

KLS

Arquitetura e Organização de Computadores

Arquitetura e Organização de Computadores

Rogerio Carlos dos Santos
Maurício Acconcia Dias
Michel Bernardo Fernandes da Silva

© 2020 por Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

Presidência
Rodrigo Galindo

**Vice-Presidência de Produto, Gestão
e Expansão**
Julia Gonçalves

Vice-Presidência Acadêmica
Marcos Lemos

**Diretoria de Produção e
Responsabilidade Social**
Camilla Veiga

Gerência Editorial
Fernanda Miglioranza

Editoração Gráfica e Eletrônica
Renata Galdino
Luana Mercurio

Supervisão da Disciplina
Marcilyanne Moreira Gois

Revisão Técnica
Marcilyanne Moreira Gois
Marcio Aparecido Artero

Imagens

Adaptadas de Shutterstock.

Todos os esforços foram empregados para localizar os detentores dos direitos autorais das imagens reproduzidas neste livro; qualquer eventual omissão será corrigida em futuras edições.

Conteúdo em websites

Os endereços de websites listados neste livro podem ser alterados ou desativados a qualquer momento pelos seus mantenedores. Sendo assim, a Editora não se responsabiliza pelo conteúdo de terceiros.

Dados Internacionais de Catalogação na Publicação (CIP)

Santos, Rogerio Carlos dos
S237a Arquitetura e organização de computadores / Rogerio Carlos dos Santos et al. – 2. ed. – Londrina: Editora e Distribuidora Educacional S.A., 2020.
248 p.

ISBN 978-65-86461-15-2

1. Arquitetura de Computador. I. Título.

CDD 621.381952

Jorge Eduardo de Almeida CRB-8/8753

2020

Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: <http://www.kroton.com.br/>



Sumário

Unidade 1

Fundamentos de sistemas computacionais.....	7
---	---

Seção 1

Conceitos básicos de arquitetura e organização de computadores.....	9
---	---

Seção 2

Desenvolvimento histórico	21
---------------------------------	----

Seção 3

A estrutura básica de computadores.....	34
---	----

Seção 4

A hierarquia de níveis de computador.....	47
---	----

Unidade 2

Componentes básicos de um computador.....	61
---	----

Seção 1

Unidade central de processamento	63
--	----

Seção 2

Memória principal e memória cache	76
---	----

Seção 3

Memória secundária	92
--------------------------	----

Seção 4

Dispositivos de entrada e saída.....	107
--------------------------------------	-----

Unidade 3

Bases numéricas, representação dos dados e instruções de máquinas	121
---	-----

Seção 1

Sistemas numéricos: conceitos, simbologia e representação de base numérica	123
--	-----

Seção 2

Introdução a instruções de máquinas.....	136
--	-----

Seção 3

<i>Pipeline</i> de instruções.....	150
------------------------------------	-----

Seção 4

Introdução à linguagem assembly	167
---------------------------------------	-----

Unidade 4

Arquiteturas de alto desempenho	187
---------------------------------------	-----

Seção 1

Introdução a arquiteturas de alto desempenho	189
--	-----

Seção 2	
Arquitetura de sistema de processamento paralelo	203
Seção 3	
Arquiteturas multithreaded	216
Seção 4	
Arquiteturas multicore.....	229

Palavras do autor

Prezado aluno, seja bem-vindo à disciplina de Arquitetura e Organização de Computadores.

Com base neste estudo, você aprenderá como essa tecnologia foi pensada – a evolução dela até os dias atuais –, terá uma visão mais clara dos computadores e de suas diversas versões e vai entender que, por mais diferentes que sejam os tipos e modelos de computadores, a tecnologia deles é baseada na evolução de padrões que há muito tempo são utilizados.

Na Unidade 1, você verá os conceitos básicos da área de arquitetura e organização de computadores, aprendendo que tal arquitetura foi pensada há muito tempo e está em constante evolução. Entenderá, ainda, como ela está dividida, a unidade central de processamento (CPU) dela, os tipos de memórias, dispositivos de entrada e saída e os sistemas de interconexão usados pelos atuais computadores.

Na Unidade 2, você compreenderá melhor essa tecnologia, o modo como ela está dividida e quais são os dispositivos e elementos básicos que compõem um computador. Além disso, conhecerá mais profundamente a unidade central de processamento (CPU), as memórias e também os dispositivos de entrada e saída, entendendo como evoluíram até aos dispositivos usados atualmente.

Na Unidade 3, por sua vez, preparamos um estudo sobre as técnicas em sistemas numéricos de máquinas, assim como sobre a identificação e a aplicação das instruções de máquinas, do paralelismo e da introdução de linguagem de montagem.

Na Unidade 4, para terminar, direcionamos o estudo para o entendimento e a identificação dos tipos de arquitetura de alto desempenho, reconhecendo as arquiteturas de processamento paralelo, assim como conceituando a arquiteturas multithread e multicore.

Bons estudos e um excelente aprendizado. Aproveite ao máximo para conhecer a fundo a tecnologia de um computador e, com certeza, começar a se destacar na área.

Unidade 1

Rogério Carlos dos Santos

Fundamentos de sistemas computacionais

Convite ao estudo

Para o estudo dos Fundamentos de Sistemas Computacionais é interessante que você observe os computadores atuais, como eles eram há alguns anos e como tiveram uma rápida evolução. Toda essa tecnologia é baseada em uma arquitetura pensada e desenvolvida em meio à Segunda Guerra Mundial e que segue em uma evolução constante. Nesta unidade você conhecerá a competência de fundamento desta área e os objetivos específicos das próximas seções.

A competência de fundamento de área da disciplina Arquitetura e Organização de Computadores é conhecer os conceitos básicos da arquitetura dos computadores, o seu desenvolvimento histórico, a estrutura básica de um computador e o modelo tecnológico adotado para os computadores atuais.

Os objetivos de aprendizagem que serão trabalhados em cada seção são:

- Conhecer os conceitos básicos de arquitetura e organização de computadores e suas funções.
- Aprender sobre como essa arquitetura foi pensada e sua evolução até os dias atuais.
- Entender como está dividida a estrutura básica de um computador, sua CPU, suas memórias, dispositivos de entrada e saída e os sistemas de interconexão.
- Conhecer como foi pensado o modelo tecnológico adotado para os computadores.

Para melhor compreensão e aprofundamento dos conceitos apresentados, apresentamos uma situação que você poderá encontrar no mercado de trabalho, “O Momento da Contratação”.

Você participará de um processo seletivo em uma empresa de desenvolvimento de tecnologia para computadores de última geração que ampliará sua fábrica no Brasil, com o objetivo de desenvolver novas estruturas de placas-mãe (*mainboards* ou *motherboards*) de alta velocidade que serão usadas em servidores de dados de grandes instituições financeiras e bancos

internacionais. Para isso, ela iniciará um processo seletivo para contratar profissionais com conhecimentos técnicos em arquitetura de computadores, o que será feito por meio de um treinamento interno com os candidatos a fim de que adquiram os conhecimentos específicos necessários. Ao final serão aplicados vários testes e serão contratados os candidatos com maior nota, em número igual ao número de vagas disponíveis no momento da contratação.

Assim, você resolverá os testes do processo seletivo ao longo da unidade para se preparar e adquirir os conhecimentos técnicos necessários para sua contratação.

Bom trabalho e bons estudos!

Seção 1

Conceitos básicos de arquitetura e organização de computadores

Diálogo aberto

Você já deve ter notado que os computadores têm muito em comum: todos têm um monitor ou tela para podermos ver as informações desejadas, teclado e dispositivos de entrada, são dotados de espaços para armazenamento e de memórias de processamento, o que permite que programas sejam usados e que você possa usar a internet, além de muitos outros recursos que eles oferecem.

Embora isso seja comum, o profissional das áreas de computação e tecnologia de informação deve conhecer o funcionamento dessas máquinas, como foram pensadas as suas estruturas e como foram divididas as funções de suas placas e componentes para que possam processar dados e comandos e retornar resultados para serem visualizados e/ou armazenados em disco.

Você vai aprofundar seus conhecimentos técnicos sobre a arquitetura e organização dos computadores para que seja bem-sucedido no processo seletivo da empresa de desenvolvimento de tecnologia para computadores.

A sua primeira tarefa é fazer a resolução de testes de conhecimento sobre a arquitetura e organização dos computadores. É necessário que você entenda que os computadores são organizados em quatro funções básicas, que dividem seus dispositivos em unidades. Você foi levado a uma sala cheia de componentes e aparelhos de computadores diversos e agora precisa classificá-los de acordo com a função desses componentes em um computador.

Mas quais conhecimentos deverão ser estudados neste ponto? Quais os conceitos e funções básicos serão necessários para que você possa participar deste processo seletivo com maiores chances de aprovação?

Existem diversas classificações para as funções dos computadores. Em uma delas, segundo Oliveira (2007), as funções básicas dos computadores são:

- Entrada de dados.
- Armazenamento de informações.
- Processamento de dados.
- Saída de informações.

Faça um relatório relacionando os componentes mais atuais do mercado. Para ajudar nesta fase da seleção, você estudará nesta seção as funções básicas de um computador. Está preparado?

Bons estudos!

Não pode faltar

Antes de conceituar arquitetura e organização de computadores, você precisa entender que o computador está dividido em duas partes principais: **hardware** e **software**. De forma simplificada e provavelmente já conhecida por você, podemos definir **hardware** como sendo toda parte física do computador, ou seja, tudo que você pode tocar do computador, e **software** como sendo a parte lógica do computador, na qual por meio de códigos os programas são executados. Ambos trabalham em sincronia para o funcionamento do computador. Todos os dados manipulados nos computadores são executados de forma binária, ou seja, pela representação 0 e 1, gerados por circuitos que assumem a função de ligado ou desligado.

Agora sim, segundo Stallings (2017), a **arquitetura de computadores** está direcionada aos atributos de um sistema visível a um programador, ou seja, todas os atributos lógicos para a execução de um programa. Outro fato bem interessante é o termo ISA (*Instruction Set Architecture*) que em português significa “Arquitetura de Conjunto de Instrução”, que representa, segundo o mesmo autor, os formatos de instruções, número de bits usados para representar os vários tipos de dados, os mecanismos de E/S (Entrada e Saída) e as técnicas de endereçamento de memória.

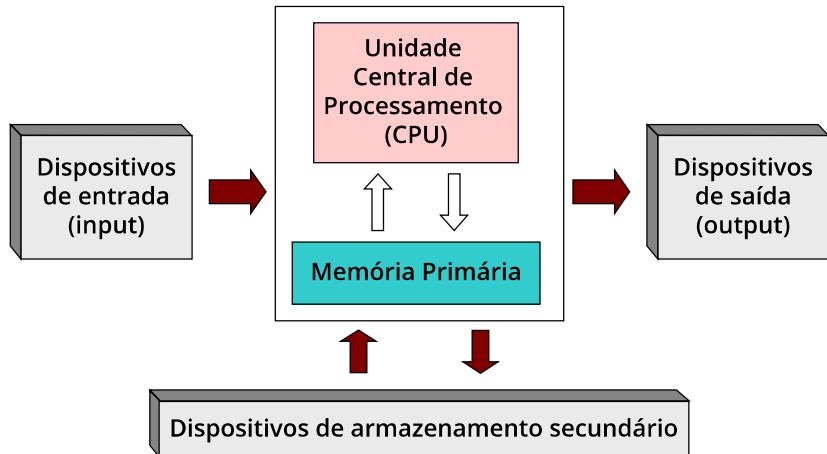
Ainda seguindo a contextualização feita por Stallings (2017), a **organização de computadores** está direcionada para as unidades operacionais e suas interconexões, ou seja, que percebam as especificações de arquitetura.

Seguindo o contexto da arquitetura do conjunto de instruções (ISA), podemos dizer que essa arquitetura representa a interface entre o hardware e software. Hennessy e Patterson (2014) definem que para a implementação de um computador é preciso ter **organização** e **hardware**. Os autores definem **organização** como sendo a interconexão entre a memória e a CPU/UCP (unidade central de processamento). Vale salientar que a organização também é colocada como sendo representada pela microarquitetura, na qual dois processadores podem implementar um conjunto de instruções, apresentando, no entanto, organizações diferentes. Já os **hardwares** são caracterizados aos detalhes específicos do computador. Muitas vezes os computadores têm arquiteturas de conjunto de instruções idênticas e organizações quase idênticas, diferindo na implementação detalhada do hardware. Podemos

citar os exemplos apresentados por Hennessy e Patterson (2014), em que um processador Intel Core i7 (modelo de processador com melhor performance de processamento, ou seja, mais rápido para executar as operações) e o Intel Xeon 7560 (um ótimo processador, porém, com uma performance menor) são praticamente idênticos, mas oferecem taxas de clock (velocidade) e sistemas de memórias diferentes. Cabe dizer que o processador Xeon 7560 é mais aconselhável para computadores que exigem menor tempo de processamento.

Entender como funciona a estrutura básica de um computador faz parte do início desta jornada de conhecimento. Veja na Figura 1.1 as funções básicas de um computador.

Figura 1.1 | Funções básicas de um computador



Fonte: Introdução... (2012, [s.p.]).

Agora, vamos caracterizar alguns dos itens citados na Figura 1.1:

Dispositivos de entrada (input) de um computador: são caracterizados pelo mouse, teclado, monitores com recursos *touch screen*, leitores ópticos, entre outros dispositivos que fornecem informação para serem processadas. É neste momento que acontece a interação entre o usuário e o computador.

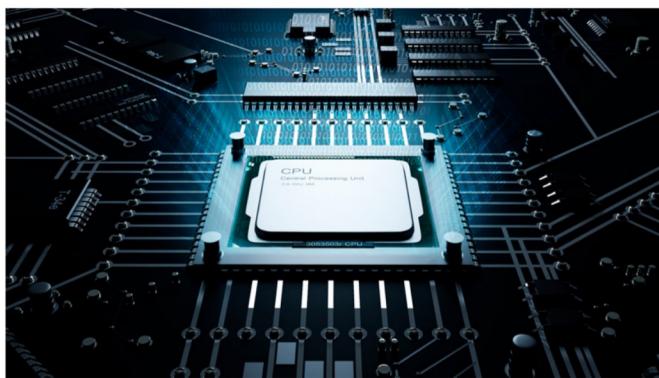
Figura 1.2 | Dispositivos de entrada



Fonte: Shutterstock.

Unidade de Processamento do computador: também conhecida como CPU (*Central Processing Unit*) em português chamada de unidade central de processamento. Responsável pela execução de instruções lógicas e matemáticas, operação de busca, de leitura e gravação de dados. A CPU é o canal direto com a memória principal do computador, proporcionando resultados dos processamentos para os usuários.

Figura 1.3 | Unidade de processamento do computador (CPU)



Fonte: Shutterstock.

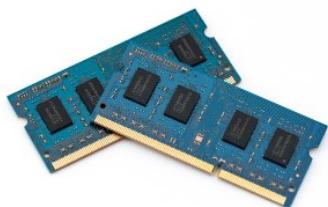
É sabido que a CPU é o principal componente do computador, pois fica em constante comunicação com todos os seus dispositivos. Universalmente conhecida, podemos caracterizar as três partes mais importantes da CPU:

- Unidade Aritmética e Lógica (ULA): executa as operações aritméticas do computador (soma, subtração, divisão e multiplicação, entre outras) e as operações lógicas (and, or, xor, not, entre outras).
- Unidade Controle: apresenta uma alta complexidade em relação à CPU, pois controla todas as ações da ULA e instruções direcionadas para o processamento.
- Registradores: considerada a memória de maior relevância do computador, situada no core (núcleo) do computador, é a memória de mais veloz.

Memória principal: é um depositório temporário, em que os dados e instruções são armazenadas por um tempo determinado. Uma vez processadas as informações da memória, ela é liberada para novas informações e, por este motivo, é chamada de memórias voláteis. São exemplos de memória principal: memória RAM (memória de acesso aleatório) e memória cache.

Figura 1.4 | a) Memória RAM e b) cache

a)



b)



Fonte: Shutterstock.

Dispositivos de armazenamento: são dispositivos que podem armazenar dados para serem recuperados quando necessário. As fontes de armazenamentos podem ser do tipo magnéticos, ópticos e por meios eletrônicos.

Armazenamento magnético: utilizado para grandes volumes de dados e de custo relativamente acessível, seus mecanismos são formados por uma cabeça magnética de leitura e gravação, gerando um campo magnético e magnetizando, assim, os dipolos magnéticos, que por sua vez representam os bits (0 e 1) de acordo com a polaridade.

Um exemplo clássico de armazenamento por meio magnético são os discos rígidos, os sobreviventes HDs. Vale lembrar que esses dispositivos podem ser instalados internamente no computador ou externamente, podendo ser utilizados como dispositivos móveis, os chamados HDs externos.

Figura 1.5 | Dispositivos de armazenamento – HD



Fonte: Shutterstock.

Armazenamento por meios ópticos: utiliza dispositivos que fazem leituras e gravações por meio de feixes de laser. Podemos citar como exemplo os CD-ROMs, CD-RWs, DVD-ROMs e DVD-RWs, entre outros. Sua tecnologia está quase que totalmente substituída pelos dispositivos de armazenamento de meio eletrônicos e nuvem (armazenamento realizado em dispositivos de acesso pela internet).

