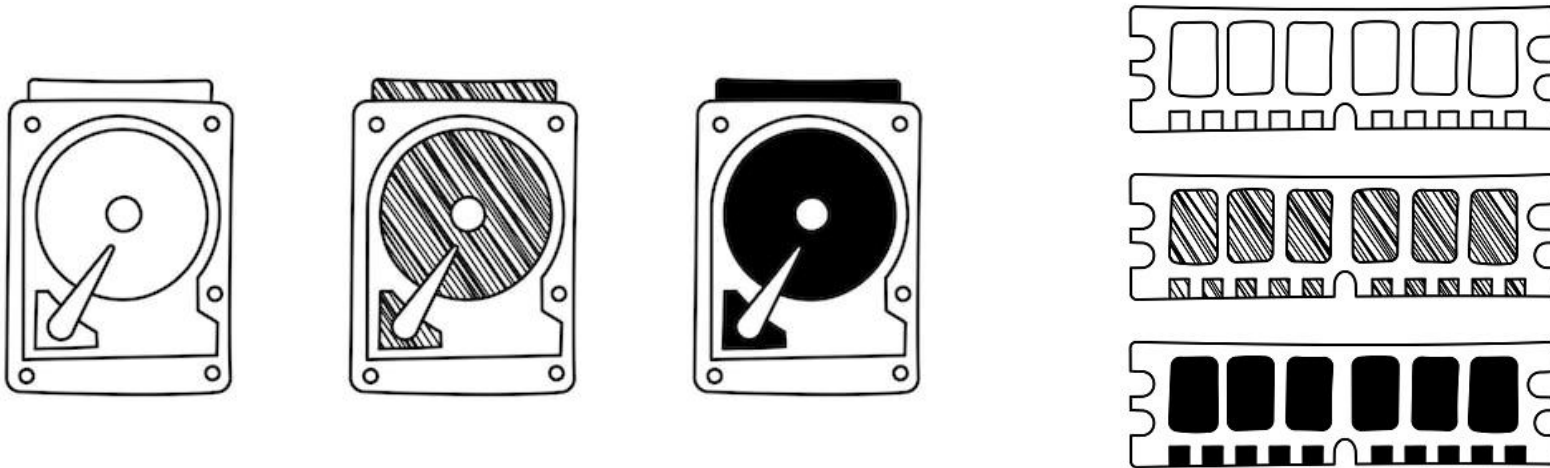
- Sequencial: Leitura do início ao ponto de acesso. *E.g.*, Fita magnética.
- Direta: Acessa o ponto direto e depende de referência anterior. *E.g.*, HD, SSD.
- Randômico: Acesso individual. *E.g.*, Memória RAM.
- Associativa: Acesso baseado na comparação de conteúdo. *E.g.*, Memória cache.

## 3. Quanto à hierarquia, podem ser:

- Registradores: localizado dentro da CPU.
- Memória Interna ou Principal: Memória RAM e *cache*.
- Memória Externa: Armazenamento de massa de dados/informações.

#### 4. O desempenho pode ser medido por alguns parâmetros, tais como:

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





# Hierarquia de memória

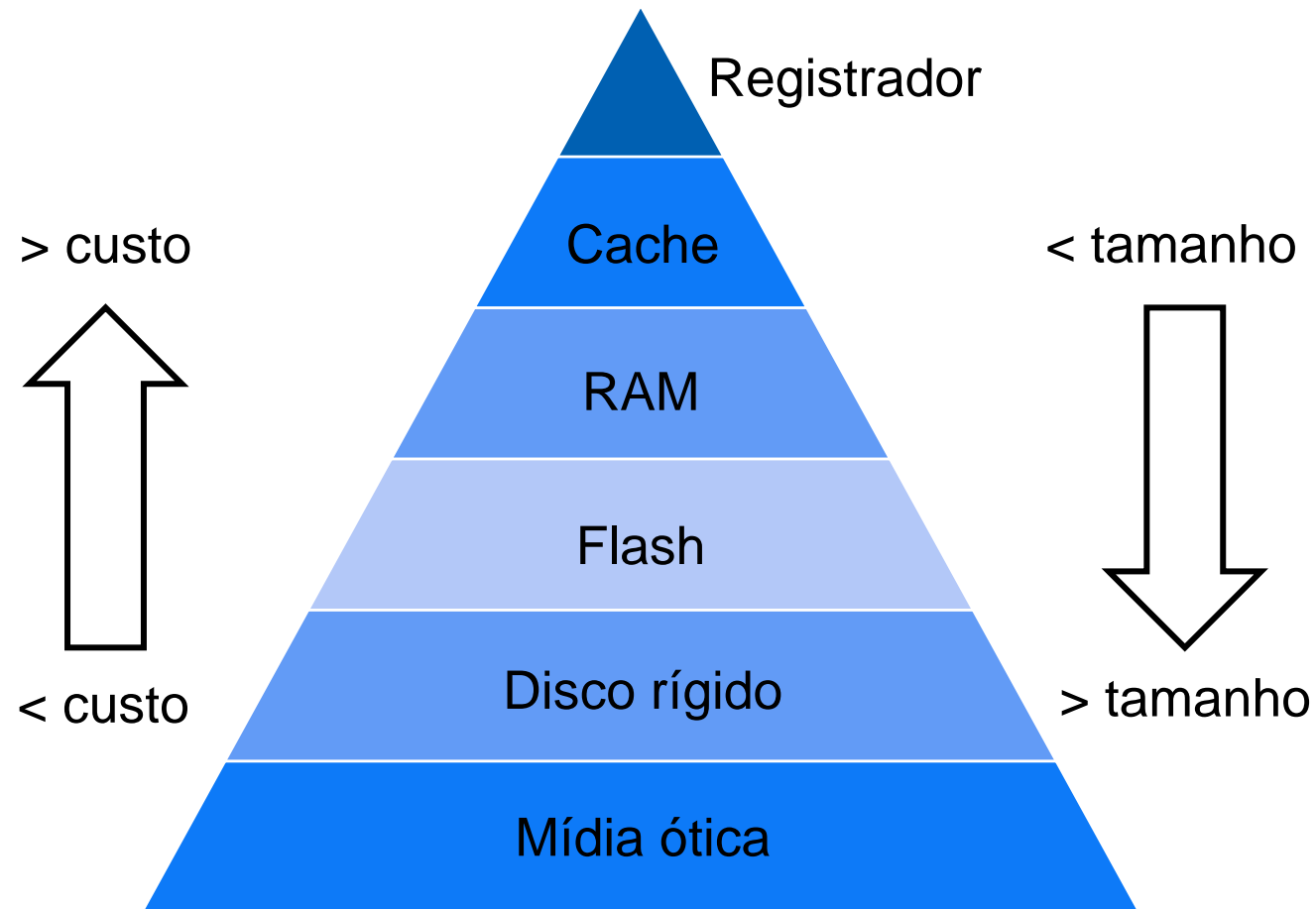


Figura 14. Hierarquia de memória.

---

## 7.3.2. MEMÓRIA PRINCIPAL

- A memória do computador consiste numa coleção de registradores numerados consecutivamente (endereçados), onde cada um possui um tamanho denominado de palavra que podem variar em 16, 32, 64 e 128 *bits*. Considerando que atualmente a palavra de 64 *bits* é a mais utilizada. (FARIAS, 2013).

16 *bits* = 65.535

32 *bits* = 4.294.967.295

64 *bits* = 18.446.744.073.709.551.615



Um espaço de endereçamento de 64 *bits* pode acessar qualquer palavra da memória em um intervalo de 0 a  $2^{64} - 1$ .

- O espaço de endereçamento pode ser dividido em regiões distintas usadas pelo S.O, dispositivos de E/S, programas de usuário e pilha do sistema.

Reservado para os dispositivos de E/S.