Figura 1.6 | Dispositivo de meios ópticos – CD-ROM



Fonte: Shutterstock.

Armazenamento por meios eletrônicos: se dá com dispositivos compostos por circuitos eletrônicos, que utilizam corrente elétrica para armazenar os dados. Ao contrário dos demais dispositivos, não dispõem de mecanismos de

movimentação. São caracterizados armazenamentos SSD (*solid state drive*), em português “Unidade de Estado Sólido”. São exemplos de armazenamento por meios eletrônicos: pen drives, cartões de memórias, SSD de alta capacidade e substitutos dos HDs, entre outros dispositivos.

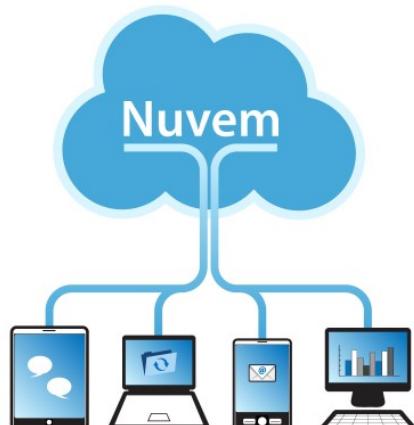
Figura 1.7 | Dispositivos por meios eletrônicos: a) SSD e b) pen drives e cartões de memória



Fonte: Shutterstock.

Armazenamento em nuvem (cloud): tecnologia usada para armazenar as informações em um servidor (computador de alta capacidade de armazenamento e velocidade) por meio do acesso pela internet. Neste caso os arquivos podem ser gravados, apagados, editados e executados. Os Data Centers, assim chamados, oferecem seus serviços para acesso em qualquer lugar e com a máxima segurança dos arquivos trafegados, bastando ter uma conta e acesso à internet.

Figura 1.8 | Acesso à nuvem (cloud)



Fonte: Shutterstock.

Dispositivos de saída (output): são os dispositivos que retornam tudo que foi processado pelo computador, pelos quais acontece a interação entre computador e usuário. Podem ser caracterizados como dispositivos de saída: monitores, impressoras, projetores e dispositivos de som, entre outros.

Figura 1.9 | Periféricos de saída: a) impressora; b) desktop, notebook, tablet, smartphone; c) fones de ouvido sem fio



Fonte: Shutterstock.

As medidas de tamanho usadas em um computador são baseadas em bytes, que são uma sequência de 8 bits. Um único bit pode ser representado pelos números 0 e 1. Esta medida é adotada por todas as áreas que envolvam processamento, envio e recebimento de dados e informações, sendo que cada byte representa um caractere de texto no computador. Com esse raciocínio, agora podemos entender a Tabela 1.1. As medidas de bytes usadas são:

Tabela 1.1 | Unidade de medida do computador

Medida	Sigla		Caracteres	
Byte		2^0	1 (8 bits)	1 byte
Kilobyte	KB	2^{10}	1.024	1.024 bytes
Megabyte	MB	2^{20}	1.048.576	1.024 KBytes
Gigabyte	GB	2^{30}	1.073.741.824	1.024 MBytes
Terabyte	TB	2^{40}	1.099.511.627.776	1.024 GBytes
Pentabyte	PB	2^{50}	1.125.899.906.842.624	1.024 TBytes
Hexabyte	HB	2^{60}	1.152.921.504.606.846.976	1.024 PBytes
Yotabyte	YB	2^{80}	1.208.925.819.614.630.000.000.000	1.024 Hexabyte

Fonte: Unidades... ([s.d.], [s.p.]).

Podemos dizer que vários dispositivos móveis, laptops e desktops são considerados computadores, pois estes equipamentos apresentam dispositivos de entrada, processadores, memórias e dispositivos de saída, certo? Porém, cabe a você decidir qual computador melhor atende suas necessidades e as da sua empresa.

Bons estudos!

Sem medo de errar

Para que você possa se preparar para a situação geradora de aprendizagem proposta nesta seção, que é a resolução de testes de conhecimento sobre a arquitetura e organização dos computadores, é necessário que você entenda que os computadores são organizados em quatro funções básicas, que dividem seus dispositivos em unidades. Essa arquitetura é usada até hoje, porém sofre constante evolução.

Imagine que você seja levado a uma sala cheia de componentes e aparelhos de computadores diversos e tenha que os classificar de acordo com a função que têm em um computador.

Claro que a solução desse desafio pode mudar a qualquer momento, certo?

Entrada de dados:

Teclado – selecionado o teclado com conexão *wireless*, ou seja, sem fio.

Mouse – selecionado o mouse Gamer de 11 botões, *wireless* recarregável.

Monitor Touch Screen – 24” em Led.

Processamento de dados:

Foram selecionados dois processadores:

1. Ryzen 7 2700

- 3.2GHz de frequência.
- 8 núcleos com 16 threads.
- 16MB de memória cache.
- Suporte à memória DDR4.
- Soquete AM4.

2. Core i9 9900K

- 3.6GHz de frequência.
- 8 núcleos com 16 threads.
- 16MB de memória cache.
- Suporte à memória DDR4.
- Soquete LGA 1151 v2.

Foi selecionada a seguinte memória para trabalhar em conjunto com os processadores apresentados:

Memória HyperX Fury, 16GB, 2400MHz, DDR4, CL15.

Armazenamento de informações:

Seguindo as unidades de armazenamento mais atuais, foram selecionadas:

Ssd 1.9Tb Sata 3 Intel 2.5 S4500.

HDD Seagate Barracuda 4 TB P/ Desktop.

Cartão de Memória MicroSDXC 400GB.

Pen drive USB 3.0 – 1TB.

Saída de informações:

Relacionadas as unidades de saída mais atuais do momento:

Monitor:

Tamanho: 24.5 polegadas.

Tipo: TN.

Resolução: Full HD (1920 x 1080).

Frequência: 240Hz.

Tempo de resposta: 1ms.

Impressora:

Laser 20 ppm 1200 x 1200 DPI A4 Wi-fi – Multifuncional.

Fone de ouvido:

Conectividade: Bluetooth 4.0.

Autonomia de bateria: 12 horas.

Função Fast Fuel para recarregamento rápido.

Função Remote Talk para comandos de voz.

Resistência a suor e água.

Muito bem, esperamos que você tenha entendido os conceitos das arquiteturas de um computador. Até a próxima situação-problema e bons estudos!

Faça valer a pena

- 1.** Com base nas funções básicas dos computadores, analise a tabela a seguir que apresenta uma lista de componentes presentes nos computadores.

Tabela | Componentes de um computador

Coluna I	Coluna II
1 – Entrada de dados	A – Impressora 3D
2 – Processamento de dados	B – Data centers
3 – Armazenamento de informações	C – Tela sensível ao toque
4 – Saída de informações	D – Unidade de controle

Fonte: elaborada pelo autor.

Assinale a alternativa que apresenta corretamente a associação entre as colunas.

- a. 1 – A; 2 – C; 3 – D; 4 – B.
- b. 1 – C; 2 – D; 3 – B; 4 – A.
- c. 1 – B; 2 – C; 3 – D; 4 – B.
- d. 1 – B; 2 – D; 3 – A; 4 – C.
- e. 1 – C; 2 – A; 3 – D; 4 – B.

- 2.** A arquitetura dos computadores representa a interface entre o hardware e software, enquanto a organização é a interconexão entre a memória e a CPU/UCP (unidade central de processamento). Neste contexto, a memória do computador pode trabalhar de duas formas distintas: como primária ou como secundária.

Analise as asserções a seguir e a relação proposta entre elas:

- I. A memórias primárias são aquelas com maior capacidade de armazenamento.

PORQUE

- II. O papel dela é ter de registrar e armazenar os dados dos usuários.

A respeito dessas asserções, assinale a alternativa correta.

- a. As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- b. As asserções I e II são proposições verdadeiras e a II justifica a I.
- c. A asserção I é uma proposição verdadeira e a II, falsa.

- d. A asserção I é uma proposição falsa e a II, verdadeira.
 - e. As asserções I e II são proposições falsas.
- 3.** Os ciclos de inovação para o mercado de Tecnologia da Informação e da Comunicação estão cada vez mais curtos, o que exige dos profissionais uma alta capacidade para entender a organização dos computadores e de como flexibilizar as entradas de dados, processamento, armazenamento e saída. Exemplos de tecnologias para hardwares sem fio (*wireless*) são inúmeros.

Preencha com V (verdadeiro), ou F (falso) para os elementos da arquitetura de computadores que podem operar sem fio:

- () Entrada de dados.
- () Processamento e armazenamento.
- () Saída de dados.

Assinale a alternativa que apresenta a sequência correta.

- a. V – F – V.
- b. V – V – F.
- c. F – F – V.
- d. V – F – F.
- e. V – V – V.

Seção 2

Desenvolvimento histórico

Diálogo aberto

Os computadores de hoje são usados nas mais variadas atividades do dia a dia. Se você observar com atenção, vai encontrar computadores em hospitais, padarias, oficinas mecânicas, no seu carro, enfim, não há limites para a utilização de computadores. Por isso, os fabricantes e pesquisadores de tecnologia investem cada vez mais no desenvolvimento de novas aplicações de uso, programas e equipamentos que possam tornar mais ágeis as atividades feitas pelo homem.

Observando mais atentamente, você pode ver que os computadores de hoje são o resultado de anos de pesquisas, e que esse processo se iniciou há muito tempo. Essa evolução foi marcada por gerações de computadores, cada uma delas abrindo campo para a evolução e desenvolvimento da próxima geração, aperfeiçoando os conceitos empregados, seus componentes, placas e circuitos.

Nesta seção você verá quais foram as gerações, em que época elas foram utilizadas e como contribuíram para as tecnologias dos computadores atuais.

Sua segunda tarefa é fazer a resolução de testes de conhecimento sobre qual geração de computadores engloba quais modelos. É necessário que você conheça a história da evolução dos computadores, suas gerações, os componentes principais dessas tecnologias e onde eles foram empregados para que você possa classificar cada modelo de computador de acordo com sua geração. Lembre-se de que você está participando de um processo seletivo em uma empresa de desenvolvimento de tecnologia para computadores de última geração que ampliará sua fábrica no Brasil e que, ao final, serão contratados os candidatos com maior nota, em número igual ao número de vagas disponíveis no momento da contratação.

Os computadores do passado eram relativamente parecidos com os que temos hoje? Sua arquitetura básica e suas funções eram iguais aos dos nossos computadores atuais? Onde eles eram usados? Para essas e outras questões você encontrará respostas conhecendo cada uma das gerações dos computadores e suas tecnologias.

Bom trabalho e bons estudos!

Não pode faltar

É possível encontrar muitas semelhanças entre computadores antigos e os diversos tipos de computadores usados nos dias de hoje. Sua arquitetura e funções básicas foram mantidas e isso traz a sensação de que não existe nada de muito novo, que apenas foi melhorado o que já existia. Essa sensação se dá quando não olhamos com maior atenção o passado dessas máquinas, como elas eram e como são hoje. Os conceitos envolvidos em um computador, em sua arquitetura e em suas funções básicas são mais antigos do que o próprio computador. Você verá, ao longo da história, como surgiram as gerações dos computadores, quantas são e em qual estamos nos dias de hoje. Acompanhe como foi essa evolução.

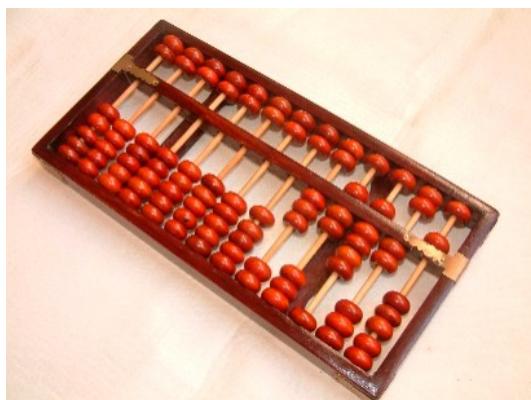
Máquinas mecânicas de cálculo

O Ábaco

Considerado o primeiro tipo de computador, é uma máquina de cálculo mecânica e rudimentar usada por vários povos da antiguidade. Há estudos arqueológicos que apontam seu uso 4.000 anos a.C. (antes de Cristo).

O ábaco usa um determinado método de cálculo no qual os números são representados por bolas de madeira sistematicamente colocadas em uma estrutura em que uma pessoa pode executar cálculos aritméticos, desde os mais simples até os mais complexos e elaborados (SOUZA FILHO; ALEXANDRE, 2014).

Figura 1.10 | Funções básicas de um computador



Fonte: Wikimedia.

Ossos de Napier

No ano de 1614, John Napier descobriu os cálculos através de logaritmos. Em matemática, logaritmos são expoentes utilizados em números para gerar outros números. Por exemplo, o logaritmo do número 1000 em base 10 é 3, pois $10^3 = 1000$. Napier desenvolveu assim uma tabela de logaritmos chamada de Osso de Napier, que auxiliava na realização de multiplicações facilitando a obtenção de resultados em cálculos complexos (SOUZA FILHO; ALEXANDRE, 2014).

As Rodas Dentadas de Pascal – Pascaline

Figura 1.11 | Máquina mecânica de cálculo de Pascal – Pascaline



Fonte: Wikimedia.

Inventada em 1642 por Blaise Pascal, essa máquina, chamada na época de Pascaline, foi a primeira calculadora do mundo. Com uma estrutura mecânica de engrenagens, foi projetada para realizar as quatro operações matemáticas, porém, na prática, realizava automaticamente as operações de soma e subtração, e realizava as operações de multiplicação e divisão por meio de um processo de repetição (SOUZA FILHO; ALEXANDRE, 2014).

A máquina analítica de Babbage

Projetada por Charles Babbage em 1837, era uma máquina para uso genérico que teria uma programação feita por comandos escritos e descritos em cartões perfurados. Estes cartões poderiam ser usados para armazenar ideias abstratas ou números, e esse conceito abriu caminho para a definição das unidades de armazenamento e processamento de dados. Passado algum

tempo, Ada Byron, ou Ada Lovelace (Condessa de Lovelace), filha do famoso Lord Byron, interessou-se por esta máquina e estabeleceu contato com Babbage com cartas e pessoalmente. Ela passou a escrever sequências de códigos que poderiam ser executados pela máquina caso esta fosse construída. Também observou que tais comandos necessitavam de *loops* (laços de execução de comandos) e de sub-rotinas para serem executados. Isso rendeu à Ada o reconhecimento de primeira programadora da história. A máquina de Babbage nunca chegou a ser construída de fato, mas seus conceitos contribuíram em muito para os computadores modernos (SOUZA FILHO; ALEXANDRE, 2014).

Máquina de Turing

Alan Turing, matemático britânico, publicou em 1936, em Cambridge, um artigo com o título *Máquina Universal*, que descrevia uma máquina conceitual, um modelo abstrato que estudava apenas os aspectos lógicos do funcionamento de um computador, como memória, processamento e linguagens aplicadas na resolução de algoritmos e problemas matemáticos computáveis. As máquinas universais são chamadas também de Máquinas de Turing e serviram de base para a Ciência da Computação e para o surgimento da arquitetura dos computadores modernos (TEIXEIRA, 1998).

Essa máquina teórica foi aperfeiçoada pelo matemático John von Neumann, que definiu a arquitetura básica dos computadores modernos, chamada de Arquitetura de Neumann (LOPES, 1997).

Reflita

Os computadores são máquinas capazes de realizar cálculos de forma automática e tiveram sua estrutura básica pensada já na teoria de Alan Turing sobre Máquinas Universais, que criava uma máquina abstrata capaz de testar os aspectos lógicos de um computador, sua linguagem e quais cálculos poderiam ser computados por meio de uma máquina (TEIXEIRA, 1998, p. 20-21).

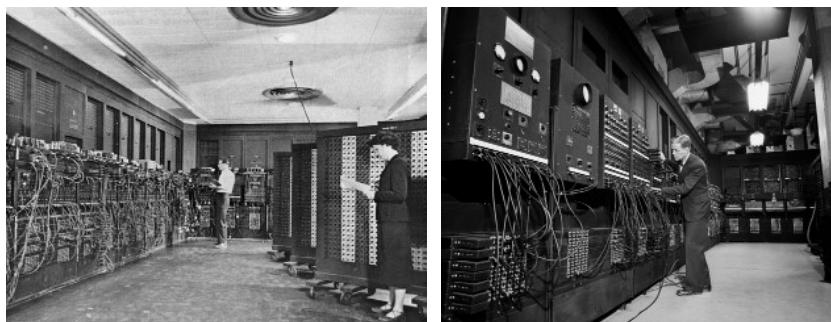
Primeira geração de computadores

Os computadores são máquinas capazes de realizar cálculos de forma automática e armazenar seus resultados. Para isso há dispositivos que permitem a entrada dos dados, e sua visualização acontece por meio de dispositivos de saída (SOUZA FILHO; ALEXANDRE, 2014).

A primeira geração dessas máquinas apareceu entre 1946 e 1954. Eram computadores que funcionavam à válvula, um tubo de vidro parecido com uma lâmpada e que tinha a função de proporcionar o processamento de informações. As instruções eram programadas diretamente em linguagem de máquina e gravadas em cartões perfurados, o que tornava o seu funcionamento lento e sua programação difícil de ser executada (SOUZA FILHO; ALEXANDRE, 2014).

Uma máquina dessa geração era a ENIAC, com 17.468 válvulas, 180 metros quadrados de área e, para a época, a incrível velocidade de 100 Quilohertz (KHz) e memória RAM de 200 bits.

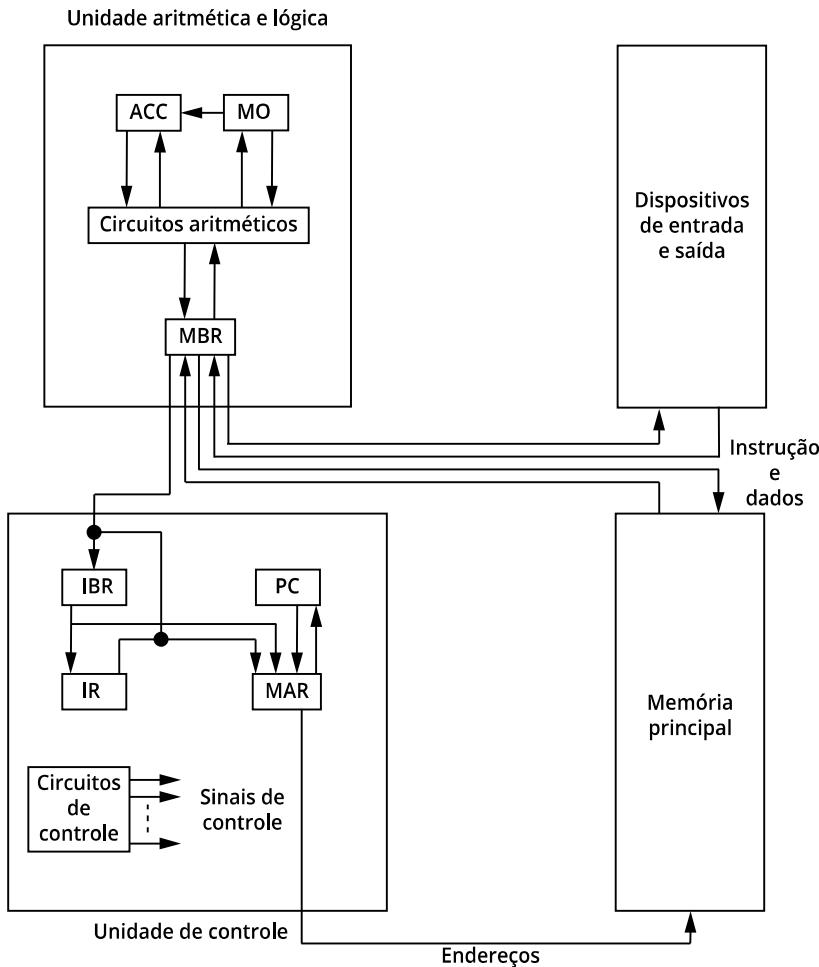
Figura 1.12 | Máquina ENIAC



Fonte: Wikimedia.

Em 1946, ainda na primeira geração, Von Neumann, entre outros cientistas de Princeton, deram início a um novo projeto chamada de IAS utilizando os mesmos princípios do EDVAC.

Figura 1.13 | Diagrama em blocos da estrutura do IAS



Fonte: Monteiro (2007, p. 19).

Segundo Monteiro (2007), o IAS ainda é base de estudo para a arquitetura de computadores atualmente. Como você pode visualizar na Figura 1.13, o IAS é constituído das quatro unidades principais que compõem o computador.

Assimile

Você sabia que o termo *bug* (“inseto”) é utilizado para representar um erro ou uma falha nos sistemas computacionais? Pois bem, a história relata que em 9 de setembro de 1947, especificamente no computador Mark II, onde William Burke, um dos operadores do computador trabalhava, apareceu uma mariposa entre os fios do computador, causando o mau funcionamento da máquina. Daí para a frente todo tipo de erro, seja causado por hardware ou software, é chamada por muitos de *bug*. (STIVANI, 2019, [s.p.]).

Segunda geração de computadores

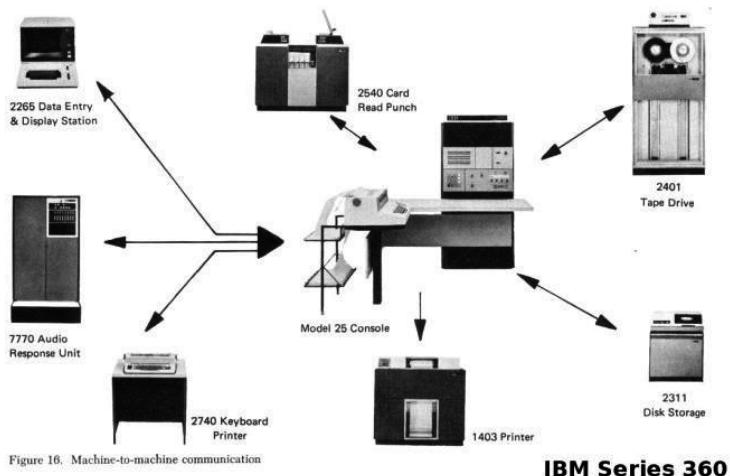
A segunda geração de computadores surgiu entre 1955 e 1964, e sua principal evolução foi a substituição das válvulas pelos transistores. Eles revolucionaram a eletrônica da época por serem muito menores do que as válvulas, além de não precisarem de um preaquecimento para funcionar, sendo, assim, incorporados aos computadores. Além disso, outra evolução importante foi a criação da linguagem Assembly em substituição à linguagem de máquina, e, em seguida, das linguagens Fortran e Pascal. Pertence a essa geração também o surgimento de armazenamento em disco e fita magnética, formas de acesso rápido aos dados gravados (SOUZA FILHO; ALEXANDRE, 2014).

Terceira geração de computadores

Entre 1964 e 1977, a terceira geração de computadores surgiu e sua principal evolução foram os circuitos integrados, chamados assim porque integravam milhares de transistores em um único componente eletrônico, reduzindo drasticamente o tamanho das máquinas e também aumentando muito rapidamente a capacidade de processamento dos computadores. Os circuitos integrados também foram chamados de microchips. Os computadores passaram a ser programados em linguagens de alto nível, como Cobol e Fortran (SOUZA FILHO; ALEXANDRE, 2014).

A partir da terceira geração, a facilidade de reprodução dos dispositivos foi de total relevância para produção em massa dos circuitos integrados. Um dos grandes destaques desta geração foi o IBM's System/360, direcionado para área científica e comercial. Suas principais características eram a facilidade de substituição e integração entre seus componentes.

Figura 1.14 | Arquitetura de conexões e integração da série 360 da IBM



Fonte: IBM ([s.d.]).

Exemplificando

Segundo descrito no site da Fundação Bradesco, surgiram os aplicativos capazes de realizarem integração entre softwares, editores de textos, planilhas eletrônicas, gerenciadores de banco de dados, criação de gráficos complexos e gerenciadores de comunicação (FUNDAÇÃO BRADESCO, [s.d.]).

Seguindo a tendência tecnológica, os avanços para o processo de miniaturização dos circuitos integrados não pararam, e podemos citar: LSI (*Large Scale of Integration*), VLSI (*Very Large Scale of Integration*) e ULSI (*Ultra Large Scale of Integration*), utilizado a partir de 1980. Nesta geração começa a utilização das linguagens de altíssimo nível, orientadas para um problema.

Quarta geração de computadores

Entre 1977 e 1991, a quarta geração de computadores trouxe o processador, que é um chip dotado de unidade central de processamento. Nesse momento foram criados sistemas operacionais que revolucionaram o uso de computadores, como o Unix, o MS-DOS e o Apple Macintosh. Linguagens como Smalltalk, C e C++ foram desenvolvidas e equipamentos complementares a essa tecnologia, tais como discos rígidos, impressoras e teclados com os modelos atuais, foram criados. Um grande avanço que mudaria o destino

dos computadores e do mundo moderno foram os microcomputadores pessoais, também chamados de PCs (*Personal Computers*) (SOUZA FILHO; ALEXANDRE, 2014).

Quinta geração de computadores

Desde 1991 até os dias atuais, os computadores estão em sua quinta geração. Esta geração trouxe inúmeras inovações, tais como o processador de 64 bits, discos rígidos de grande capacidade, memórias de trabalho e processamento cada vez maiores e inúmeros dispositivos que tornaram o uso do computador cada vez maior. Essa quinta geração de computadores é marcada também por sua grande capacidade de conexão, fundamental para a internet, e por proporcionar evoluções no campo da inteligência artificial (SOUZA FILHO; ALEXANDRE, 2014).

Nesse contexto, você pode se perguntar: como será a evolução dos computadores e seus componentes daqui por diante?

Uma teoria foi criada sobre isso e por vários anos foi feita uma observação sobre o que ela descrevia: a chamada “Lei de Moore”. Em 1965, Gordon Moore, que fundou a empresa Intel (um dos maiores fabricantes de processadores e chips de computadores do mundo até hoje), previu que a densidade de transistores em um circuito integrado dobraria a cada ano. Moore fez essa projeção com base na relação preço/desempenho dos chips produzidos nos anos anteriores. Essa afirmação acabou sendo chamada de Lei de Moore e, na prática, a densidade de transistores dentro de um chip dobrou a cada 18 meses, em média. Porém, devido à própria limitação tecnológica encontrada no processo de fabricação e os custos cada vez mais altos envolvidos, essa máxima não será aplicada para sempre. Mesmo assim, por muitos anos confirmou-se a afirmação de Moore, que permaneceu verdadeira até praticamente o final da década de 2010. Ainda que não mantendo o ritmo da Lei de Moore, o processo de evolução dos chips dos computadores permanece, e novos chips e tecnologias são constantemente lançados no mercado (TAURION, 2005).

Assimile

Para gravar! Os computadores modernos foram classificados em gerações, de acordo com sua tecnologia e desempenho:

1^a Geração: entre 1946 e 1954 – válvulas.

2^a Geração: entre 1955 e 1964 – transistores.

3^a Geração: entre 1964 e 1977 – circuitos integrados.

4^a Geração: entre 1977 e 1991 – microchips (8 e 16 bits).

5^a Geração: entre 1991 até os dias atuais – microchips (>16 bits), multi-mídia, rede.

Ainda podemos destacar que a quinta geração está marcada pela conectividade e pelo avanço nas redes neurais e inteligência artificial. Estes, por sua vez, integram todas as áreas de conhecimento (Ciências Humanas e suas tecnologias, Ciências da Natureza e suas tecnologias, Linguagens, Códigos e suas tecnologias, Matemática e suas tecnologias).

Muito bem, esperamos que você tenha adquirido o conhecimento necessário para interpretar e conceituar as gerações dos computadores. Lembre-se: não só os entusiastas preveem a sexta geração de computação, mas todos que presenciam os avanços tecnológicos.

Bons estudos!

Sem medo de errar

Para que você possa se preparar para a situação geradora de aprendizagem proposta nesta unidade, que é a resolução de testes de conhecimento sobre a arquitetura e organização dos computadores, é necessário que você entenda também sobre a evolução dos computadores e suas gerações tecnológicas.

Imagine que você seja levado a uma sala cheia de computadores de diversas épocas e tenha que os classificar de acordo com a sua geração. Como provavelmente eles estarão sem funcionar, é necessário que você conheça a história da evolução dos computadores, suas gerações e suas principais características, para que possa classificar cada modelo de computador corretamente, de acordo com sua geração.

Quadro 1.1 | História da evolução dos computadores

Lembre-se!	<p>Os computadores modernos foram classificados em gerações, de acordo com sua tecnologia e desempenho:</p> <ul style="list-style-type: none">• 1^a Geração: entre 1946 e 1954 – eram computadores que funcionavam à válvula, um tubo de vidro parecido com lâmpadas e que tinha a função de proporcionar o processamento de informações.• 2^a Geração: entre 1955 e 1964 – sua principal evolução foi a substituição das válvulas pelos transistores e o surgimento de armazenamento em disco e fita magnética.• 3^a Geração: entre 1964 e 1977 – sua principal evolução foram os circuitos integrados, chamados assim porque integravam milhares de transistores em um único componente eletrônico.• 4^a Geração: entre 1977 e 1991 – trouxe aos computadores o processador, um chip dotado de unidade central de processamento. Foram criados sistemas como o Unix, o MS-DOS e o Apple Macintosh. Um grande avanço foi o lançamento dos microcomputadores pessoais, também chamados de PCs.• 5^a Geração: entre 1991 até os dias atuais – trouxe aos computadores inúmeras inovações, tais como o processador de 64 bits, discos rígidos de grande capacidade, memórias de trabalho e processamento cada vez maiores e inúmeros dispositivos que tornaram o uso do computador progressivamente mais difundido, como a capacidade de conexão fundamental para a internet.
------------	---

Fonte: elaborado pelo autor.

Os computadores que você classificará por geração são:

- Um lote de computadores com gabinete, teclado, mouse, monitor e kit multimídia.
- Um computador desmontado, com placas quadradas grandes, como se fossem quadros de madeira, e cheias de válvulas.
- Um computador parecido com um grande armário, na parte frontal um compartimento formando uma caixa, com porta de vidro; dentro, dois grandes rolos de fita magnética.
- Um computador IBM/PC antigo, com a inscrição PX/XT.
- Um notebook com wi-fi e bluetooth, 16GB de RAM e HD de 2 TB.
- Um tablet com o símbolo Android e IOS.

Boa sorte e ótimos estudos!

Faça valer a pena

1. Segundo as gerações de computadores, analise as afirmações a seguir marcando (V) para Verdadeiro (V) ou (F) para Falso:

- () Os computadores da 4^a Geração apareceram entre 1977 e 1991.
- () Os computadores da 4^a Geração apareceram entre 1991 e 2013.
- () Entre 1936 e 1945 foram lançados os computadores primitivos de 1^a Geração.
- () Entre 1946 e 1954 os computadores já estavam em sua 3^a Geração.
- () Entre 1964 e 1977 os computadores estavam em sua 3^a Geração.

Assinale a alternativa que corresponde à sequência correta das afirmações.

- a. F – V – F – F – V.
- b. F – V – F – V – F.
- c. V – F – F – F – V.
- d. V – F – F – V – F.
- e. V – F – V – V – F.

2. Os primeiros computadores funcionavam com válvulas, que foram substituídas por transistores. Em seguida surgiram os circuitos integrados, que traziam em sua estrutura milhares de transistores integrados.

Segundo a Lei de Moore, a densidade de transistores em um circuito integrado, na prática, dobrou:

- a. A cada novo modelo de chip.
- b. A cada 1 ano.
- c. A cada 2 anos.
- d. A cada 36 meses.
- e. A cada 18 meses.

3. A primeira geração dos computadores aconteceu entre 1946 e 1954. Eram computadores que funcionavam à válvula, um tubo de vidro parecido com uma lâmpada com a função de proporcionar o processamento de informações.

Da primeira geração de computadores podemos dizer que:

- a. As instruções eram programadas diretamente em linguagem de máquina e gravadas em cartões perfurados, o que tornava o seu funcionamento lento e sua programação difícil de ser executada.
- b. Ainda não existia uma programação, pois os computadores trabalhavam apenas com válvulas em periféricos de saída.
- c. Eles revolucionaram a eletrônica da época, eram muito menores que as válvulas, não precisavam de um aquecimento para poder funcionar.
- d. Reduziu drasticamente o tamanho das máquinas, aumentando muito rapidamente a capacidade de processamento dos computadores.
- e. As linguagens como Smalltalk, C e C++ foram desenvolvidas e equipamentos complementares a essa tecnologia, tais como discos rígidos, impressoras e teclados, foram criados.

Seção 3

A estrutura básica de computadores

Diálogo aberto

Olá, caro aluno! Como você já deve ter notado, existem diversos tipos de computadores: desktops, notebooks, tablets, smartphones, consoles de games e muitos outros. Em todos você poderá notar muitas semelhanças, como a capacidade de processamento, memórias e armazenamento de informações.

Nesta seção você vai aprofundar seus conhecimentos sobre a estrutura básica de um computador, sua Unidade Central de Processamento (CPU), sua memória principal e seus dispositivos de entrada e saída, além dos sistemas de interconexão usados pelos computadores atuais. Essa estrutura foi implementada logo após a Segunda Guerra Mundial, e foi proposta por John von Neumann, matemático húngaro, radicado e naturalizado nos Estados Unidos da América, envolvido com o desenvolvimento dos primeiros computadores usados. Ela é chamada de Arquitetura de von Neumann e tem servido como base para as novas tecnologias.

Ao conhecer mais sobre essa estrutura, você poderá perceber que os computadores seguem esse modelo tecnológico, que existem muitos avanços já realizados nessas tecnologias e quanto mais detalhadamente você olhar para isso, mais claro e descomplicado será entender os computadores, seu funcionamento e seus dispositivos e componentes. Tais conhecimentos serão necessários para que você seja bem-sucedido no processo seletivo da empresa de desenvolvimento de tecnologia para computadores de última geração que vai ampliar sua fábrica no Brasil. Lembre-se: serão contratados os candidatos com maior nota no processo seletivo.

Quanto mais você conhecer sobre essa estrutura, mais entenderá como os computadores são montados e como funcionam. Nesta situação-problema, você deverá elaborar um relatório com as principais configurações de computadores que serão adquiridos pela empresa.