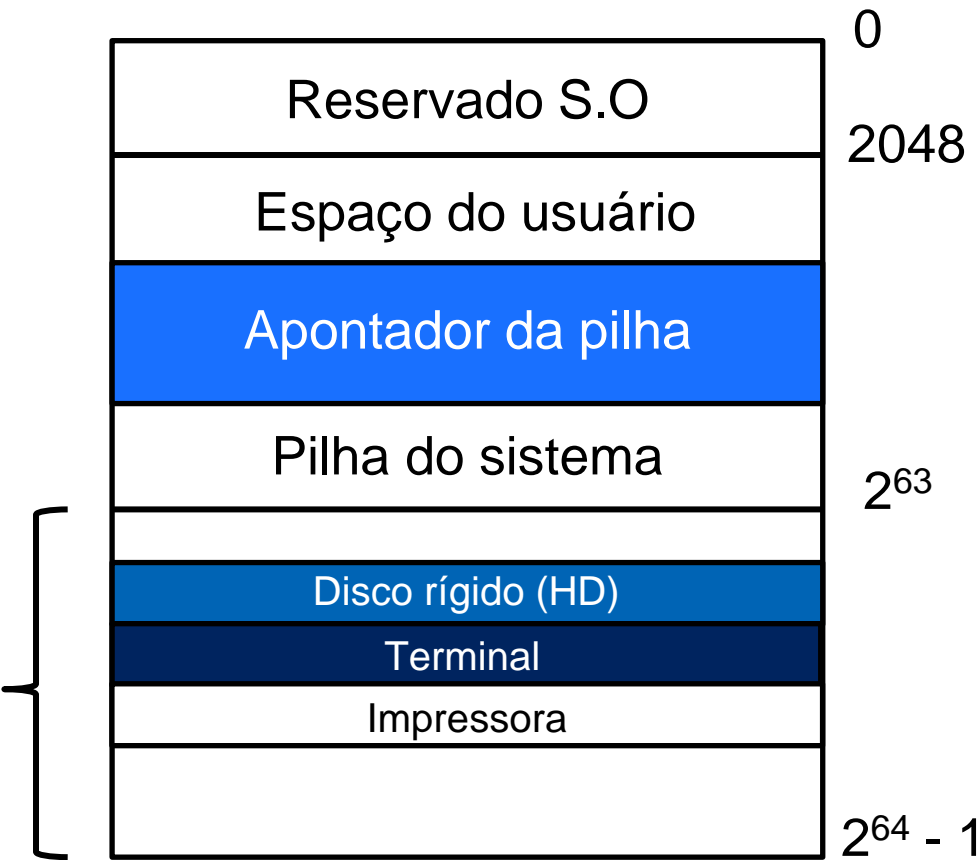


Figura 15. Mapa de memória.

- Uma memória pode ser vista como uma estrutura unidimensional, onde cada posição é uma célula, que possuem os mesmos números de *bits*. (FARIAS, 2013).

- É importante manter clara a distinção entre o que é endereço e o que é dado. (FARIAS, 2013).
- Uma palavra na memória, pode ter distintas representações dependendo do seu uso. Ela pode armazenar uma instrução contendo a operação e os operandos (dados de entrada) para a realização de uma específica operação, mas também pode armazenar o endereço de uma outra região da memória. (*ibidem*).
- Logo, o endereço é um apontador para uma posição de memória que contém dados, e estes são informações significativas para a realização de alguma atividade no computador, ou a representação de alguma informação. (*ibidem*).

---

### 7.3.3. TIPOS DE MEMÓRIAS

TIPOS	RAM – <i>Random Access Memory</i>	Estática e Dinâmica
	ROM – <i>Ready Only Memory</i>	PROM, EPROM, EEPROM, FLASH, etc.

#### a. Memória RAM Estática

- ✓ Os *bits* são armazenados em chaves ON/OFF (*flip-flops*).
- ✓ Os dados permanecem até o computador ser desligado, não precisando ser “re-frescadas” (do comando *refresh*).
- ✓ Possui construção complexa e ocupa uma grande área, por isso é mais cara. A vantagem é que são muito mais rápidas, sendo apropriadas para construção de memória *cache*.



## b. Memória RAM Dinâmica

- ✓ Os *bits* são armazenados em capacitores que se descarregam com o tempo, por isso é necessário “refrescar” o valor armazenado (*refresh*). (FERNANDEZ, 2015).
- ✓ Apesar da necessidade de um circuito especial para realizar o comando *refresh*, a construção dessa memória é simples, ocupa menos espaço e é barata, permitindo a construção de módulos grandes de memória. No entanto, elas são mais lentas que a memória estáticas. (*ibidem*).
- ✓ O circuito *refresh* é geralmente incluso no próprio *chip* e consiste em reescrever os dados para regenerar as informações. Por essa razão, há uma queda do desempenho do sistema. (*ibidem*).

c. Memória ROM

- ✓ As memórias ROM tem armazenamento permanente e são usadas para guardar microprogramas e BIOS - *Basic I/O System* - Sistema básico de um computador. (FERNANDEZ, 2015). Podem ser:

TIPOS		CARACTERÍSTICAS	
FIRMWARE		o Bem cara e é usado em lotes pequenos.	
PROM		o Programadas apenas uma vez.	
EPROM		o Programadas eletronicamente e apagadas por luz UV (ultravioleta).	
EEPROM		o Programadas e apagadas eletronicamente – um <i>byte</i> por vez.	
FLASH		o Programadas e apagadas eletronicamente (apaga toda memória ou um bloco).	

### 7.3.4. MEMÓRIA CACHE

- Trata-se de uma pequena quantidade de memória rápida colocada entre o CPU e a memória principal. É usada para armazenar e acessar de forma mais rápida dados de uso de maior frequência.
- Eventualmente, poderia ter sido colocada dentro do *chip* da CPU, mas como se tratava de um bloco grande da memória principal, resolveu-se isolar da CPU para otimizar o desempenho.

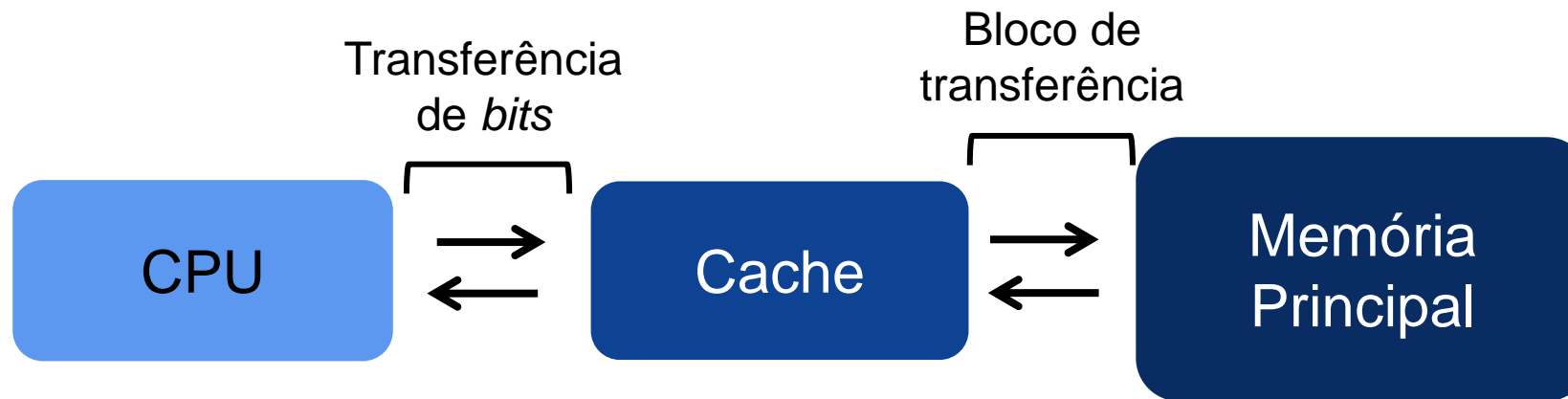


Figura 16. Diagrama de uma arquitetura de um computador com memória cache.

- Segundo Monteiro (2014), os estudos da memória *cache* surgiram inicialmente devido ao *gap* de velocidade entre a memória principal e o processador.
- Devido à necessidade de redução desse *gap*, os pesquisadores realizaram vários estudos de comportamento de programas/*softwares*. Outro estudo importante foi o da observância do Princípio da Localidade.
- Parte do problema de limitação de desempenho dos processadores, refere-se à diferença de velocidade entre o ciclo de tempo do processador e o ciclo de tempo da memória principal, ou seja, a memória principal sempre transfere dados para o processador em velocidades inferiores ao seu ritmo, e isso acarreta em muitas vezes no que chamamos de *wait state* (estado de espera).