Processador:

Memória:

Placa mãe:

Periféricos de entrada e saída:

Faça uma análise das interconexões dos dispositivos selecionados por você.

Utilize os conceitos desta seção para enriquecer seus conhecimentos e desvendar a diversidade de inovações que a tecnologia proporciona. Vamos começar?

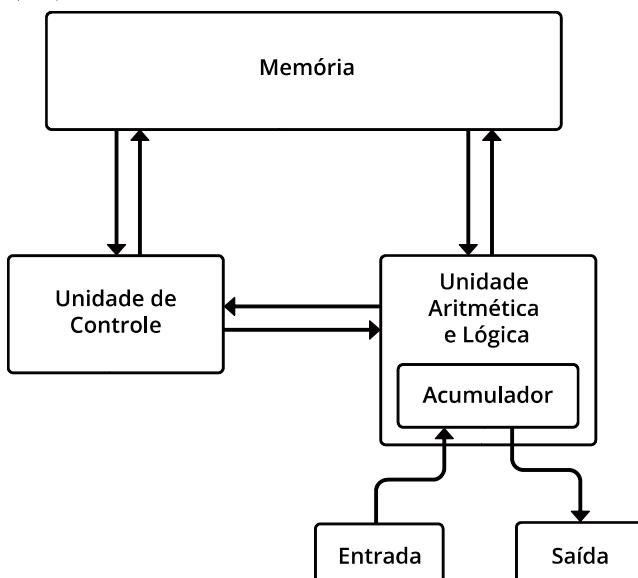
Bom trabalho e bons estudos!

Não pode faltar

A arquitetura dos computadores é resultado da evolução de vários equipamentos inventados com a finalidade de facilitar a execução de cálculos matemáticos (SOUZA FILHO; ALEXANDRE, 2014).

Os conceitos de máquinas mecânicas de cálculo foram usados em parte na teoria das máquinas universais, por Alan Turing. Após o final da Segunda Guerra, John von Neumann aperfeiçoou essas teorias e as usou na implementação da arquitetura de uma máquina digital, chamada de “Arquitetura de von Neumann”. Esta arquitetura prevê a possibilidade de uma máquina digital armazenar os programas e os dados no mesmo espaço de memória, e estes serão processados por uma unidade de processamento central (CPU) composta pela unidade de controle, memória e a unidade aritmética e lógica (ULA). Os dados são fornecidos por meio de dispositivos de entrada e retornados por dispositivos de saída (RAINER; CEGIESLK, 2012).

Figura 1.15 | Arquitetura de von Neumann



Fonte: Shutterstock.

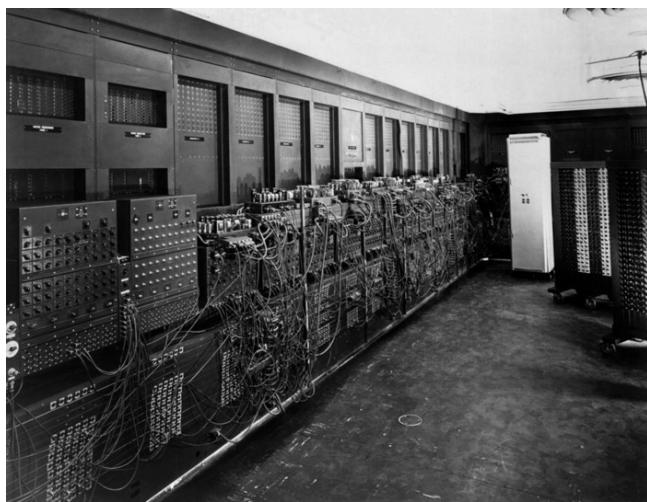
Conhecendo essa estrutura, é possível perceber claramente que as funções básicas de um computador foram pensadas a partir dela e que esta foi adotada para os computadores modernos. Agora você poderá aprofundar seus conhecimentos em cada um dos elementos que a compõe.

Unidade Central de Processamento (CPU)

A CPU (*Central Processor Unit*, ou Unidade Central de Processamento) é composta por uma Unidade Lógica Aritmética, a Unidade de Controle, que controla as unidades de memória e os dispositivos de entrada e saída do computador. Ela é responsável também por carregar e executar os programas (SOUZA FILHO; ALEXANDRE, 2014).

Quando a Segunda Guerra Mundial terminou, em 1945, os primeiros computadores começaram a ser usados comercialmente. O ENIAC, primeiro computador lançado, funcionava com válvulas colocadas em quadros interligados e não dispunha de uma CPU, por isso tinha que ser programado manualmente cada vez que fosse executar uma nova tarefa: cabos e chaves deveriam ser repositionados até que um novo programa fosse carregado. Na prática, toda a programação era feita dessa forma e só depois o computador processava as informações recebidas pela programação (ARRUDA, 2011).

Figura 1.16 | Computador ENIAC



Fonte: Shutterstock.

Um pouco depois desse período, John von Neumann introduziu a ideia de uma unidade central de processamento em um projeto de computador

chamado EDVAC, que ficou em operação entre 1949 e 1961. A arquitetura descrita e utilizada na CPU desse computador, que permitia o armazenamento de dados e programas na mesma unidade de memória pelos seus endereçamentos, deu origem aos primeiros processadores da forma como os conhecemos hoje em dia (SOUZA FILHO; ALEXANDRE, 2014).

Entre as décadas de 1960 e 1970 surgiram as CPUs, desenvolvidas em circuitos integrados, um único chip de silício, que traziam as instruções observadas pela arquitetura de von Neumann (SOUZA FILHO; ALEXANDRE, 2014). Após esse período, o mundo viu surgir os microcomputadores, dotados de processadores cada vez mais rápidos, que ganharam cada vez mais espaço e hoje são indispensáveis (ARRUDA, 2011). Essa tecnologia evoluiu rapidamente e em poucas décadas tomou proporções nunca antes imaginadas. Pode-se dizer que a tecnologia de processadores sempre foi dividida em gerações.

Um microprocessador criado pela Intel, o 4004, lançado em 1971, foi desenvolvido para o uso em calculadoras, trazia um clock máximo de 740 KHz (quilohertz) e podia calcular até 92.000 operações por segundo. Este pode ser considerado o primeiro processador aplicado a processar informações que utilizava a arquitetura de uma CPU (ARRUDA, 2011).

Já em meados da década de 1970 surgiu o processador que marcou o início dos computadores como os conhecemos hoje, ou seja, os microcomputadores da linha PC (*Personal Computer*). O processador Intel 8086 foi lançado e trazia uma tecnologia de processamento de 8 bits. Em 1970, foi lançado o processador 8088, que possuía barramento externo de 8 bits com registradores de 16 bits, e esse chip foi utilizado no IBM PC original. Pode-se dizer que essa foi a primeira geração dos microcomputadores PC (ARRUDA, 2011). A quantidade de bits de um processador representa a quantidade de informação que pode ser processada de cada vez, porém, a velocidade do processamento da informação se dá de acordo com a velocidade que o processador funciona em medida de Hertz, nos computadores atuais na casa dos Gigahertz (TANENBAUM, 2006).

A geração seguinte de processadores dobrou a capacidade de processamento, ou seja, os computadores dessa geração funcionavam com 32 bits. Esta arquitetura foi usada por um número muito grande de modelos de microcomputadores, sendo chamada de x86 de 32 bits; x86 porque foram sempre uma evolução do processador 8086 inicial e deram vida a processadores conhecidos, como o 286, 386, 486, Pentium I, II, III e IV, Pentium Celeron e outros. Também podemos dizer que os processadores atuais são o resultado da evolução desses processadores (ARRUDA, 2011).

No final da década de 1990 e começo dos anos 2000, os processadores de 32 bits tinham a capacidade de endereçamento de memória de no máximo 4 GB de memória RAM. Esta capacidade é determinada pelo número de bits do processador e quantos endereços podem ser conseguidos com esses bits. No caso de 32 bits, conseguimos um pouco mais de 4 bilhões de endereços, representados por 4 GB. Novas tecnologias estavam sendo lançadas com mais capacidade de processamento, o que levou a uma evolução natural para processadores de 64 bits, nos quais podem ser gerenciados aproximadamente 16 PB (petabytes) de endereços de memória possíveis. (VELLOSO, 2011)

A empresa AMD foi pioneira e lançou um processador de 64 bits que funcionava muito melhor do que as soluções apresentadas até então pelo seu maior concorrente, a Intel, que tinha uma grande vantagem sobre a tecnologia de 32 bits. O modelo da AMD foi adotado como modelo para a arquitetura de 64 bits, resultado de um acordo feito entre esses dois fabricantes, AMD e Intel, pelo qual a AMD licenciou a tecnologia de 64 bits para uso da Intel e, em contrapartida, a Intel licenciou a tecnologia de 32 bits para a AMD, o que contribuiu para que ambas dominassem juntas o mercado desse período. Um modelo que marcou a tecnologia de 64 bits foi o processador ATHLON 64 da AMD, líder nos microcomputadores da época (ARRUDA, 2011).

Você pode observar que a cada dia novos componentes surgem e novas tecnologias são inventadas, e essa realidade é observada mais intensamente nos computadores. Como não poderia ser diferente, o próximo passo nesta evolução foi a necessidade de aumentar a velocidade dos processadores, e por limitações técnicas, principalmente pelo calor gerado pelos chips de processadores rápidos, isso não estava sendo possível. A solução encontrada para essa limitação foi colocar dentro de um único chip mais de um núcleo de processamento, ou seja, mais de um processador. Essa tecnologia foi chamada de Multicore, possibilitando um aumento de capacidade de processamento sem a necessidade de aumentar as velocidades de cada núcleo. A ideia foi aumentar o número de núcleos, ampliando assim a capacidade final de processamento (ARRUDA, 2011).

Veremos a seguir os processadores recentemente lançados no mercado. Essas informações podem mudar a qualquer momento, e cabe a você ficar atento às evoluções dos novos processadores e suas principais tecnologias.

Quadro 1.2 | Linha dos principais processadores da Intel para 2020 para Desktop

Intel® Core™	Intel® Xeon®	Intel Atom®
Intel® Core™ Série X Intel® Core™ i9 da 9ª geração Intel® Core™ i7 da 10ª geração Intel® Core™ i5 da 10ª geração Intel® Core™ i3 da 10ª geração Intel® Core™ m3 da 8ª geração Intel® Core™ vPro™ da 9ª geração	Intel® Xeon® escalável Intel® Xeon® D Intel® Xeon® W Intel® Xeon® E	Intel Atom® C Intel Atom® E Intel Atom® X

Fonte: Intel (2020a, [s.p.]).

Quadro 1.3 | Linha dos principais processadores da AMD para 2020 para Desktop

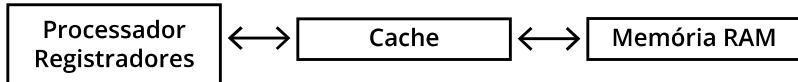
Processador AMD Ryzen™ Threadripper™	Processadores AMD Athlon™ com placa de vídeo Radeon™ Vega	Processadores AMD A-Series™ com gráficos Radeon™
Processador AMD Ryzen™ Threadripper™ 3990X AMD Ryzen™ Threadripper™ 3970X Processor AMD Ryzen™ Threadripper™ 3960X Processor AMD Ryzen™ Threadripper™ 2990WX Processor AMD Ryzen™ Threadripper™ 2970WX AMD Ryzen™ Threadripper 2950X Processor	Processador AMD Athlon™ 240GE com placa de vídeo Radeon™ Vega 3 Processador AMD Athlon™ 220GE com placa de vídeo Radeon™ Vega 3 Processador AMD Athlon™ 200GE com placa de vídeo Radeon™ Vega 3	7th Gen A12-9800 APU 7th Gen A12-9800E APU 7th Gen A10-9700 APU 7th Gen A10-9700E APU 7th Gen A8-9600 APU 7th Gen A6-9550 APU

Fonte: AMD (2020, [s.p.]).

Memórias

De forma simples vamos iniciar o assunto sobre memórias descrevendo sobre os registradores e cache. Os **registradores** são circuitos lógicos que fazem parte da CPU (processador), são memórias que armazenam e destinam todas as informações binárias que chegam para serem processadas (calculadas). Essas memórias são consideráveis voláteis, ou seja, de armazenamento temporário. Para intermediar os registradores e as memórias RAM (*Random Access Memory* – Memória de Acesso Aleatório), encontramos as memórias cache, que ficam próximas à CPU. Extremamente mais rápidas que as memórias RAM, as memórias **cache** foram desenvolvidas para armazenar e distribuir rapidamente os dados para o registrador, e devolvê-los com maior velocidade.

Figura 1.17 | Fluxo de dados entre: Registrador – Cache – Memória RAM



Fonte: adaptada de Stallings (2017).

A memória RAM possibilita aos processadores endereçar dados divididos em regiões distintas, usadas pelo sistema operacional da máquina, verificar informações de dispositivos de entrada e saída, de programas do usuário e dados gerados por esses programas. A capacidade de administrar a quantidade de memória RAM cresceu a cada geração de processador. Nos processadores de 32 bits era possível o endereçamento de no máximo 4 GB de memória RAM, e somente nos processadores de 64 bits passou a ser possível lidar com quantidades maiores de memória (SOUZA FILHO; ALEXANDRE, 2014).

Figura 1.18 | Memória RAM



Fonte: Intel (2020b, [s.p.]).

Hoje podemos contar com memórias as seguintes características:

Capacidade: 8GB

- Dual/Quad Channel.
- Tipo: DDR4.
- Formato: DIMM.
- Desempenho do perfil: XMP 2.0.
- Velocidade testada: 4000Mhz.
- Voltagem: 1.35V.
- Classificação de velocidade: PC4-32000 (4000MHz).

Refita

O mercado oferece uma diversidade de modelos e velocidades de memória. Cabe lembrar que as memórias trabalham em conjunto com as placas mães e processadores. Sendo assim, será necessário verificar as especificações e compatibilidade das memórias?

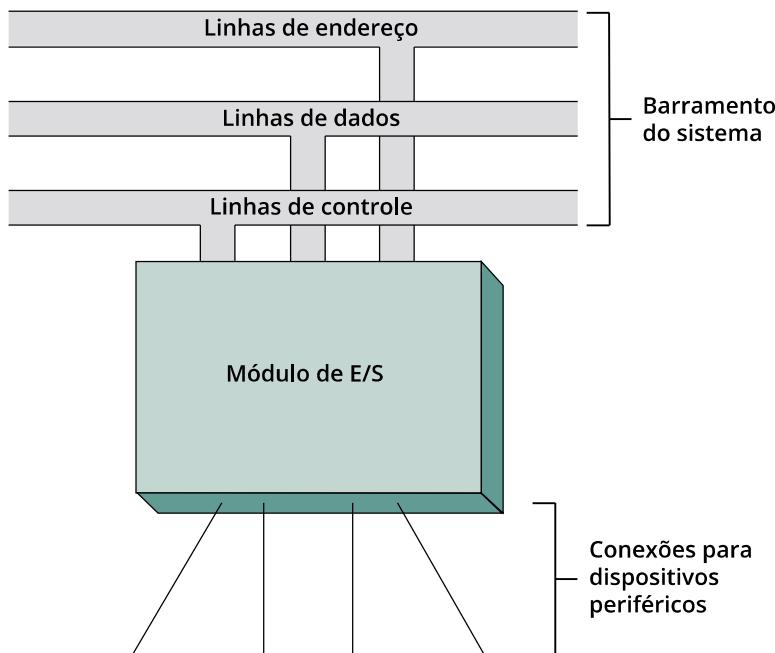
Entrada e Saída (E/S)

Segundo Stallings (2017, p. 194)

As operações de E/S são realizadas por meio de uma grande variedade de dispositivos externos, que oferecem um meio de trocar dados entre o ambiente externo e o computador. Um dispositivo externo conecta-se ao computador por uma conexão com um módulo de E/S.

A figura a seguir demonstra essa afirmação.

Figura 1.19 | Modelo genérico de um módulo de E/S



Fonte: Stallings (2017, p. 194).

Os dispositivos externos ligados ao módulo de E/S são chamados de periféricos, e podem ser classificados em três categorias, segundo Stallings (2017, p. 195):

Inteligíveis ao ser humano: adequados para a comunicação com usuários de computador. Por exemplo: monitores e impressora, entre outros.

Inteligíveis à máquina: adequados para a comunicação com equipamentos. Por exemplo: disco e fita magnética, sensores e atuadores, como aqueles usados em uma aplicação de robótica.

Comunicação: adequados para a comunicação com dispositivos remotos. Por exemplo: comunicação entre computadores, como é o caso dos terminais.

Em contexto geral para ao nosso primeiro contato com a E/S do computador, podemos dizer que o teclado e o monitor caracterizam bem essa interface homem/computador e vice-versa, em que o usuário, por meio do teclado, fornece instruções necessárias para o processamento, e o monitor exibe os dados de resposta do computador.

Cabe lembrar que o módulo de E/S, segundo Stallings (2017), tem a função de controle e temporização, comunicação com o processador, comunicação com o dispositivo, *buffering* de dados e detecção de erro.

Interconexão

Finalizando essa jornada de conhecimento, vale destacar que para ter comunicação entre os componentes do computador (processador, memória e E/S) será necessária a interconexão entre eles.

Segundo Stallings (2017), a estrutura de interconexão deve admitir os seguintes tipos de transferências:

- Memória para processador: o processador lê uma instrução ou uma unidade de dados da memória.
- Processador para memória: o processador escreve uma unidade de dados na memória.
- E/S para processador: o processador lê dados de um dispositivo de E/S por meio de um módulo de E/S.
- Processador para E/S: o processador envia dados para o dispositivo de E/S.

- E/S de ou para a memória: para esses dois casos, um módulo de E/S tem permissão para trocar dados diretamente com a memória, sem passar pelo processador, usando o DMA.

Vale destacar, segundo Stallings (2017), que o barramento é o caminho de comunicação que conecta dois ou mais dispositivos, sendo ele o meio da interconexão de componentes de sistema do computador.

Assimile

Segundo Stalling (2017), o barramento é constituído de vários caminhos de comunicação, também chamados de linhas de comunicação. Essas linhas são capazes de transmitir sinais representando o binário 0 e 1. Segundo Stalling (2017, p. 82),

Com o tempo, uma sequência de dígitos binários pode ser transmitida por uma única linha. Juntas, várias linhas de um barramento podem ser usadas para transmitir dígitos binários simultaneamente (em paralelo).

Por exemplo, uma unidade de dados de 8 bits pode ser transmitida por oito linhas de barramento.

Sendo assim, o barramento é um meio de transmissão compartilhado, em que os dispositivos se conectam, podendo, assim, ocorrer a transmissão dos dispositivos um de cada vez, para não ocorrer a sobreposição e os dados ficarem distorcidos.

Muito bem, você terminou esta seção e teve a oportunidade de conhecer a Unidade Central de Processamento (CPU), a memória principal, os dispositivos de E/S e os Sistemas de interconexão.

Bons estudos!

Sem medo de errar

Você está se preparando para participar de um processo seletivo que aplicará testes de conhecimento sobre arquitetura e organização dos computadores. Será necessário que você conheça a arquitetura dos computadores, seus processadores, como estes administram a quantidade de memória

do computador, os dispositivos de entrada e saída e como se conectam a uma rede.

Agora, faça um relatório das principais configurações de computadores que deverão ser adquiridos pela empresa.

Exemplo:

- Processador Intel Core i7
- Memória RAM de 16 GB
- HD de 1 TB
- SSD de 8 GB
- Placa de vídeo GeForce GTX 1060 de 6 GB
- Sistema Operacional Windows 10

Periféricos de entrada e saída: teclado e mouse wireless, impressora a laser, fone wireless.

Faça uma análise das interconexões dos dispositivos selecionados por você.

Sugestão de pesquisa para elaboração do relatório: sites de venda de equipamentos de computação e dos principais fabricantes de componentes.

Bom trabalho e bons estudos!

Faça valer a pena

1. Os **registradores** são circuitos lógicos que fazem parte da CPU (processador), são memórias que armazenam e destinam todas as informações binárias que chegam para serem processadas (calculadas). Essas memórias são consideráveis voláteis, ou seja, de armazenamento temporário.

Assinale a alternativa que define a função da memória cache.

- a. As memórias cache foram desenvolvidas para armazenar e distribuir rapidamente os dados para o registrador e devolvê-los com maior velocidade.
- b. Define-se como memória cache a memória que fica posicionado no lado externo da CPU e que tem a função de guardar as informações não voláteis.
- c. Cache é definida como sendo a memória mais importante do computador, na qual são processados todos os seus cálculos.

- d. As memórias cache têm a função de organizar os arquivos criados pelos sistemas operacionais.
- e. A memória cache têm acesso direto aos registradores e processam os dados do sistema operacional sem passar pela memória RAM.

2. Segundo Stallings (2017, p. 194), “As operações de E/S são realizadas por meio de uma grande variedade de dispositivos externos, que oferecem um meio de trocar dados entre o ambiente externo e o computador.”

Os dispositivos externos ligados ao módulo de E/S são chamados de periféricos, e podem ser classificados em três categorias. Avalie as afirmações:

- I. Inteligíveis ao ser humano: adequados para a comunicação com usuários de computador. Por exemplo: monitores e impressora, entre outros.
- II. Inteligíveis à máquina: adequados para a comunicação com dispositivos remotos. Por exemplo: comunicação entre computadores, como é o caso dos terminais.
- III. Comunicação: adequados para a comunicação com equipamentos. Por exemplo: disco e fita magnética, sensores e atuadores, como aqueles usados em uma aplicação de robótica.

Considerando o contexto apresentado, é correto o que se afirma em:

- a. I, II e III.
- b. I, apenas.
- c. II, apenas.
- d. I e II, apenas.
- e. II e III, apenas.

3. Para ter comunicação entre os componentes do computador (processador, memória e E/S) será necessária a interconexão entre eles.

Assinale V (verdadeiro) e F (Falso) para os tipos de transferências de interconexão:

- () Memória para processador: o processador lê dados de um dispositivo de E/S por meio de um módulo de E/S.
- () Processador para memória: o processador lê uma instrução ou uma unidade de dados da memória.
- () E/S para processador: o processador escreve uma unidade de dados na memória.

() Processador para E/S: o processador envia dados para o dispositivo de E/S.

() E/S de ou para a memória: para esses dois casos, um módulo de E/S tem permissão para trocar dados diretamente com a memória, sem passar pelo processador, usando o DMA.

Assinale a alternativa que apresenta a sequência correta.

- a. V – F – V – V – V.
- b. V – F – V – F – V.
- c. F – F – F – V – F.
- d. F – F – F – V – V.
- e. V – F – F – F – F.

Seção 4

A hierarquia de níveis de computador

Diálogo aberto

Quando você começa a ver as diversas tecnologias digitais que existem hoje e como elas foram pensadas, percebe vários pontos em comum entre elas. Isso acontece porque os computadores são produzidos baseados em uma arquitetura que foi criada há anos e que vem passando por constante evolução, mantendo, porém, seus pontos principais praticamente da mesma forma. Você também já viu que existem vários tipos diferentes de computadores, desktops, notebooks, tablets, smartphones, consoles de games, e em todos você pode notar muitas semelhanças, como telas, memórias RAM, discos rígidos, pen drives, cartões de memória, teclados padrão ou telas de touch screen que permitem a digitação, conexões com internet via cabo ou via wi-fi, e muitos outros dispositivos que se conectam aos computadores e auxiliam no seu uso.

Essa arquitetura foi proposta por John von Neumann, matemático húngaro radicado e naturalizado nos Estados Unidos da América e envolvido com o desenvolvimento dos primeiros computadores usados logo após a Segunda Guerra Mundial. Ela é chamada de Arquitetura de von Neumann e tem servido como base para as novas tecnologias.

Ao aprofundar seus conhecimentos nesta arquitetura você conhecerá melhor o modelo proposto por von Neumann: uma Unidade Central de Processamento (CPU) e suas unidades principais, a unidade de controle e a unidade lógica aritmética, suas memórias e também as unidades de entrada e saída. Quanto mais você conhecer sobre essa estrutura, mais entenderá como os computadores são montados e como funcionam.

Nesta etapa, você terá que identificar, de forma comparativa, as vantagens e as desvantagens da Arquitetura de von Neumann. Deverá listar quais as unidades previstas por essa arquitetura e qual a função delas. Deverá ainda citar outros tipos de arquiteturas de computação. Organize uma pequena planilha com esses pontos e demonstre, dessa forma, seus conhecimentos sobre a arquitetura dos computadores.

O entendimento desses conceitos é de extrema importância e será usado por você no processo seletivo da empresa de desenvolvimento de tecnologia para computadores de última geração, que vai ampliar sua fábrica no Brasil. Lembre-se: ao final desse processo, serão aplicados vários testes e contratados os candidatos com maior nota.

Aprofunde cada vez mais seus conhecimentos.

Bom trabalho e bons estudos!

Não pode faltar

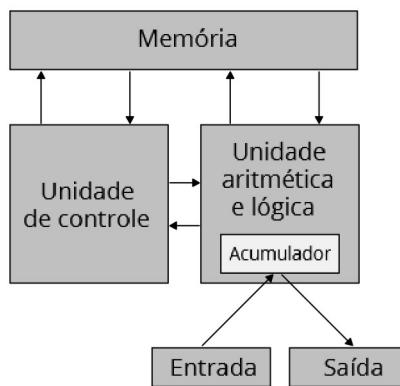
A estrutura dos primeiros computadores era limitada e sua programação complexa, como você já deve ter visto. Essas máquinas funcionavam com válvulas colocadas em quadros interligados e não dispunham de uma CPU, tendo que ser programadas manualmente cada vez que fossem executar uma nova tarefa. Na prática, toda a programação era feita reposicionando cabos e chaves até que um novo programa fosse carregado e só depois o computador processava as informações recebidas por essa programação (ARRUDA, 2011).

Após o final da Segunda Guerra, John von Neumann implementou a arquitetura de uma máquina digital, chamada de Arquitetura de von Neumann. Esta arquitetura prevê a possibilidade de uma máquina digital armazenar os programas e os dados no mesmo espaço de memória e estes serão processados por uma unidade de processamento central (CPU), composta por uma unidade de controle e uma unidade aritmética e lógica (ULA). Os dados são fornecidos pelos dispositivos de entrada e retornados pelos dispositivos de saída (RAINER; CEGIESLK, 2012).

Mas você deve estar se perguntando: como essa arquitetura, que é usada até hoje nos computadores, conseguiu estabelecer um padrão aceitável para que as máquinas pudessem processar informações? Vejamos como isso foi pensado.

A descrição da arquitetura de von Neumann prevê cinco unidades distintas: unidade aritmética, unidade de controle, unidade memória, unidade de entrada e unidade de saída lógica, como pode ser observado na Figura 1.20:

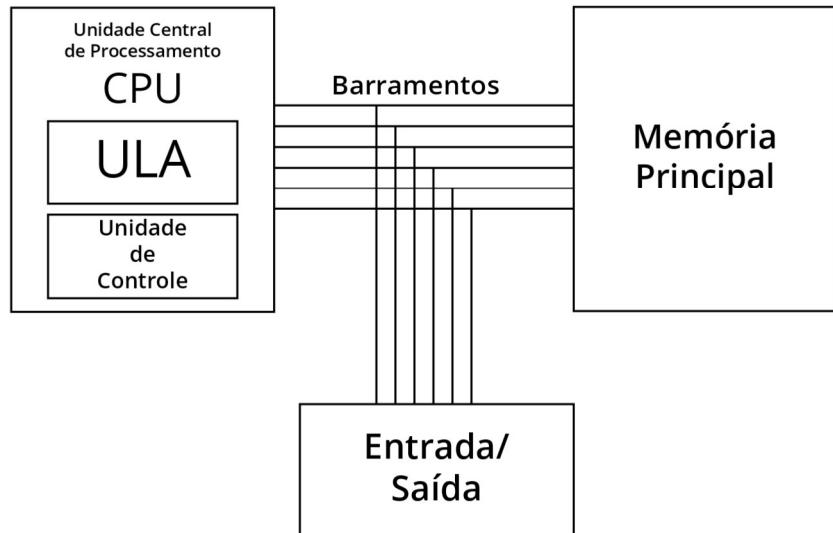
Figura 1.20 | Arquitetura de John von Neumann



Fonte: Wikimedia Commons.

Cada uma dessas unidades tem sua função no processamento e controle das demais unidades do computador. Os barramentos, que são as vias por onde passam os dados, permitem a transmissão de informações entre a CPU, os dispositivos de entrada e saída de dados e as unidades de memória (OKUYAMA; MILETTO; NICOLAU, 2014).

Figura 1.21 | CPU, Memórias, E/S e Barramentos



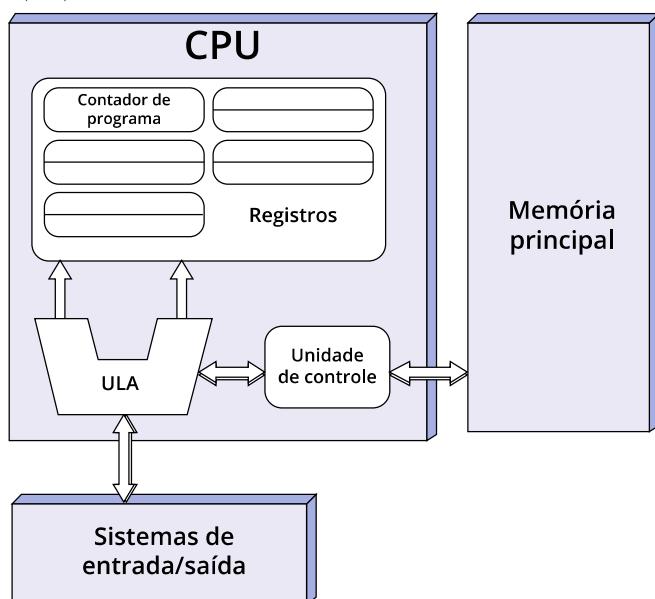
Fonte: adaptada de Souza Filho e Alexandre (2014).

Pode-se verificar também que essa estrutura lógica constitui o funcionamento dos computadores digitais, inclusive dos computadores mais modernos usados atualmente (FONSECA FILHO, 2007).

Você já viu que os computadores têm a mesma forma de lidar com as informações. O computador recebe as informações pela unidade de entrada e de seus dispositivos, a CPU processa essas informações e retorna o resultado deste processamento por meio da unidade de saída e de seus dispositivos (RAINER; CEGIESLK, 2012). As informações são convertidas pelo processador em sistema binário (0 e 1) no momento da entrada de dados e convertidas para o sistema alfanumérico usado por nós, usuários, no momento da saída desses dados. Essas informações são armazenadas nas memórias do computador e são usadas para processamento, com a finalidade de retornar resultados por meio das unidades de saída ou até para serem gravadas em dispositivos de armazenamento de memória, como discos rígidos (RAINER; CEGIESLK, 2012).

A unidade lógica e aritmética (ULA) é responsável por executar os cálculos matemáticos utilizados para processar os dados dentro do computador. Dependendo dos resultados desses cálculos, diferentes ações podem acontecer, considerando cada programa que estiver sendo executado naquele momento (OKUYAMA; MILETTO; NICOLAU, 2014).

Figura 1.22 | Arquitetura de von Neumann – ULA



Fonte: adaptada de Wikimedia Commons.

Já a unidade de controle de um processador tem a função de coordenar e direcionar as principais funções de um computador, como o processador vai enviar e receber os dados para as memórias, interpretar cada função contida em um programa e depois iniciar a ação que execute essa função. Essa unidade é a responsável por toda a ordenação de dados de um computador e até pelo funcionamento do próprio computador, pois coordena a ULA, os registradores que controlam as memórias, os barramentos internos que se comunicam com as memórias e todo o funcionamento da placa-mãe, além da interligação dos dispositivos nela inseridos (FONSECA FILHO, 2007).

Refita

Os computadores evoluíram e suas gerações foram classificadas de acordo com o tipo de processador. Você já percebeu que quando nos referimos a algum tipo de computador, falamos de seu processador? É muito comum falar que tal computador é um i3 com "X" quantidade de RAM e "X" quantidade de HD (*hard disc*) (EQUIPE DIGERATI, 2008).

A memória é o espaço que recebe as informações para serem processadas e, após seu processamento, para serem enviadas aos dispositivos de saída. Esse espaço é composto por registradores que são endereçados, ou seja, são os espaços de memória que recebem os dados e são divididos de acordo com a função, como espaço para o sistema operacional, controle de dispositivos de entrada e saída, espaço para a execução de programas e para os dados a serem processados e retornados após o processamento.

Um espaço de memória pode conter uma instrução de um programa ou um dado qualquer, que será endereçado na memória pela unidade de controle da CPU. Os dados que serão processados pela ULA ficam na memória e a unidade de controle endereça estes dados. Isso permite que a ULA identifique onde estão os dados a serem processados, execute as operações necessárias, e a unidade de controle pode definir onde armazenar os dados resultantes do processamento. A memória que recebe esse endereçamento e é usada para receber as informações da unidade de entrada e as processadas pelo computador é a memória RAM (SOUZA FILHO; ALEXANDRE, 2014).

Nessa arquitetura de computadores estão previstas também as unidades de entrada e saída de dados. Como você já deve ter visto, estas unidades são compostas por diversos dispositivos e podem ser divididos em (SOUZA FILHO; ALEXANDRE, 2014):

- **Dispositivos de Entrada:** nos quais podemos inserir/entrar com dados no computador. Exemplo: teclado, mouse, telas sensíveis ao toque (*touch screen*).

- **Dispositivos de Saída:** em que os dados podem ser visualizados. Exemplo: telas e impressoras.
- **Dispositivos de Entrada/Saída:** são dispositivos que podem enviar e receber dados, como o disco rígido, pen drives, as conexões de internet via cabo e wi-fi, monitores e telas *touch screen*, dentre outros (FONSECA FILHO, 2007).

Assimile

Segundo Sebesta (2010, p. 39),

[...] a execução de um programa em código de máquina em uma arquitetura de computadores von Neumann ocorre em um processo chamado de ciclo de obtenção e execução. Conforme mencionado, os programas residem na memória, mas são executados na CPU. Cada instrução a ser executada deve ser movida da memória para o processador.