- E se todos os circuitos do processador e da memória *cache* fossem fabricados com a mesma tecnologia?
- Para Monteiro (2014), hoje não estaríamos estudando a memória *cache*.
- Esse problema tornou-se permanente, e uma solução mais eficaz é considerada de alto custo, o que tornaria inviável a comercialização de computadores comuns.
- Ao analisar a estrutura e a execução de diversos programas/*softwares* (comerciais, científicos, acadêmicos, etc.), os pesquisadores verificaram que os programas em média são executados de forma semelhante, i.e., em blocos de instruções sequenciais, sendo que alguns são executados mais de uma vez em curtos intervalos de tempo. (*ibidem*).



- Esta forma padronizada de execução, deu-se o nome de Princípio da Localidade, que trata-se de um fenômeno relacionado com o modo pelo qual os programas em média são escritos pelo programador e executado pelo processador. Esse princípio deriva-se em duas modalidades:

LOCALIDADE	TEMPORAL	<ul style="list-style-type: none"><li>• Os programas não são executados de modo que a memória RAM seja acessada randomicamente, mas sim acessando uma palavra na memória, onde há uma grande probabilidade de que em breve essa mesma palavra seja acessada novamente.</li></ul>
	ESPACIAL	<ul style="list-style-type: none"><li>• É quando o programa ao acessar uma palavra na memória, exista uma boa probabilidade de que o acesso seguinte seja uma palavra subsequente ou de endereço adjacente àquela que ele acabou de acessar.</li></ul>

(MONTEIRO, 2014).

- Portanto, os programas tendem a acessar frequentemente os mesmos endereços em curtos espaços de tempo, em forma de *loop*. Por isso que os estudos do princípio da localidade foram importantes para diminuir esse *gap* durante as execuções dos processos entre a memória principal e o processador. Daí, conclui-se que o problema não foi resolvido, mas foi amenizado, com a implementação da memória *cache*.

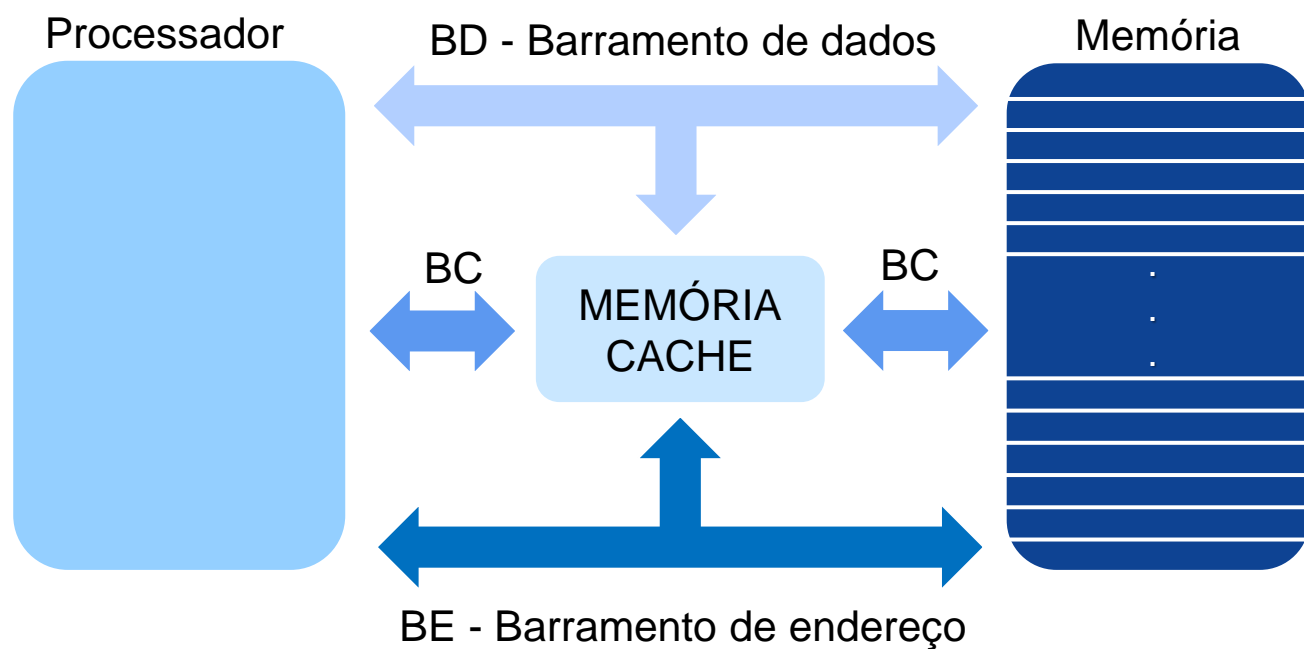


Figura 17. Exemplo de conexão e funcionamento do sistema processador, cache e MP.

- Então para que um sistema tenha um bom desempenho, é necessário medir o máximo de acertos de processos (*hits*) e um mínimo de faltas (*misses*), assim podemos medir um valor de eficiência da memória *cache* determinando a relação entre os acertos e o total de acessos.

$$E_c = \frac{\text{Acertos (hits)}}{\text{Total de acessos}} \times 100$$

Onde,  $E_c$  = Eficiência da memória *cache*

Exemplo 1. Um determinado sistema de computação possui uma memória *cache*, memória principal e processador. Em operações normais, obtêm-se 96 acertos para cada 100 acessos do processador às memórias. Qual deve ser a eficiência do sistema *cache*/memória principal?

- Então, como funciona a memória *cache*:
  1. A CPU solicita uma posição de memória.
  2. Verifica se essa posição está no *cache*.
  3. Se está, pega bloco de informações.
  4. Se não, requisita bloco da memória principal.
  5. Inclui uma *tag* para identificar bloco de memória.
  6. CPU lê a memória desejada.
  7. Muito provavelmente, a próxima posição solicitada pela CPU deverá estar no *cache*.

## MAPEAMENTO DA MEMÓRIA CACHE

- É realizado através do que chamamos de Mapeamento Direto (*Direct Mapped*), ou seja, cada palavra deve ser armazenada em um lugar específico na memória *cache*, o qual depende do seu endereço na memória principal.
- Esse endereçamento é dividido em duas partes, sendo:
  - Índice - É usado como endereço na memória *cache*. Indica a posição onde se encontra a palavra.
  - *Tag* - É usado para conferir se a palavra que está na memória *cache* é a que está sendo procurada, uma vez que endereços diferentes, com o mesmo índice serão mapeados sempre para a mesma posição da *cache*.

## MEMÓRIA VIRTUAL *VERSUS* MEMÓRIA CACHE

- A primeira é usada para ocultar a informação da memória física real do sistema. [...]. Portanto, cria uma ilusão de que um usuário tem um ou mais espaços de endereços contíguos que começam com endereço zero.
- Já a segunda armazena os dados desses endereços da memória principal que são usados várias vezes, estando disponível em uma parte reservada da memória principal ou existir como um dispositivo de armazenamento de alta velocidade independente.
- A principal diferença é que a memória virtual é uma abstração da memória principal, enquanto que a memória *cache* está disponível no computador.

### 7.3.5. UNIDADE CENTRAL DE PROCESSAMENTO (UCP)

- É composta por duas partes principais: a Unidade Lógica Aritmética (ULA), formada por circuitos que manipulam os dados através de operações binárias - AND, OR e NOT.
- E a Unidade de Controle (UC), cujos circuitos são responsáveis por coordenar as operações da UCP.