O gargalo de von Neumann

A via de transmissão de dados entre a CPU e a memória limita de certa forma a velocidade do processamento de um computador. Os barramentos têm esta função, e a troca de dados entre o processador e a memória fica limitada pela taxa de transferência de dados que esses barramentos são capazes de proporcionar, que em geral são bem menores que a capacidade dos processadores, sendo um fator limitador da velocidade atingida no processamento das informações. Esse problema aumenta a cada nova geração e o desenvolvimento de tecnologia com maior número de barramentos é uma das soluções adotadas pelos fabricantes de tecnologia (TANENBAUM, 2006).

Hierarquia de níveis

Para que programas e dados sejam processados, foi criada uma organização em uma hierarquia de níveis de forma hipotética, ou seja, essa hierarquia foi pensada para poder classificar as etapas do processamento que acontece dentro de um computador. Nessa hierarquia temos o nível mais alto, que é percebido pelo usuário e no qual são mostrados os programas e os dados, e os demais são executados internamente pelo computador (NULL; LOBUR, 2011).

Nível 6 – Usuário.	Programas executáveis.
Nível 5 – Linguagem de alto nível.	C++, Java, FORTRAN etc.
Nível 4 – Linguagem do montador.	Assembler.
Nível 3 – Sistema operacional.	Sistemas de comandos ou de janelas.
Nível 2 – Arquitetura do conjunto de instruções.	Arquitetura do conjunto de instruções.
Nível 1 – Microarquitetura.	Microcódigo implementado em hardware.
Nível 0 – Lógica digital.	Circuitos e barramentos, entre outros.

Fonte: adaptado de Null e Lobur (2011).

Veja a seguir as características de cada nível hierárquico do computador:

Nível 0: nível da lógica digital – também considerado o nível mais baixo da estrutura do computador, é nele em que são estruturadas as portas lógicas, que apresentam uma ou mais entradas digitais, realizam as funções lógicos (and, or, xor etc.). Neste nível podemos caracterizar o dispositivo mais importante do computador, o processador.

Nível 1: nível da microarquitetura – caracterizado por uma memória local (8 a 32 registradores) e a UAL (Unidade Aritmética Lógica) para realização de operações aritméticas. As operações são controladas por um microprograma, que interpreta as instruções do Nível 2, buscando, decodificando e executando as instruções uma de cada vez.

Nível 2: nível da arquitetura do conjunto de instruções – é considerado o nível da *Instruction Set Architecture* ou ISA. São características definidas pelos fabricantes, que disponibilizam manuais de referência da linguagem de máquina. Tais manuais têm a função de descrever como as instruções são executadas pelo microprograma ou executadas diretamente pelo hardware. Todas essas informações são necessárias para os desenvolvedores de SO (sistemas operacionais).

Nível 3: nível do sistema operacional – este nível também pode conter instruções do nível ISA, porém suporta organização diferente de memória. Tem a característica de executar dois ou mais programas ao mesmo tempo, suportando sistemas de comandos ou de janelas (*windows*).

Neste nível os desenvolvedores são chamados de programadores de sistema, e no nível mais baixo e no nível mais alto são chamados programadores de aplicação.

Nível 4: nível da linguagem do montador – direcionadas para a linguagem de montagem (*Assembly language*), quando é realizada a tradução/montador.

Nível 5: nível das linguagens orientadas para solução dos problemas – são as linguagens de alto nível, como linguagem C, Python e Java, entre outras. Essas linguagens podem ser compiladas (como a linguagem C) ou interpretadas (como a Java).

Nível 6: usuário – é o nível considerado do usuário, em que são utilizadas as aplicações que todos conhecemos: os editores de textos, planilhas, aplicativos de músicas, vídeos e jogos, entre muitos outros.

Máquinas com arquiteturas diferentes da arquitetura de von Neumann

Embora os computadores tenham seguido a arquitetura proposta por von Neumann, existem máquinas que computam dados e que não foram construídas usando essa arquitetura. Entre essas máquinas encontramos computadores analógicos, computadores com múltiplos processadores funcionando em paralelo e executando programas de forma cooperativa – ou seja, um programa sendo executado por mais de um processador –, redes neurais artificiais, usadas principalmente em desenvolvimento de sistemas que envolvam inteligência artificial, e máquinas de fluxos de dados, que realizam suas operações com os dados disponibilizados no momento do processamento, não havendo, nesse caso, uma programação feita antecipadamente (TANENBAUM, 2006).

Muito bem, chegamos ao final de mais uma unidade de estudo. Nesta seção, você teve a oportunidade de conhecer as hierarquias de níveis de computador e o modelo de von Neumann.

Continue firme e ótimos estudos!

Sem medo de errar

Você está se preparando para participar de um processo seletivo em que serão aplicados testes de conhecimento sobre a arquitetura e organização dos computadores. Será necessário que você conheça a arquitetura dos computadores de acordo com a arquitetura de von Neumann, suas unidades e como funciona o processamento de informações de acordo com essa arquitetura.

Neste momento você terá que identificar, de forma comparativa, as vantagens e as desvantagens da arquitetura de von Neumann. Deverá listar quais as unidades previstas por essa arquitetura e qual a função delas. Deverá, ainda, citar outros tipos de arquiteturas de computação. Organize uma pequena planilha com esses pontos e demonstre, dessa forma, seus conhecimentos sobre a arquitetura dos computadores.

Quadro 1.5 | Vantagens e desvantagens da arquitetura de von Neumann

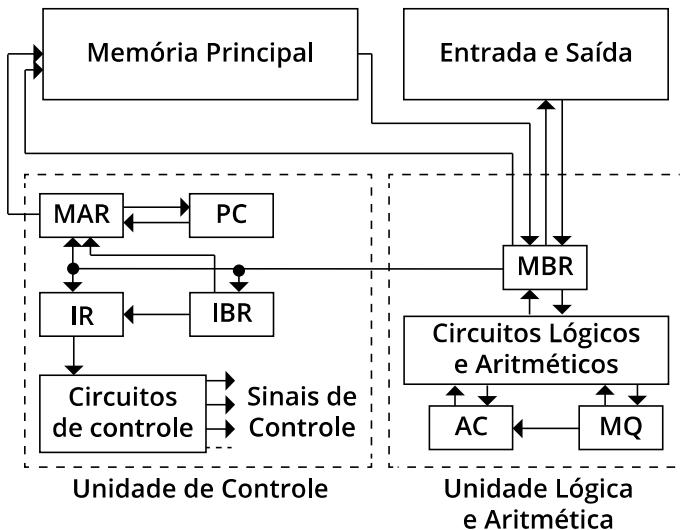
Pesquisa: Arquitetura de von Neumann	
Vantagens	Essa arquitetura prevê a possibilidade de uma máquina digital armazenar os programas e os dados no mesmo espaço de memória, os quais serão processados por uma unidade de processamento central (CPU), composta por uma unidade de controle e uma unidade aritmética e lógica (ULA). Os dados são fornecidos pelos dispositivos de entrada e retornados por meio dos dispositivos de saída.
Desvantagens	A via de transmissão de dados entre a CPU e a memória limita, de certa forma, a velocidade do processamento de um computador. Os barramentos têm essa função, e a troca de dados entre o processador e a memória fica limitada pela taxa de transferência de dados que esses barramentos são capazes de proporcionar.
Sua estrutura – unidades	Memória, CPU, Unidades de Entrada/Saída.
Função da CPU	Processar os dados.
Função da ULA	A unidade lógica e aritmética (ULA) é responsável por executar os cálculos matemáticos utilizados para processar os dados dentro do computador. Dependendo dos resultados desses cálculos, diferentes ações podem acontecer, considerando cada programa que estiver sendo executado naquele momento.
Função da Unidade de Controle	Esta unidade é a responsável por toda a ordenação de dados de um computador e até pelo funcionamento do próprio computador, pois coordena a ULA, os registradores que controlam as memórias, os barramentos internos que se comunicam com as memórias e todo o funcionamento da placa-mãe, além da interligação dos dispositivos nela inseridos.
Função dos Barramentos	Os barramentos são as vias por onde passam os dados e que permitem a transmissão de informações entre a CPU, os dispositivos de entrada e saída de dados e as unidades de memória.
Descrição do gargalo de von Neumann	A via de transmissão de dados entre a CPU e a memória limita de certa forma a velocidade do processamento de um computador. Os barramentos têm essa função, e a troca de dados entre o processador e a memória fica limitada pela taxa de transferência de dados que esses barramentos são capazes de proporcionar.

Máquinas não von Neumann	Existem máquinas que computam dados e que não foram construídas usando a arquitetura de von Neumann. Entre elas encontramos computadores analógicos, computadores com múltiplos processadores funcionando em paralelo, redes neurais artificiais e máquinas de fluxos de dados.
--------------------------	---

Fonte: elaborado pelo autor.

Computador base para essa descrição: Computador IAS, baseado na arquitetura do ENIAC.

Figura 1.23 | Descrição de como as informações são processadas na arquitetura de von Neumann



Fonte: Borin e Auler (2013, p. 22).

Bons estudos!

Faça valer a pena

1. A descrição da arquitetura de von Neumann prevê cinco unidades distintas: unidade aritmética, unidade de controle, unidade memória, unidade de entrada e unidade de saída lógica.

Assinale a alternativa que define a função da ULA (unidade lógica e aritmética).

- A unidade lógica e aritmética (ULA) é responsável por executar os cálculos aritméticos e lógicos.
- A ULA tem a função de transportar as informações para os registradores do processador.
- A unidade aritmética é responsável por executar instruções diretamente da memória RAM, deixando o processamento dos dados mais rápido.
- A unidade de controle utiliza a ULA como recurso mais rápido, evitando, assim, não utilizar os registradores do processador.
- A unidade aritmética é responsável por realizar o armazenamento das informações do processador.

2. A unidade de controle de um processador tem a função de coordenar e direcionar as principais funções de um computador, como o processador vai enviar e receber os dados para as memórias, interpretar cada função contida em um programa e depois iniciar a ação que execute essa função.

Considerando o contexto apresentado, avalie as seguintes asserções e a relação proposta entre elas:

- Essa unidade é a responsável por toda a ordenação de dados de um computador e até pelo funcionamento do próprio computador

PORQUE

- Coordena a ULA, os registradores que controlam as memórias, os barramentos internos que se comunicam com as memórias e todo o funcionamento da placa-mãe, além da interligação dos dispositivos nela inseridos.

A respeito dessas asserções, assinale a alternativa correta.

- As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- As asserções I e II são proposições verdadeiras e a II justifica a I.
- A asserção I é uma proposição verdadeira e a II, falsa.
- A asserção I é uma proposição falsa e a II, verdadeira.

- e. As asserções I e II são proposições falsas.
- 3.** Para que programas e dados sejam processados, foi criada uma organização em uma hierarquia de níveis de forma hipotética, ou seja, essa hierarquia foi pensada para poder classificar as etapas do processamento que acontece dentro de um computador.

Com base nas informações disponíveis, associe a Coluna A, que apresenta esses componentes, com a Coluna B, que apresenta suas propriedades.

Quadro | Hierarquia de níveis

Nível 6 – Usuário.	I – Programas executáveis.
Nível 5 – Linguagem de alto nível.	II – C++, Java, FORTRAN etc.
Nível 4 – Linguagem do montador.	III – Conjunto de instruções.
Nível 3 – Sistema operacional.	IV – Sistemas de comandos ou de janelas.
Nível 2 – Arquitetura do conjunto de instruções.	V – Assembler.
Nível 1 – Microarquitetura.	VI – Circuitos e barramentos, entre outros.
Nível 0 – Lógica digital.	VII – Microcódigo implementado em hardware.

Assinale a alternativa que apresenta a associação correta entre as colunas.

- a. Nível 6 – II; Nível 5 – I; Nível 4 – III; Nível 3 – IV; Nível 2 – V; Nível 1 – VII; Nível 0 – VI.
- b. Nível 6 – II; Nível 5 – IV; Nível 4 – V; Nível 3 – VI; Nível 2 – VII; Nível 1 – III; Nível 0 – I.
- c. Nível 6 – I; Nível 5 – II; Nível 4 – III; Nível 3 – VI; Nível 2 – V; Nível 1 – VI; Nível 0 – VII.
- d. Nível 6 – I; Nível 5 – II; Nível 4 – III; Nível 3 – IV; Nível 2 – VII; Nível 1 – V; Nível 0 – VI.
- e. Nível 6 – I; Nível 5 – II; Nível 4 – V; Nível 3 – IV; Nível 2 – III; Nível 1 – VII; Nível 0 – VI.

Referências

- ADVANCED MICRO DEVICES INC. **Processador AMD Ryzen™ Threadripper™ 3990X.** AMD, 2020, [s.p.]. em: <https://www.amd.com/pt/products/cpu/amd-ryzen-threadripper-3990x>. Acesso em: 24 mar. 2020.
- ALMEIDA, M. **Curso de montagem e manutenção de micros.** São Paulo: Digerati Books, 2007.
- ARRUDA, F. **A história dos processadores.** TecMundo, 16 jun. 2011. Disponível em: <https://www.tecmundo.com.br/historia/2157-a-historia-dos-processadores.htm>. Acesso em: 26 mar. 2020.
- BOLTON, W. **Mecatrônica, uma abordagem multidisciplinar.** 4. ed. São Paulo: Bookman, 2010.
- BORIN, E.; AULER, R. Uma abordagem para o ensino de linguagem de montagem, arquitetura e organização de computadores. **International Journal of Computer Architecture Education (IJCAE),** v. 2, n. 1, dez. 2013, p. 21-24. Disponível em: http://www2.sbc.org.br/ceacpad/ijcae/v2_n1_dec_2013/IJCAE_v2_n1_dez_2013_paper_6_yf.pdf. Acesso em: 26 mar. 2020.
- DELGADO, J. **Arquitetura de computadores.** 5. ed. atual. Rio de Janeiro: LTC, 2017.
- EQUIPE DIGERATI BOOKS. **Guia prático de hardware.** São Paulo: Digerati Books, 2008.
- FONSECA FILHO, C. **História da computação:** o caminho do pensamento e da tecnologia. [recurso eletrônico]. Porto Alegre: PUCRS, 2007.
- FUNDAÇÃO BRADESCO. **Resumo da evolução dos computadores.** Microinformática, [s.d.]. Disponível em: http://www.fundacaobradesco.org.br/vv-apostilas/mic_pag3.htm. Acesso em: 23 mar. 2020.
- HENNESSY, J. L.; PETTERSON, D. A. **Arquitetura de computadores:** uma abordagem quantitativa. 5. ed. Rio de Janeiro: Elsevier, 2014.
- INTEL CORPORATION. **Família de Processadores Intel® Core™.** Intel, 2020a, [s.p.]. Disponível em: <https://www.intel.com.br/content/www/br/pt/products/processors/core.html>. Acesso em: 24 mar. 2020.
- INTEL CORPORATION. **Intel® Extreme Memory Profile e RAM com overclock.** Intel, 2020b, [s.p.]. Disponível em: <https://www.intel.com.br/content/www/br/pt/gaming/extreme-memory-profile-xmp.html>. Acesso em: 24 mar. 2020.
- INTRODUÇÃO à estrutura e funcionamento de um sistema informático. **TicTic Blog,** 7 nov. 2012. Disponível em: <https://ticticblog.files.wordpress.com/2012/11/imagem32.png>. Acesso em: 21 mar. 2020.
- LOPES, A. V. **Introdução a programação com ADA 95.** Canoas: ULBRA, 1997.
- MONTEIRO, M. A. **Introdução à Organização de Computadores.** 5. edição. Rio de Janeiro: LTC, 2007.

NULL, L.; LOBUR, J. **Princípios Básicos de Arquitetura e Organização de Computadores**. 2. ed. Porto Alegre: Bookman, 2011.

OKUYAMA, F. Y.; MILETTO, E. M.; NICOLAU, M. (Org.). **Desenvolvimento de Software: conceitos básicos**. [recurso eletrônico]. Porto Alegre: Bookman, 2014.

OLIVEIRA, R. A. **Informática**. Rio de Janeiro: Elsevier, 2007. Disponível em: <http://books.google.com.br/books?id=qBaamds7kU8C&pg=PA57&dq=Defini%C3%A7%C3%B5es+de+-Software&lr=#v=onepage&q=Defini%C3%A7%C3%B5es%20de%20Software&f=false>. Acesso em: 24 mar. 2020.

RAINER, K. R.; CEGIESLK, C. G. **Introdução a sistemas de informação**. Tradução: Multinet Produtos. [recurso eletrônico]. 3. ed. Rio de Janeiro: Elsevier, 2012.

SEBESTA, R. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2010.

SOUZA FILHO, G.; ALEXANDRE, E. S. M. **Introdução à computação**. 2. ed. João Pessoa: UFPB, 2014.

STALLINGS, W. **Arquitetura e organização de computadores**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

STIVANI, M. O que é bug? Entenda a origem da palavra e conheça exemplos. **TechTudo**, 21 jan. 2019. Disponível em: <https://www.techtudo.com.br/noticias/2019/01/o-que-e-bug-entenda-a-origem-da-palavra-e-conheca-exemplos.ghtml>. Acesso em: 23 mar. 2020.

TANEMBAUM, A. S. **Organização estruturada de computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013.

TAURION, C. **Software embarcado: oportunidades e potencial de mercado**. Rio de Janeiro: Brasport, 2005.

TEIXEIRA, J. F. **Mentes e máquinas: uma introdução a ciência cognitiva**. Porto Alegre: Artes Médicas, 1998.

UNIDADES de Medida do Computador. **Algo Sobre**, [s.d.]. Disponível em: <https://www.algosobre.com.br/informatica/unidades-de-medida-do-computador.html>. Acesso em: 21 mar. 2020.

VELLOSO, F. C. **Informática: conceitos básicos**. 9. ed. Rio de Janeiro: Elsevier, 2011.

Unidade 2

Rogério Carlos dos Santos

Componentes básicos de um computador

Convite ao estudo

Neste estudo, você irá aprofundar seus conhecimentos sobre os componentes de um computador, além de conhecer a evolução das arquiteturas de computadores ao longo dos anos. Toda a tecnologia dos processadores atuais é uma evolução de uma arquitetura pensada e desenvolvida em meio à Segunda Guerra Mundial e que segue em uma evolução constante até os dias atuais. Nesta unidade, você irá conhecer os componentes básicos de um computador e também os objetivos específicos da unidade.

A competência de fundamento da área da disciplina Arquitetura e Organização de Computadores é conhecer e compreender, além dos princípios da arquitetura e organização, os processadores, a memória principal, a memória secundária e os dispositivos de entrada e saída de um computador, seus conceitos, sua evolução, os diversos tipos desses componentes e como funcionam.

Os objetivos de aprendizagem que serão trabalhados em cada seção são os seguintes:

- Conhecer os processadores, seus conceitos, sua evolução, os diferentes tipos de processadores e seu funcionamento.
- Conhecer a memória principal de um computador, sua evolução, seus tipos e como ele funciona, permitindo o processamento do computador.
- Conhecer o que é a memória secundária do computador, como evoluiu, seus tipos, seus dispositivos e como funcionam.
- Apresentar os dispositivos de entrada e saída do computador, como funcionam internamente no computador e como eles evoluíram até os dias atuais.

Para melhor compreensão e aprofundamento dos conceitos, vamos analisar a situação em que se encontra uma fábrica de componentes de computadores de altíssima tecnologia. Nesse contexto, vamos considerar o setor de pesquisa e desenvolvimento que está sempre em busca de mecanismos e

formas de aprimorar e melhorar esses componentes: de microprocessadores, placas de memória, disco rígido e vários outros até a entrega de equipamentos completos. Você será um dos integrantes do time de pesquisa e desenvolvimento dessa empresa e poderá aprimorar esse desenvolvimento e melhorar esses componentes.

Bom trabalho e bons estudos!

Seção 1

Unidade central de processamento

Diálogo aberto

Caro aluno, nesta unidade você irá aprofundar o seu conhecimento nos conceitos de componentes de um sistema computacional; serão apresentadas as características de cada componente e como eles são interligados; vamos passar pela invenção dos computadores digitais até chegarmos aos dias atuais; além disso, será apresentada a arquitetura de Von Neumann, bem como seus componentes. Foi a partir dessa arquitetura que os processadores de hoje foram construídos, por meio de suas unidades, seus principais componentes e estudos sobre a unidade central de processamento, a unidade de memória e os dispositivos de entrada e saída.

Neste momento, você verificará, de forma mais detalhada, o funcionamento da CPU – unidade central de processamento –, que é o principal componente de um computador. Ela é responsável por controlar as unidades de memória e os dispositivos de entrada e saída do computador, bem como carregar e executar os programas (SOUZA FILHO, 2014).

Uma das tendências identificadas pela empresa de fabricação de microprocessadores é a integração de operações básicas de controle, serviços e oferta de segurança para se ampliar a qualidade de vida da população que se pretende inserir com as “cidades inteligentes”. Por exemplo: disponibilizar ao cidadão a identificação de locais que têm vagas de estacionamento disponíveis, pontos da cidade em obras e/ou congestionados, disponibilidade de agenda para serviços de saúde e uma infinidade de situações que possam exigir integração, comunicação, etc. Para isso, é necessário que os computadores e dispositivos que executarão essas tarefas tenham processadores com grande desempenho e permitam viabilizar essas operações. O desafio, então, consiste em apresentar as características e identificar processadores que permitam tais operações, bem como identificar um modelo que já esteja disponível no mercado, minimizando os investimentos e fortalecendo parcerias comerciais.

Faça a sugestão de um processador existente no mercado e que possa atender aos requisitos dos sistemas propostos na situação real, como, por exemplo, identificação de locais com vagas de estacionamento disponíveis, pontos da cidade em obras e/ou congestionados e disponibilidade de agenda para serviços de saúde.

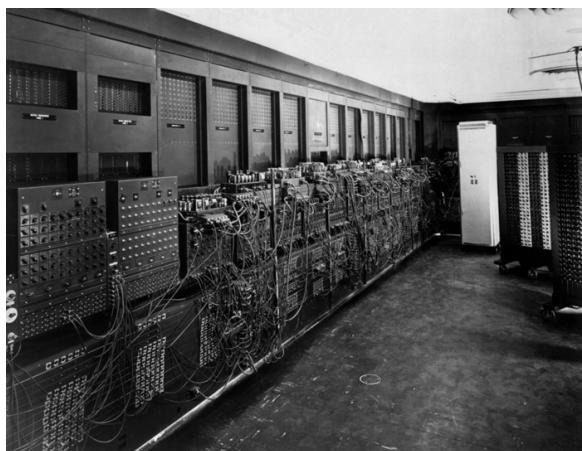
Vamos conhecer mais sobre os processadores e encontrar uma boa opção para todas essas questões? Esse é o seu desafio. Vamos ao trabalho!

Não pode faltar

Atualmente, os computadores estão presentes em várias atividades do cotidiano, seja em tarefas domésticas ou em seu ambiente de trabalho, como quando você faz compras online ou utiliza algum tipo de ferramenta de edição de textos, operações bancárias, impressão de documentos, telecomunicações, enfim, a lista de atividades que você pode realizar por meio de um computador é enorme. Nesse sentido, é muito importante conhecer o funcionamento dos computadores e seus componentes, bem como sua evolução desde os primórdios até os tempos atuais.

Os computadores atuais seguem uma arquitetura implementada logo após o final da Segunda Guerra, na década de 1940, por John von Neumann, chamada de “Arquitetura de Von Neumann”. Ele introduziu o conceito de programa armazenado em que os programas e os dados podem ser armazenados em um mesmo espaço de memória. Essa abordagem foi muito importante para o projeto dos computadores atuais, que têm evoluído ao longo dos anos. Antes da criação da arquitetura de Von Neumann, os programas não eram armazenados, toda a programação era realizada no *hardware*, via conexão e reconexão de cabos; a cada diferente cálculo, os cabos eram deslocados para diferentes posições. O computador chamado ENIAC (*Electronic Numerical Integrator and Computer*) está ilustrado na Figura 2.1.

Figura 2.1 | Primeiro computador eletrônico – ENIAC

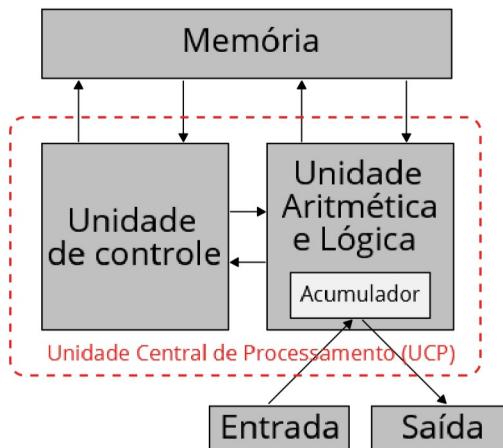


Fonte: Shutterstock.

Posteriormente, surgiu a arquitera de Von Neumann tomada como princípio de funcionamento dos computadores atuais. A arquitetura de Von Neumann é composta por unidade lógica aritmética, unidade de controle, memória e dispositivos de entrada e saída (RAINER, 2012).

O processador, também chamado de unidade central de processamento (UCP) ou CPU, do inglês *central processing unit*, é o componente fundamental de um sistema de computação, uma vez que ele é o responsável por realizar importantes operações, como computar, calcular e processar os dados de modo a produzir algum resultado ou ação. A Figura 2.2 ilustra a máquina de Von Neumann simplificada, em que mostra as conexões entre os componentes.

Figura 2.2 | Arquitetura de Von Neumann



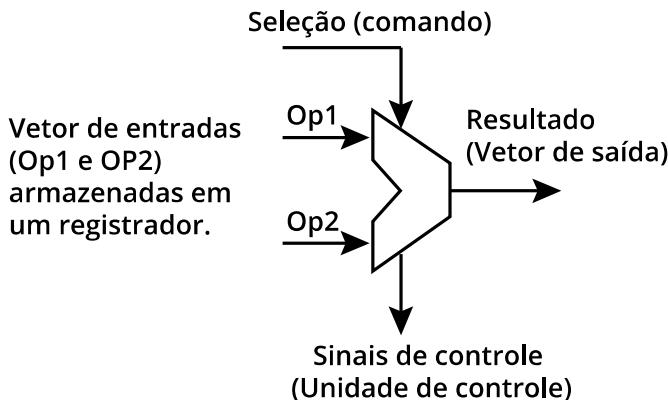
Fonte: adaptada de Wikimedia.

A UCP tem duas funções essenciais: controle e processamento. Na função de processamento dos dados é realizada a manipulação de dados em uma determinada sequência cujas instruções (uma instrução é uma tarefa a ser processada pelo UCP) estão na memória. Outra função da UCP é o controle. Após a interpretação de uma instrução, em geral, temos alguma ação / algum resultado, e daí surge a função controle, que emite sinais para os demais componentes interligados. A unidade de controle é a responsável por essa função; ela define a sequência de instruções a serem executadas e dispara sinais de controle para os outros componentes, como memória e unidade lógica aritmética.

A UCP contém, além da unidade de controle mencionada, a unidade lógica e aritmética, que é responsável por executar operações aritméticas (como subtração (SUB), adição (ADD) e comparações), operações lógicas e transferência de dados, como operações de carregamento (LOAD) e armazenamento (STORE) dos dados. Os dados usados para esses cálculos são armazenados na memória do computador, também chamados de registradores (OKUYAMA, 2014). A Figura 2.3 ilustra a representação de uma unidade aritmética. Na figura, é ilustrada uma ULA que possui, basicamente, duas entradas e duas saídas. Como entradas, temos a entrada de seleção e os dois operandos (Op1 e Op2) ou um vetor de valores de entradas; como saída, temos o resultado (onde se dará o resultado da operação) e o controle, onde são enviados os sinais de controle para realização de alguma ação. A entrada de seleção é a responsável pela escolha da operação a ser realizada pela ULA, como, por exemplo, se deve ser realizada uma soma, subtração ou uma operação lógica, a depender do valor da entrada de seleção.

É importante destacarmos que os valores dos operandos a serem trabalhados pela ULA ficam armazenados em um registrador devido ao rápido acesso que esse tipo de memória permite.

Figura 2.3 | Representação de uma unidade lógica aritmética



Fonte: elaborada pelo autor.

Para que haja comunicação entre os componentes do computador, eles devem ser interligados via barramentos, que possibilitam o fluxo dos dados entre os componentes para que sejam processados e/ou armazenados, por exemplo.

Outro componente são os dispositivos de entradas, que são responsáveis por enviar alguma informação do mundo exterior a ser processada pelo computador (são exemplos de dispositivos de entrada, teclado, *mouse*, *scanner*, entre outros). Por fim, os dispositivos de saída retornam alguma informação resposta (monitor, impressora).

Exemplificando

Imagine que você solicita, via dispositivo de entrada, a impressão de algum documento para que seja impresso em um dispositivo de saída (impressora). A instrução de impressão deve ser processada e sinais de controle devem ser emitidos para que a solicitação de impressão seja atendida no momento correto. Esses sinais de controle são gerados via unidade de controle.

A arquitetura de Von Neumann foi uma grande contribuição para o projeto de processadores atuais, entretanto, possuía algumas limitações: a comunicação entre a unidade central de processamento e a memória principal eram realizadas por meio de um barramento único, logo, havia um tráfego intenso de dados e instruções no mesmo barramento, havendo a necessidade da UCP esperar o processamento de instruções em uma determinada sequência. Assim, posteriormente, as arquiteturas de computadores evoluíram, incluindo outros componentes, como, por exemplo, barramentos adicionais para comunicação entre componentes; além disso, técnicas de paralelismo (processamento paralelo – *pipelining*), foram incorporadas às arquiteturas que antes executavam instruções sequencialmente, melhorando a eficiência dos processadores em termos de tempo de execução.

Agora, vamos detalhar uma arquitetura de computadores em uma visão não simplificada, incluindo algumas melhorias das arquiteturas.

Um processador ou UCP, como já mencionado, manipula dados executando ações e com o objetivo de obter resultados. São ações comuns à execução de operações aritméticas simples, tais como: somar, subtrair, multiplicar e dividir; operações lógicas e de movimentação de dados entre a UCP e a memória. Para que haja essas operações, os dados devem ser armazenados em memórias. A UCP, internamente, possui memórias de alta velocidade para o armazenamento temporário, e essas memórias permitem o rápido acesso às informações e são denominadas registradores. Existem vários tipos de registradores, cada um deles possui uma função predefinida. Alguns dos registradores que podemos citar são os seguintes: acumulador (AC), registrador de instrução (RI), contador de instrução (CI, também conhecido por PC, do inglês, *program counter*), registrador de dados de memória (RDM) e

registrador de endereços de memória (REM) (FÁVERO, 2011). O RDM e o REM são utilizados pela UCP e pela memória para a comunicação e transferência de dados. Cada um desses registradores armazena informações específicas, como a lista a seguir:

- Acumulador (ACC) – utilizado para a unidade lógica aritmética, ele armazena os operandos da ULA.
- Registrador de instrução (RI) – armazenamento da instrução que está sendo executada naquele momento pela UCP.
- Contador de instruções (CI) – aponta o endereço da próxima instrução a ser executada.
- Registrador de dados de memória (RDM), também chamado de MBR (*memory buffer register*) – armazena os dados de memória.
- Registrador de endereços de memória (REM), também chamado de MAR (do inglês, *mempory address register*) – guarda os endereços de memória a serem acessados.

Além desses componentes, temos o decodificador de instruções. O decodificador é responsável pela identificação de operações a serem realizadas, assim, cada operação deve conter um código que foi previamente decodificado; em outras palavras: imagine que você tem muitas instruções a serem executadas, quais seriam as operações executadas para essas instruções? O decodificador de instruções permite que seja acionada uma entrada específica da unidade de controle para que esta dispare sinais de controle a fim de ser realizada alguma ação.

Assimile

Uma unidade central de processamento é composta por uma unidade de controle, uma unidade lógica aritmética e registradores. Os componentes são interligados via barramento. A unidade de controle é o componente responsável por gerar e emitir sinais de controle para que possibilite a movimentação dos dados e de instruções do processador para outros componentes, como, por exemplo, para dispositivos de entrada e saída e memória.

Os componentes do processador são interligados via barramentos. Um barramento é o caminho por onde trafegam todas as informações de um computador, e existem três tipos principais:

- Barramento de dados.
- Barramento de endereços.
- Barramento de controle.

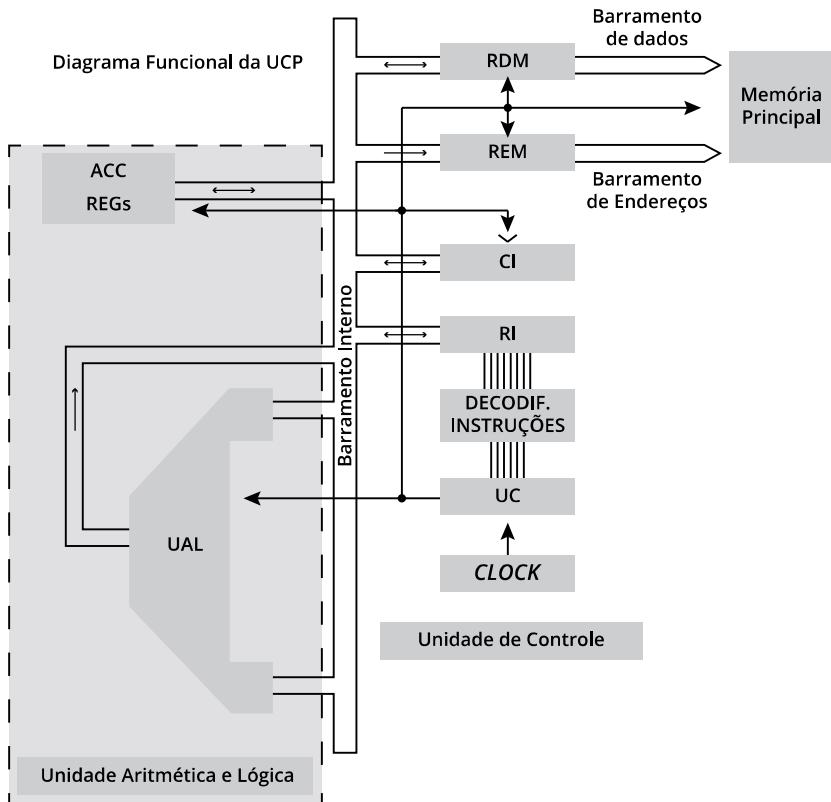
Barramento de dados: esse barramento interliga a CPU à memória (e vice-versa) para a transferência das informações que serão processadas. Ele determina diretamente o desempenho do sistema, pois, quanto maior o número de vias de comunicação, maior o número de *bits* transferidos e, consequentemente, maior a rapidez com que esses dados serão processados. Os primeiros PCs possuíam barramento de 8 vias. Atualmente, dependendo do processador, esse número de vias pode ser de 32, 64 e até de 128 vias (FÁVERO, 2011).