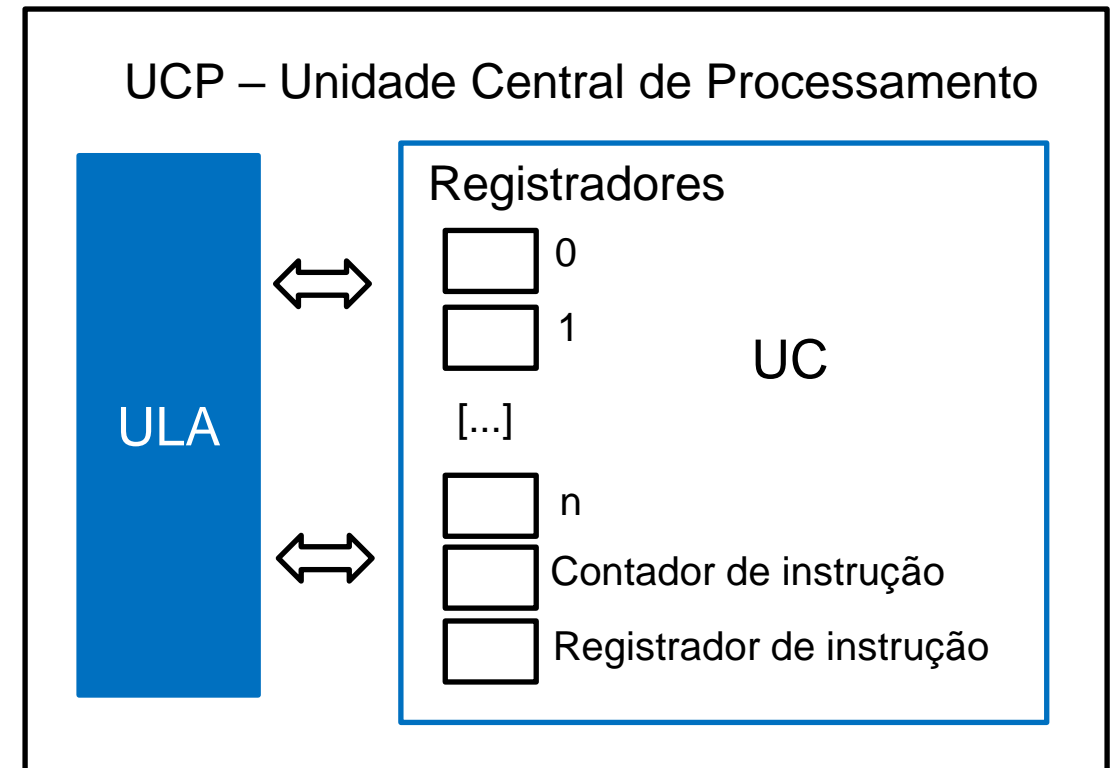


Figura 18. Componentes lógicos da UCP.



- Para o armazenamento e a comunicação entre estas duas unidades, a UCP contém circuitos chamados de registradores, que se assemelham às células de armazenamento da memória principal. (FARIAS, 2013).
- Já os dados a serem manipulados pela ULA tem origem na memória principal, sendo de responsabilidade da UC transferir estes dados aos registradores, informar à ULA sobre quais registradores estão os dados de entrada, ativar o circuito da operação apropriada e informar em que registrador deve guardar o resultado da operação. (*ibidem*).
- A transferência desta informação oriunda da memória principal se dá através do barramento que é responsável por transmitir padrões de *bits* entre a UCP, os dispositivos de E/S e a memória principal.
- Portanto, executar uma simples operação de soma é mais complexo que apenas somar estes números. (*ibidem*).

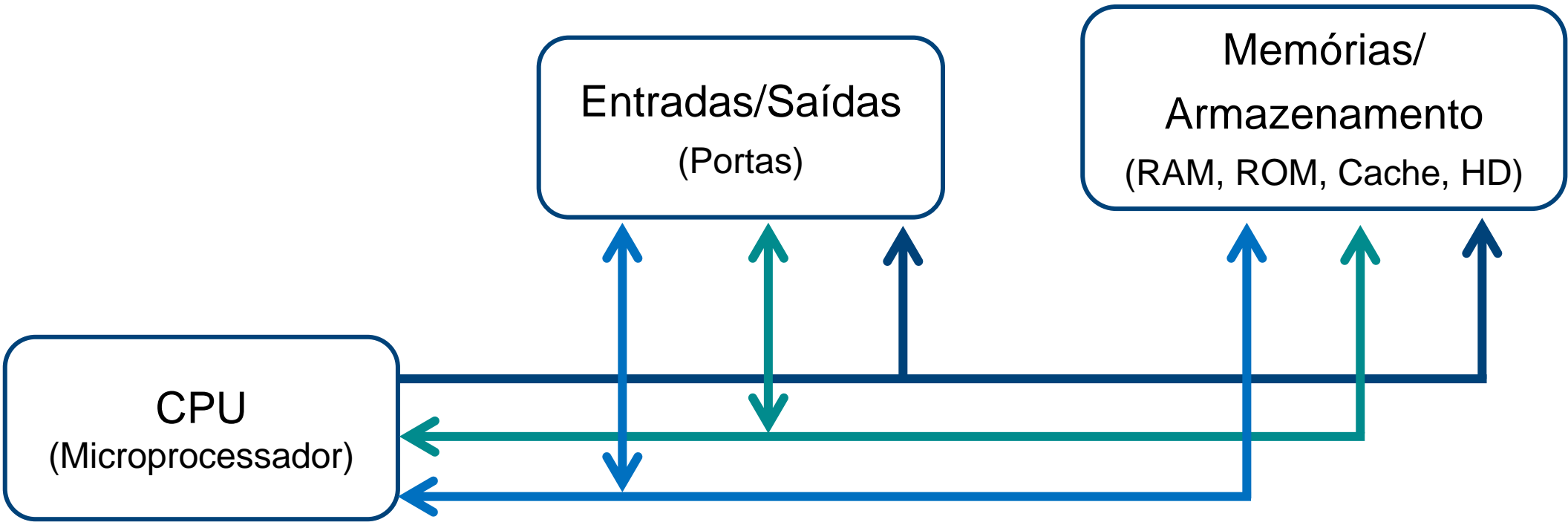





Figura 19. Microarquitetura de um sistema computacional.

Legenda

-  Barramento de controle (*Control bus*).
-  Barramento de dados (*Data bus*).
-  Barramento de endereço (*Address bus*).

---

### 7.3.6. UNIDADES DE ENTRADA E SAÍDA (E/S OU I/O)

- Segundo Farias (2013), Entrada/Saída (E/S ou I/O) compreende todas as maneiras como o computador se comunica com os usuários e outras máquinas ou dispositivos.

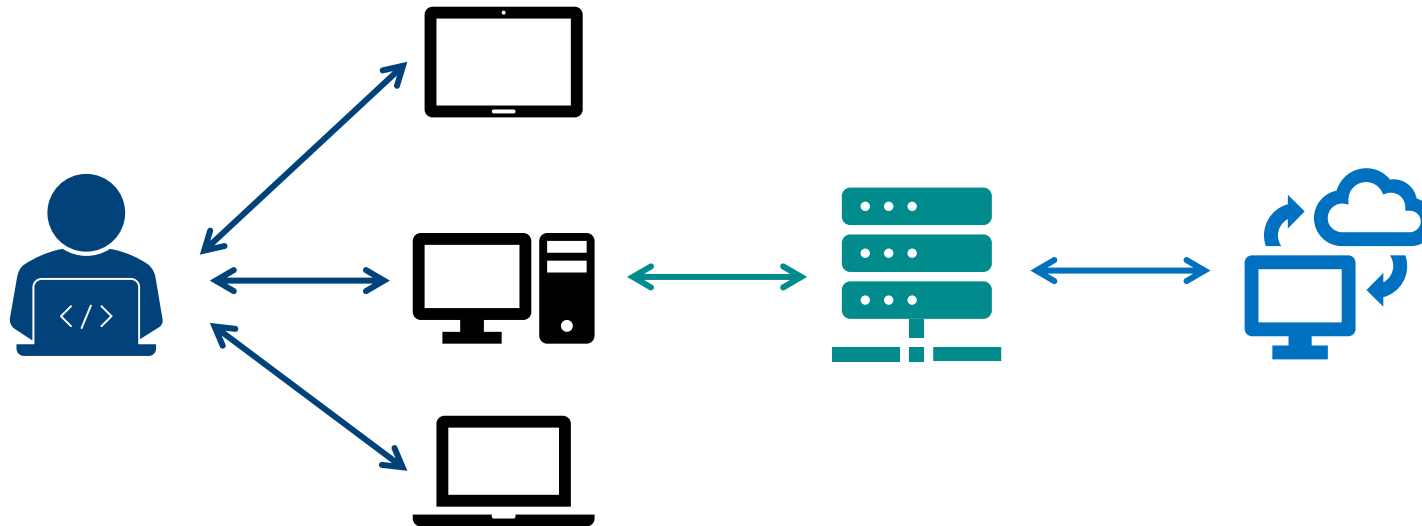


Figura 20. Comunicação entre as interfaces ser humano-máquina.

- Os dispositivos de entrada aceitam dados e instruções do usuário, os dispositivos de saída retornam os dados processados.



Figura 21. Etapas básicas de um processamento de dados. (CAIXETA, 2021 *apud* MONTEIRO, 2014).

- Os dispositivos de saída mais comuns são a tela de vídeo (monitor) e impressora.
- Já os de entrada mais conhecidos são teclado, *mouse*, mesa digitalizadora, etc. Os sistemas de multimídia possuem alto-falante como saída e microfone como entrada adicional.

- Os dispositivos de E/S (I/O) trabalham com a memória do computador do seguinte modo (FARIAS, 2013):
  1. Os dados captados pelos dispositivos de entrada são representados em pulsos elétricos e transmitidos ao computador, ali estes pulsos são convertidos em dados binários e armazenados na memória do computador.
  2. No caminho inverso, a informação binária é transformada em pulso elétrico e encaminhada para o dispositivo de saída especialista para tratá-lo e gerar uma saída ao usuário.

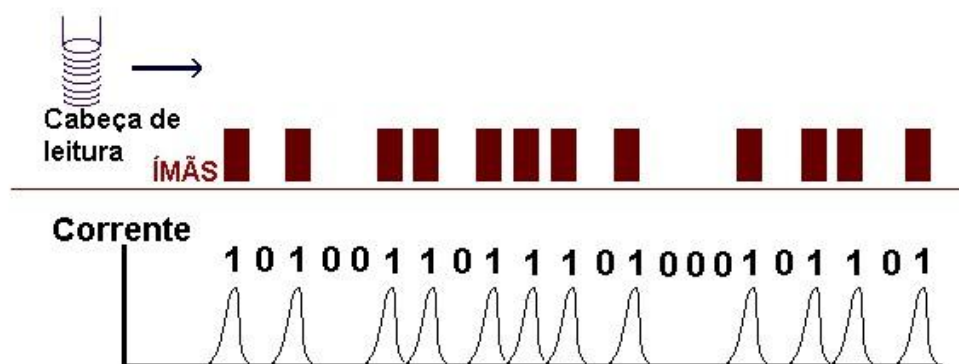


Figura 22. Exemplo de um mecanismo transmissão de pulso elétrico em binário.

- Um dispositivo especial de E/S de um computador é o disco rígido (*Hard Disk*), nele são armazenados todos os dados que devem persistir num sistema computacional, mesmo na ausência de energia.
- Todos os programas que não estão em execução se encontram no disco, seu único problema é o tempo excessivo para a recuperação e escrita de uma informação, havendo assim a necessidade de se trabalhar com a memória volátil (memória principal), mais rápida, porém mais cara.



- Hoje temos os dispositivos SSD (*Solid State Drive*) que é uma espécie de híbrido (memória RAM + Disco) que possui alto desempenho em relação aos dispositivos mais “antigos”.

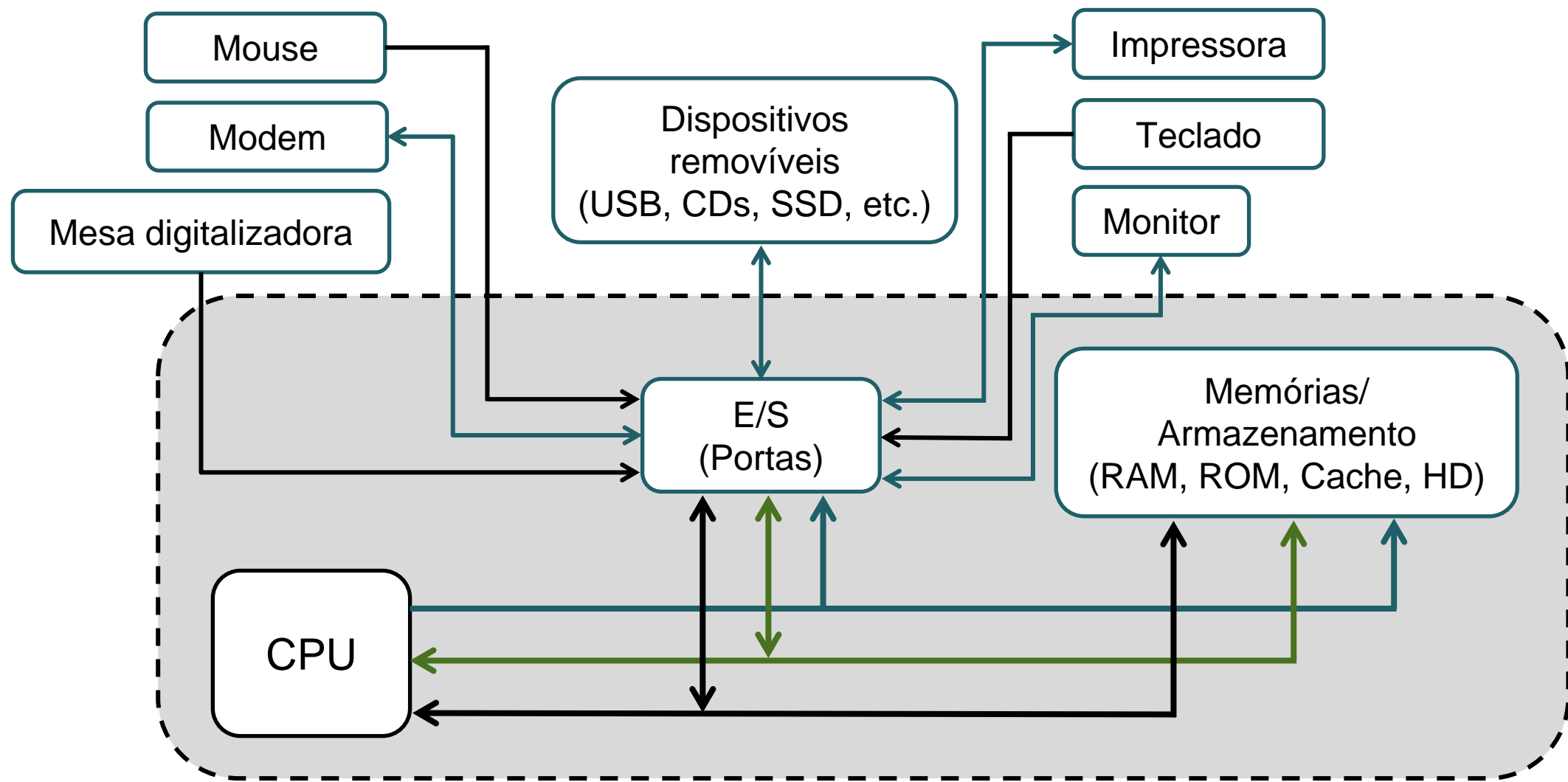


Figura 23. Sistema computacional (arquitetura do *hardware*) com mais periféricos.



---

### 7.3.7. O BARRAMENTO

- O objetivo do barramento é reduzir o número de interconexões entre a UCP e seus subsistemas. (FARIAS, 2013).
- Em lugar de mantermos um caminho de comunicação entre a memória e cada um dos dispositivos de entrada e saída, a UCP é interconectada com os mesmos via barramento de sistema compartilhado.
- As interconexões dos componentes ao barramento é de responsabilidade da UCP, que gera/cria todos os registros de endereçamentos, e também na memória.



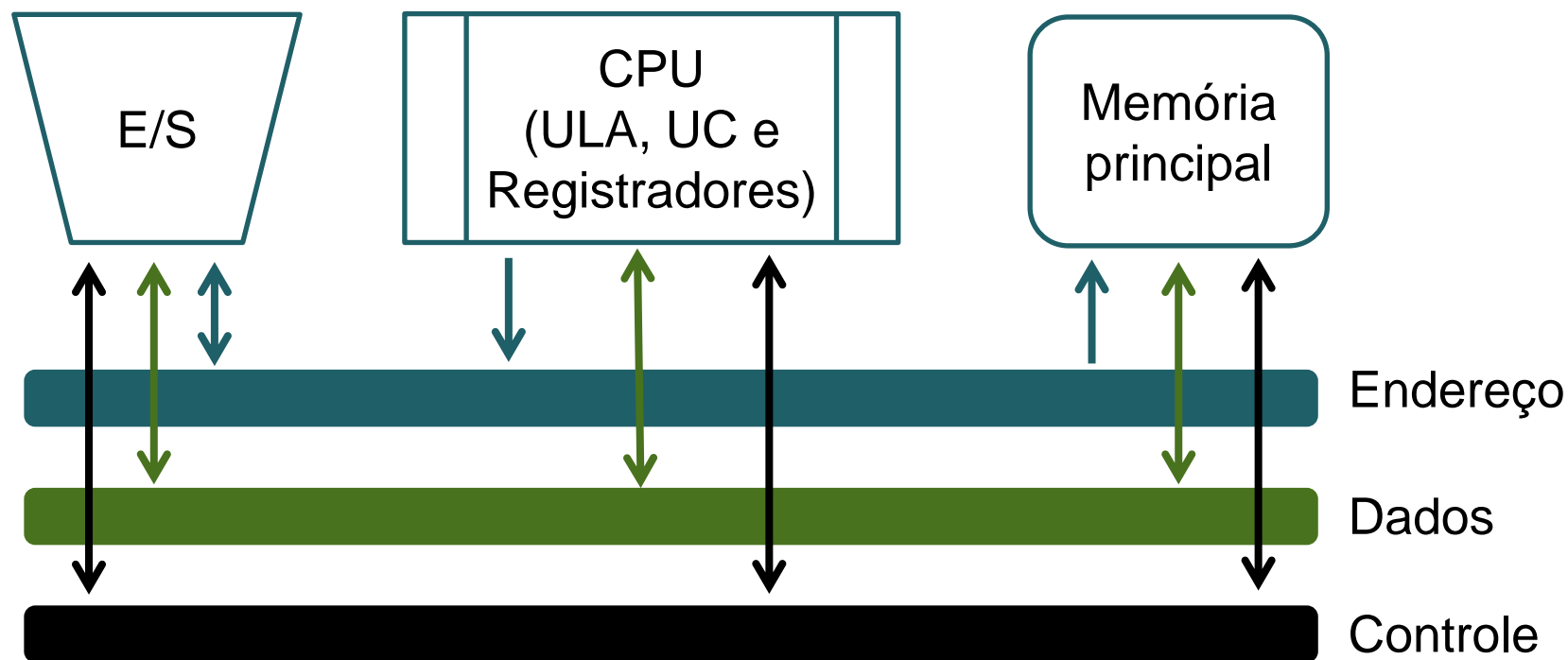


Figura 24. Exemplo de barramento.

- Durante a execução de um programa, cada instrução é levada até a ULA e depois enviado para a memória, e a partir daí as instruções são executadas uma de cada vez, sendo os resultados apresentados nos dispositivos de saída, *e.g.*, monitor.

---

### 7.3.8. REGISTRADORES E *DATAPATHS*

- Registradores são pequenas unidades capazes de armazenar informações - dados, endereços, instruções, etc., sendo usado geralmente em operações lógica-aritméticas.
- Possuem rápido acesso e custo elevado, por isso a existência de poucos registradores no processador, além do mais seu armazenamento é volátil.
- A grande maioria dos processadores têm registradores para:
  - i. PC (*Program Counter*): Endereço da próxima instrução a ser executada.
  - ii. IR (*Instruction Register*): Próxima instrução a ser executada.
  - iii. SP (*Stack Pointer*): Endereço da informação que está no topo da pilha de execução.

- Já o *Datapath* executa as instruções e operações enviadas pela UC. É composto por:
  - ✓ Unidade Lógico Aritmética.
  - ✓ Conjunto de registradores.
- Na ULA realiza-se as operações lógicas e aritméticas: soma, subtração, comparação, etc. *E.g.*:
  - A UC busca na memória a instrução Executar soma: add, A, B.
  - A ULA realiza soma dos dois operandos, produzindo um resultado.

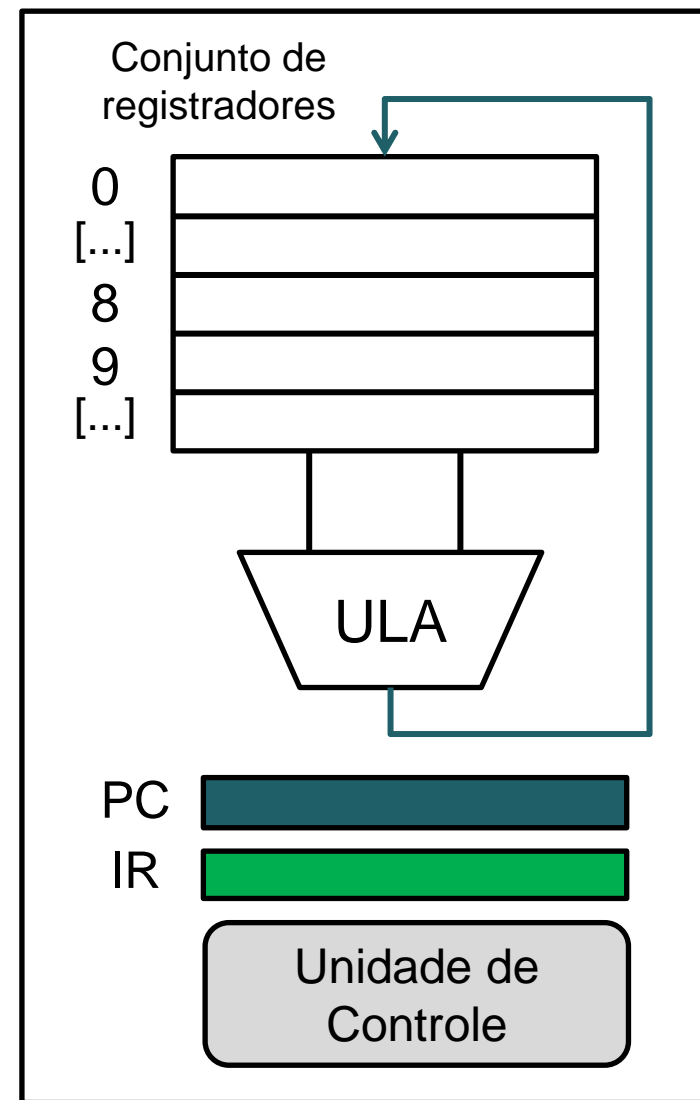


Figura 25. Parte interna de um processador.

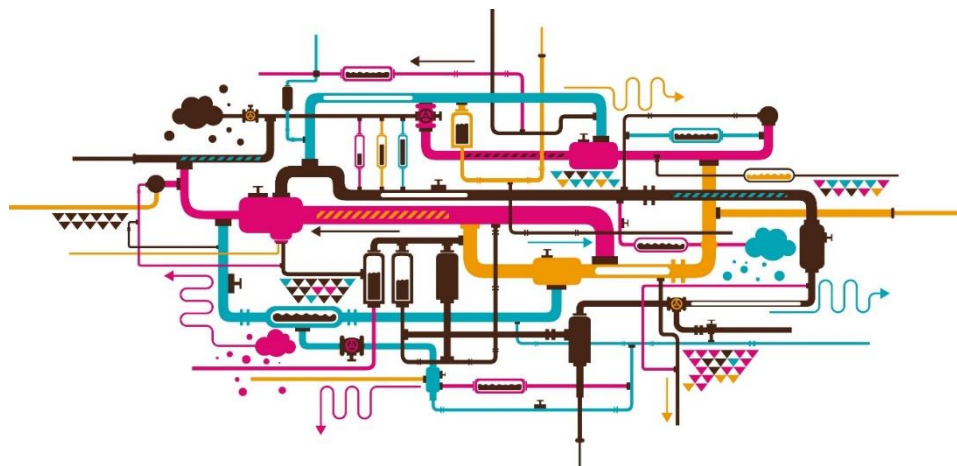
---

### 7.3.9. PIPELINE

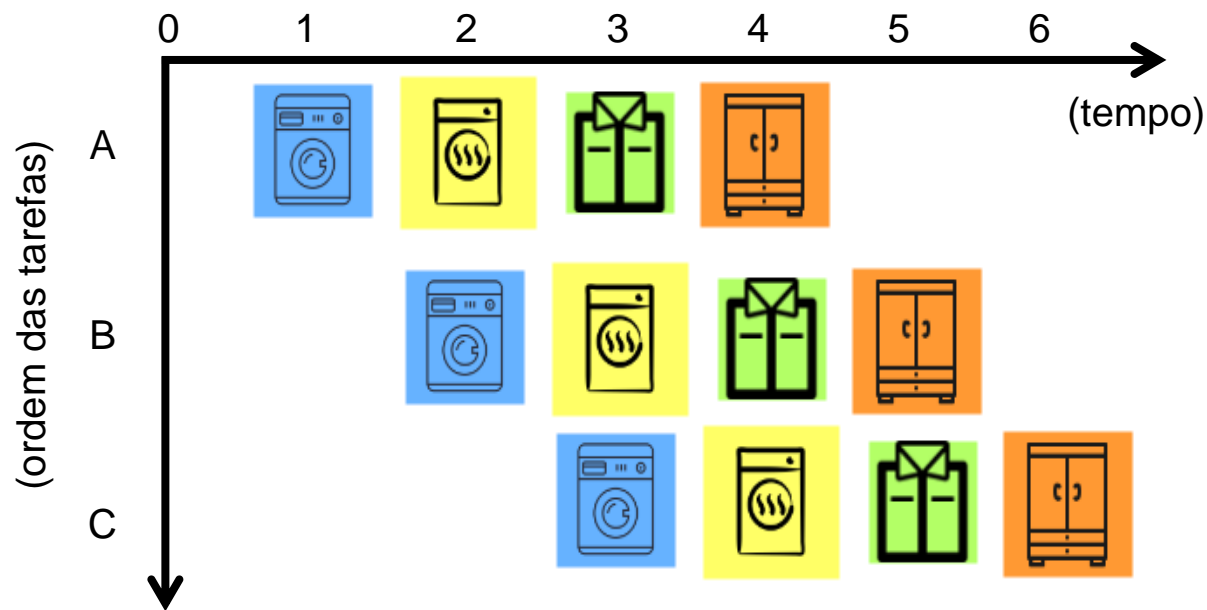
- Existem algumas definições para o conceito de *pipeline*. É claro que o nosso foco está na computação. Então, qual é o significado?
- É um termo inglês que podemos traduzir por “tubagem” ou “canalização”.
- Trata-se de uma arquitetura baseada no funcionamento dos oleodutos, que estão segmentados em diversos ramais e que dispõem de bombas para impulsar, em cada trecho, o avance do gás.
- Embora não faça parte dos dicionários de língua portuguesa, esse conceito é utilizado por nós para fazer referência a uma Arquitetura em Computadores.



- É usado em microprocessadores, *software*, etc.
- Na verdade este processo, em um sistema computacional, cria canalizações virtuais para segmentar os dados e, deste modo, incrementar o rendimento de um sistema digital.
- A segmentação dos cálculos de transmissão, permite melhorar a frequência de trabalho. Este tipo de fluxo de dados implica que a saída de uma fase é uma entrada de outra. Pois, observa-se que as fases encadeiam-se como uma canalização, agilizando o fluxo através da tubulação ou *pipeline*.



- Pantuza (2020), define *Pipeline* como uma técnica que permite aos processadores executarem tarefas diferentes ao mesmo tempo, respeitando a ordem das instruções que chegam ao processador. Esse método aumenta o desempenho e reduz o tempo de execução global de tarefas. Vejamos os próximos exemplos:



- A figura 26, apresenta um exemplo de execução com *pipeline* de 3 tarefas (A, B e C) com 4 atividades – lavar, secar, dobrar e guardar as roupas.
- Observa-se as sobreposições de atividades durante a execução do processo.

Figura 26. Exemplo de *pipeline* na execução de uma tarefa diária em casa.



- Para compreender porque o *pipeline* faz com que os processadores trabalhem mais rápido, vamos ver a execução dessas mesmas tarefas sem o seu uso.
- Exemplo: Considere que cada uma das 4 atividades (lavar, secar, dobrar e guardar) da tarefa demore 1 minuto cada.

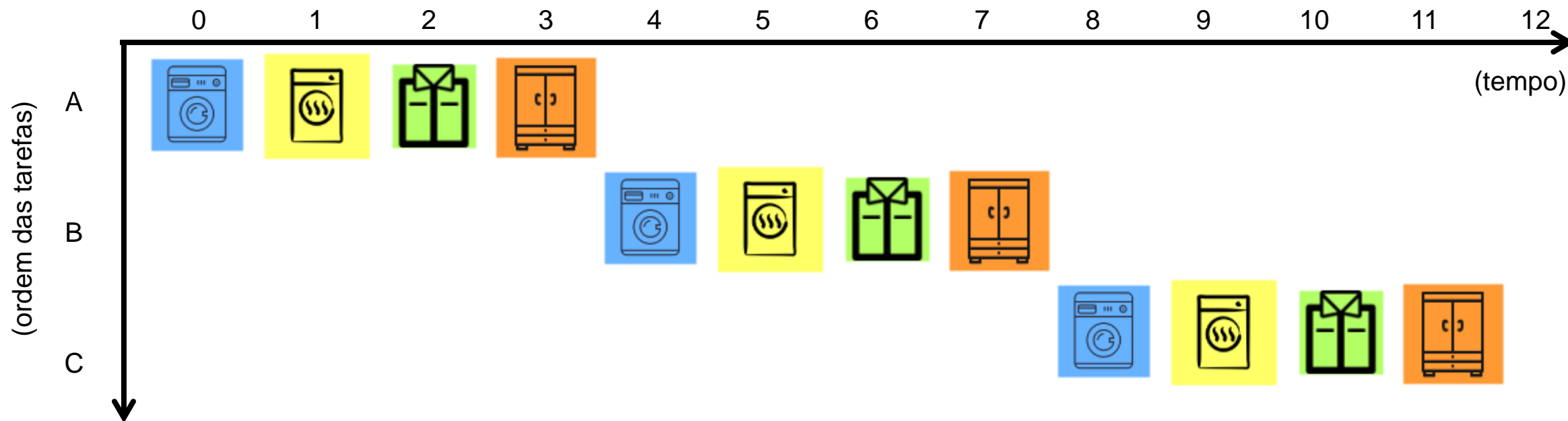


Figura 27. Exemplo de execução de um processo sem o uso do *pipeline*.

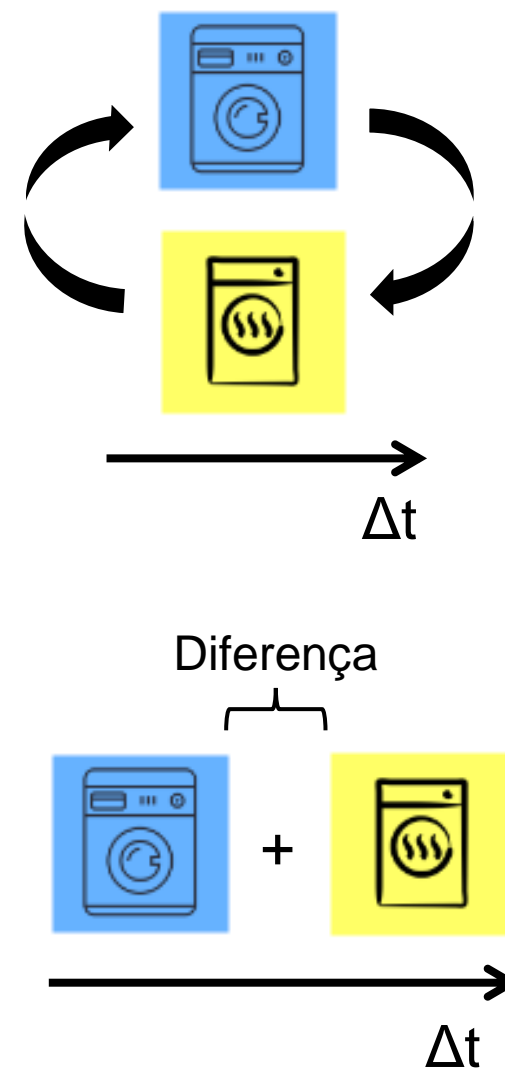


- Então, calculando o tempo estimado para a execução desses processos teríamos em minutos:

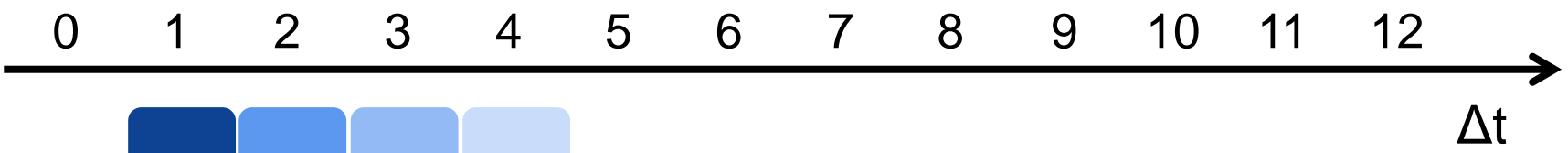
$$\begin{array}{ccccccc} & \nearrow & \text{Qt. atividades} & & & & \\ 4 & \times & 3 & = & 12 \text{ tarefas/min.} & \longrightarrow & \text{Tempo total} \\ & & \searrow & & \text{Qt. tarefas} & & \end{array}$$

- Uma observação importante é que sem o procedimento com o uso do *pipeline* só poderíamos começar uma nova tarefa após finalizada a tarefa anterior.

- É exatamente em função desse problema que o *pipeline* em processadores possibilita reduzir o tempo total de execução de múltiplas tarefas. (PANTUZA, 2020).
- Agora imagine o processo descrito e representado:
  1. Após tirar as roupas da máquina de lavar, coloque-as na secadora e quase ao mesmo tempo coloque novas roupas sujas para lavar.
  2. É possível que ao mesmo instante de tempo você terá a máquina de lavar e a secadora funcionando juntas.
- A execução paralela de múltiplas atividades permite maximizar o uso dos recursos e, simultaneamente, reduzir o tempo total de execução das tarefas.

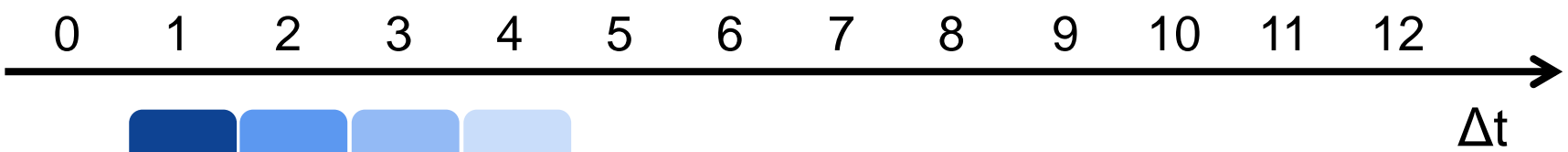


cont.[...]



4 min.

Execução SEM Pipeline



6 min.

Execução COM Pipeline

Figura 28. Exemplo de execuções de processos com e sem o uso de *pipeline*.

- Os cálculos de transmissão que são realizados no processo de programação devem ser sincronizados com um relógio (*clock*) para evitar a inundação por excesso de tráfego de dados.
- A segmentação dos cálculos de transmissão, permite melhorar a frequência de trabalho. Este tipo de fluxo de dados implica que a saída de uma fase é uma entrada de outra. Pois, observa-se que as fases encadeiam-se como uma canalização, agilizando o fluxo através do *pipeline*.
- Ressalta-se que a execução é realizada paralelamente, o que faz que se recorra a um *buffer* (um tipo de armazenamento) quando os elementos se encontram em posições consecutivas.
- Agora imaginem os vários circuitos especializados de um processador: soma, leitura de dados na memória principal, armazenamento em registradores, etc.

- Para o devido funcionamento, planeja-se que cada um desses circuitos tomem apenas o tempo de um ciclo de *clock* (sinal de relógio). Logo, todo o conjunto de circuitos do processador deve ser ajustado para que 1 ciclo de *clock* demore o tempo da instrução mais lenta dentre todas as instruções disponíveis.
- Vamos considerar um exemplo em que um processo tenha 5.000 instruções a serem executadas. Supomos um *clock* de 10 nano-segundos. Se tivermos um processador com um *pipeline* de 12 estágios qual seria o tempo total de execução de todas as instruções?

$$(5.000 \times 10) / 12 = 3333,3 \text{ ns} \quad \Rightarrow \quad \text{Processador COM pipeline}$$

$$5.000 \times 10 = 50.000 \text{ ns} \quad \Rightarrow \quad \text{Processador SEM pipeline}$$

- Devemos levar em consideração a existência de conflitos estruturais que ocorrem quando um circuito implementa mais de uma instrução. Se essas instruções forem necessárias em dois lugares diferentes apenas um poderá executar. (PANTOZA, 2020, grifo meu)
- Esse é um dos fatores que fazem com que processadores especializados para gráficos (e.g., placas de vídeo dedicada) tenha muito mais instruções de máquina e núcleos (processadores isolados), pois em sua arquiteturas há implementações de instruções que evitam conflitos estruturais. (*ibidem*).
- Outros conflitos seriam os conflitos de controle. Eles ocorrem quando a entrada de uma instrução depende da execução de outra e ambas estão em execução no *pipeline*. Nesse caso o processador precisa esperar para dar sequência na execução. (*ibidem*).

- Portanto, o *Pipeline* é uma técnica de implementação de sistemas computacionais onde o processador consegue paralelizar a execução de instruções de modo a maximizar a vazão de instruções processadas. Essa técnica adiciona complexidade na criação desse tipo de processador mas garante processadores mais eficientes. (PANTOZA, 2020).
- Os estudos desta natureza servem para medir a performance do processador através do *pipeline*, e este indicativo acaba-se por ser tornar uma forma de separação de qualidade de uso dos *hardwares*, ou seja, computadores mais caros não estão em sua estética visual, mas sim em seu *design* de arquitetura de *hardware*.

---

## REFERÊNCIAS

Equipe editorial de Conceito.de. (2 de Maio de 2015). Conceito de pipeline. Conceito.de. <https://conceito.de/pipeline>. Acessado em: 24.abr.22.

FARIAS, Gilberto. Introdução à computação. UFPB, 2013.

PANTUZA, Gustavo. Organização e Arquitetura de Computadores – Pipeline e Processadores. Atualizado em: 31.jan.20. Disponível em: <https://blog.pantuza.com/artigos/organizacao-e-arquitetura-de-computadores-pipeline-em-processadores>. Acessado em: 24.abr.22.

TANENBAUM, Andrew S. Sistemas Operacionais Modernos. 3ª ed. Pearson, 2010.

TANENBAUM, Andrew S. AUSTIN, Todd. Organização Estruturada de Computadores. 6ª ed. Pearson, 2013.

MONTEIRO, Mário A. Introdução à Organização dos Computadores. 5ª ed. LTC. 2014.

TANENBAUM, Andrew S. VAN STEEN, Maarten. Sistemas Distribuídos. Princípios e Paradigmas. 2ª ed. Pearson, 2007.

RIBEIRO, Carlos; DELGADO, José. Arquitetura de Computadores. 2ª ed. Rio de Janeiro: LTC, 2009.





# Obrigado!

Disciplina: Introdução à Computação (I.C)