Barramento de endereços: interliga a CPU à memória, fazendo seu endereçamento, e tem o número de vias correspondente à tecnologia de *bits* do processador, ou seja, nos computadores mais modernos, 32 *bits* ou 64 *bits*, e conforme já visto por você, permitindo endereçar até 4 GB (*Gigabytes*) de memória em processadores 32 *bits*, e cerca de 16 PB (*Petabytes*) no caso de processadores de 64 *bits* (SOUZA FILHO, 2014).

Barramento de controle: interliga a CPU à unidade de controle, aos componentes e aos dispositivos de um computador, aos componentes de entrada e saída, às memórias auxiliares e de armazenamento, entre outros. Por trabalhar com componentes externos ao processador, pode ser chamado também de barramento externo (MONTEIRO, 2007).

Na Figura 2.4, pode-se observar o diagrama funcional básico da unidade central de processamento em que temos, internamente, os registradores. Além disso, temos barramentos de dados, endereços e barramento interno. Observe que o barramento de dados interliga o registrador de dados da memória à memória principal, e o registrador de endereços está interligando o registrador de endereços e a memória principal (também chamada de memória RAM) por onde ocorre o fluxo de informações entre os componentes da CPU. Os registradores são utilizados para armazenar temporariamente informações que serão posteriormente utilizadas pela UCP.

Figura 2.4 | CPU – diagrama funcional



Fonte: Fávero (2011, p. 60).

Outro ponto importante a ser destacado é a velocidade com que a CPU trabalha, medida por ciclos de *clock*. Ciclo de *clock* é o tempo gasto pelo processador para executar uma instrução, transferir um dado entre ele e a memória e definir sua velocidade. Esse tempo é medido em Hertz (Hz), ou seja, a quantidade de ciclos é processada por segundo, sendo que 1 Hz equivale a 1 ciclo por segundo. Os processadores atuais trabalham com velocidades na casa dos Gigahertz (GHz) (PATTERSON, 2014). Algumas medidas são apresentadas a seguir:

1000 Hz equivale a 1kHz (Quilohertz); 1000 kHz = 1.000.000 ou 1 MHz (mega-hertz) e 1000 MHz equivale a 1.000.000.000 ou 1 GHz.

Refita

Um processador que executa cem operações por segundo possui um *clock* de 100 Hz. Logo, imagine quantas operações um processador da atualidade com frequência de 5 GHz realiza por segundo? Seria apenas 5 bilhões de cálculos em um único segundo. E quando temos processadores com múltiplos núcleos, quantos cálculos ele pode realizar por segundo?

Evolução de processadores

Os microcomputadores surgiram na década de 1970 e trouxeram, em sua tecnologia, novos componentes. Em um primeiro momento, as CPUs foram desenvolvidas em circuitos integrados, que eram um único *chip* de silício contendo milhares de transistores que traziam as instruções observadas pela arquitetura de Von Neumann (SOUZA FILHO, 2014); após isso, com a chegada dos microprocessadores, a prioridade passou a ser a ampliação da sua capacidade de processamento. Os primeiros microcomputadores tinham processadores com tecnologia de 8 *bits* e barramento com 8 *bits*, que era o caso do então processador 8080. Após esses primeiros modelos, foram lançados processadores com 16 *bits* de processamento interno e barramento, e na sequência, os processadores de 32 *bits*, de 64 *bits* e 128 bits, devido a uma maior demanda por processadores com maior desempenho, com maior capacidade de processar dados, principalmente informações em tempo real. Em técnicas da computação gráfica, como a renderização, os processadores passaram a contar com a possibilidade de ter mais de um núcleo de processamento, como é o exemplo dos processadores com múltiplos núcleos de processamento, também chamados de *multicore*, dos quais fazem parte os modernos processadores da intel i3, i5, i7, i9 e Xeon. O processador Xeon pode contar com até 56 núcleos de processamento. A tecnologia desses processadores avançou tanto que, hoje, processadores escalonáveis com técnicas de inteligência artificial, tais como *deep learning*, são embutidos nos processadores, como o processador *Xeon Platinum* (INTEL, 2020).

Arquiteturas CISC e RISC: segundo Monteiro (2007), o que define um projeto de um processador é a quantidade de instruções de máquina que se deseja que ele, processador, execute; quanto menor esse conjunto de instruções, mais rápido se torna um processador. Partindo desse princípio, os processadores têm dois tipos de arquiteturas empregados pelos seus fabricantes: a arquitetura CISC (*Complex Instruction Set Computers*) – sistema com um conjunto de instruções complexo, utilizado pelos processadores de computadores pessoais; e a arquitetura RISC (*Reduced Instruction Set Computer*) – sistema com um conjunto de instruções reduzido, que é empregado nos processadores ARM utilizados pelos *smartphones* e tablets atuais (MONTEIRO, 2007).

Assimile

Os processadores podem ser de dois tipos, de acordo com sua tecnologia: processadores CISC, que processam centenas de conjuntos complexos de instruções simples, o que significa que cada instrução isoladamente é considerada simples, curta e pouco potente, porém, várias dessas instruções agrupadas formam um conjunto complexo que é executado pelo processador; e os processadores RISC, que têm um conjunto reduzido de instruções, e diferentemente da tecnologia CISC, essas instruções são consideradas complexas, pois cada uma delas executa várias tarefas conjuntas.

Processadores CISC (*Complex Instruction Set Computers*)

Os processadores com tecnologia CISC são capazes de processar centenas de conjuntos complexos de instruções simples. Isso significa que cada instrução isoladamente é considerada simples, curta e pouco potente, porém, várias dessas instruções agrupadas formam um conjunto complexo que é executado pelo processador. Inicialmente, existia uma grande tendência a essa tecnologia de processadores, porém, havia algumas desvantagens, como o desempenho reduzido, que se dava justamente pelo excesso de instruções executadas pelo processador e pela velocidade de processamento elevada para que o desempenho fosse melhorado. A ideia dos fabricantes era produzir processadores cada vez mais potentes, baseados na complexidade desses conjuntos de instruções (BROOKSHEAR, 2013).

Processadores RISC (*Reduced Instruction Set Computer*)

Os processadores com tecnologia RISC têm um conjunto reduzido de instruções, e diferentemente da tecnologia CISC, essas instruções são consideradas complexas, pois cada uma delas executa várias tarefas conjuntas, e isso permite uma vantagem dos processadores RISC em relação aos processadores CISC, que, por terem um número menor de instruções, têm menos circuitos internos e, assim, podem trabalhar com frequências muito maiores sem ter problemas de superaquecimento dos processadores (BROOKSHEAR, 2013).

Chegamos ao final da seção, exerçite os conhecimentos para aplicá-los no seu cotidiano. Agora, você já tem conhecimento para solucionar mais um desafio! Vamos lá!

Sem medo de errar

Uma das tendências identificadas pela empresa de fabricação de microprocessadores é a integração de operações básicas de controle, a disponibilidade de serviços e a oferta de segurança para se ampliar a qualidade de vida da população que se pretende inserir com as “cidades inteligentes”. A empresa quer investir em sistemas de navegação GPS para a linha automotiva que possam ser conectados à internet e definir melhores rotas considerando as informações do trânsito em tempo real.

O desafio consiste em apresentar as características de um processador que permita a realização dessas operações e, ainda, identifique no mercado um modelo que já esteja disponível, minimizando os investimentos e fortalecendo parcerias comerciais.

É importante destacarmos os processadores da atualidade, como os da série i7, i9 e Xeon. Esses são processadores com alto desempenho para diversas aplicações, não apenas para informações de trânsito em tempo real, mas para qualquer tipo de aplicação que trabalhe com grandes quantidades de dados. Alguns dos processadores que podem ser citados para a resolução da SP são as séries, i7, i9 e Xeon; além desses processadores, podemos mencionar os processadores de alto desempenho da AMD.

Quadro 2.1 | Processadores

Modelo	Frequência	Núcleos	Cache	Frequência Turbo max
i7-10510U	4.8 GHz	4	8 MB	4,90 GHz
i7-9700K	4.90 GHz	8	12 MB	4.90 GHz
AMD Ryzen 9 3900X	3.8 GHz	12	64 MB (máx L3)	4.6 GHz
i9-9900KS	4 GHz *(5GHz)	8	16 MB	5 GHz
i9-9900	3.10 GHz *(5 GHz)	8	18 MB	5 GHz
i9 -10980XE	4.80 GHz	10	24.75 MB	4.60 GHz
i9 -7920 X	2.90 GHz	12	16.5 MB	4.30 GHz
i9 -79980XE	2.60 GHz	18	24.75 MB	4.20 GHz
Xeon Platinun 8168	2.70 GHz (*3.70 GHz)	28	33 MB	3.70 GHz
XEON Platinum 9282	2.60 GHz *(3.80 GHz)	56	77 MB	3.80 GHz
AMD Ryzen™ Threadripper 3990 X	2.9 GHz	64	256 MB (máx L3)	4.3 GHz

Fonte: adaptado de Intel ([s.d.]).

É importante destacarmos que os processadores evoluíram tanto que, hoje, temos técnicas de inteligência artificial embutidas.

Faça valer a pena

1. A Arquitetura de Von Neumann prevê uma unidade de processamento central (CPU), unidade de memória, unidade de controle e unidades de entrada e saída.

Em relação a esses componentes, a unidade de controle é:

- a. Responsável por realizar operações de lógica aritmética.
- b. Responsável por selecionar as operações que serão realizadas pela ULA.
- c. Responsável pelo fluxo de dados entre os componentes de uma unidade central de processamento.
- d. Responsável por enviar sinais que controlam as ações a serem executadas pela UCP.
- e. Responsável por enviar os dados à memória e armazená-los nos registradores.

2. Os registradores são memórias de alta velocidade que permitem o acesso rápido às operações a serem realizadas pela UCP.

Sobre os diferentes registradores existentes, assinale a alternativa correta.

- a. O registrador de instrução é responsável por armazenar as operações realizadas pela ULA.
- b. O contador de instrução é responsável por apontar o endereço da instrução que está em execução.
- c. O registrador de endereços de memória armazena instruções que serão executadas na memória.
- d. O registrador de dados da memória guarda os endereços das instruções executadas no momento atual.
- e. O registrador de instrução armazena a instrução que está em execução.

3. Um barramento de dados determina diretamente o desempenho do sistema, pois quanto maior o número de vias de comunicação, maior o número de *bits* transferidos e, consequentemente, maior a rapidez com que esses dados serão processados.

Sobre os barramentos existentes, assinale a alternativa correta.

- a. O barramento de dados interliga a CPU e a memória para a transferência da informação a ser executada.
- b. O barramento de controle conecta a CPU e a ULA para a transferência dos operandos a fim de que a ULA realize as operações.
- c. O barramento de endereços conecta a memória RAM à ULA.
- d. O barramento de controle interliga a memória principal à ULA.
- e. O barramento de endereço envia as instruções a serem processadas para a CPU.

Seção 2

Memória principal e memória cache

Diálogo aberto

Neste momento, você terá a oportunidade de conhecer melhor alguns importantes aspectos da arquitetura de um computador pensada por Von Neumann, em particular as memórias, seu funcionamento e seus tipos, e mais detalhadamente a memória principal, também chamada de memória RAM de um computador; nela, a função básica do processador é receber os dados e as instruções dos programas e processá-los. Desde que foi pensada e introduzida a arquitetura de Von Neumann, para que um processador possa executar esse processamento, é necessária a utilização de memória, e como você já teve a oportunidade de ver, uma das grandes conquistas dessa arquitetura foi poder ter, na mesma memória, dados e processamento de programas.

Retomando as questões vistas pela empresa de fabricação de microprocessadores, por meio das câmeras de segurança das cidades inteligentes, pretende-se inserir a maior quantidade de serviços possíveis por meio da comparação de dados e de imagens capturadas por elas. O cruzamento de dados e informações, além de microprocessadores de alta capacidade, requer alto índice de desempenho em questões de armazenamento. Para que a empresa de fabricação possa incluir essa inovação em suas soluções, é necessário que o setor de pesquisa e desenvolvimento consiga inserir uma memória que conte com essa necessidade de armazenamento e ofereça a oportunidade de identificação, comparação e localização, utilizando os dados armazenados em seus servidores. Para tal, a documentação das especificações técnicas desse produto e o estreitamento das relações comerciais com o fornecedor dessa tecnologia são essenciais e devem ser contemplados nessa etapa. Uma das tendências identificadas pela empresa de fabricação de microprocessadores é a integração de operações básicas de controle, a disponibilidade de serviços e a oferta de segurança para ampliar a qualidade de vida da população que se pretende inserir com as “cidades inteligentes”. A empresa quer investir em uma linha de computadores que possam oferecer grande capacidade de processamento para atender a essa demanda, porém, precisa que os computadores tenham máxima eficiência de consumo de energia pelas limitações atuais e mundiais de geração de energia. O desafio consiste em realizar uma pesquisa e apresentar as características de um computador servidor que permita o uso de memórias com alto desempenho e baixo consumo de energia. Monte uma tabela com as descrições de alguns fabricantes de memória e relate um

perfil de usuário com as memórias, por exemplo: um usuário frequente, um jogador, um usuário profissional.

Encontre e relate a descrição técnica apresentada pelo fabricante e apresente essa descrição em forma de relatório ao seu professor.

Bom trabalho!

Não pode faltar

A memória é indispensável e tão importante quanto a unidade central de processamento (UCP); é onde os dados e as instruções processadas a serem executadas por uma unidade central de processamento (UCP) devem estar armazenados; por isso, quando compramos ou estamos em busca de um computador, pesquisamos sua capacidade de memória RAM, capacidade de armazenamento do HD (do inglês, *hard disk*), além da quantidade de níveis (um, dois ou três níveis) de cache do processador.

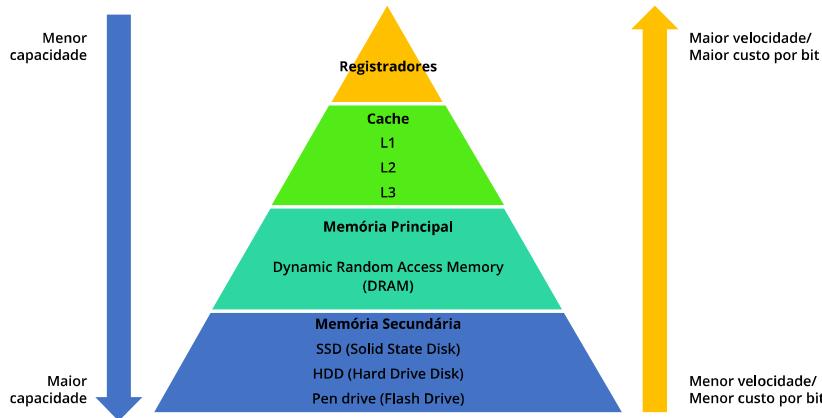
Existem diversos tipos de memória com diferentes propósitos, objetivos e velocidades; algumas apenas para armazenamento de dados e outras possuem a função de aumentar o desempenho dos processadores. Imagine se os computadores tivessem apenas um tipo de memória, ela deveria ter a mesma velocidade que a UCP esperando por dados que estivessem sendo transferidos para processamento. Na prática, o que acontece, por exemplo, é que, em um computador que processa um dado em 5 ns (nano segundos), a memória transfere o dado em 60 ns (PATTERSON, 2005); nesse sentido, dependendo da memória e de sua velocidade, o processador fica mais ou menos tempo ocioso.

A memória de um computador não é uma única peça isolada, existem vários tipos de memória. Por exemplo, a memória que armazena as instruções e os dados que devem ser enviados para o processador é um tipo diferente da memória de armazenamento, em que os dados são guardados em um computador. Com o aumento do desempenho das arquiteturas de processadores atuais, houve a necessidade da evolução da hierarquia de memória, de se pensar uma forma de projetar memórias que permitissem que o processador trabalhasse com o máximo de eficiência e o menor desperdício de recursos de processamento. Nesse sentido, existem vários tipos de memórias para suprir as necessidades do usuário, cada qual com sua função específica (FÁVERO, 2011).

Algumas memórias são classificadas em registradores, cache, memória principal e memória secundária. Nesta seção, vamos focar as memórias principal e cache (MONTEIRO, 2007).

As memórias de um computador podem variar também em sua tecnologia, sua capacidade de armazenamento, sua velocidade e seu custo; elas são interligadas de forma estruturada, compondo um subsistema de memória. Essa hierarquia organiza as memórias quanto à sua capacidade, à sua velocidade e ao seu custo. Quanto mais próximo da base do triângulo (abaixo), maior a capacidade de armazenamento, porém, menor a velocidade; e quanto mais para o topo do triângulo, maior a velocidade e o custo, todavia, menor a capacidade de armazenamento (FÁVERO, 2011).

Figura 2.5 | Hierarquia de memórias

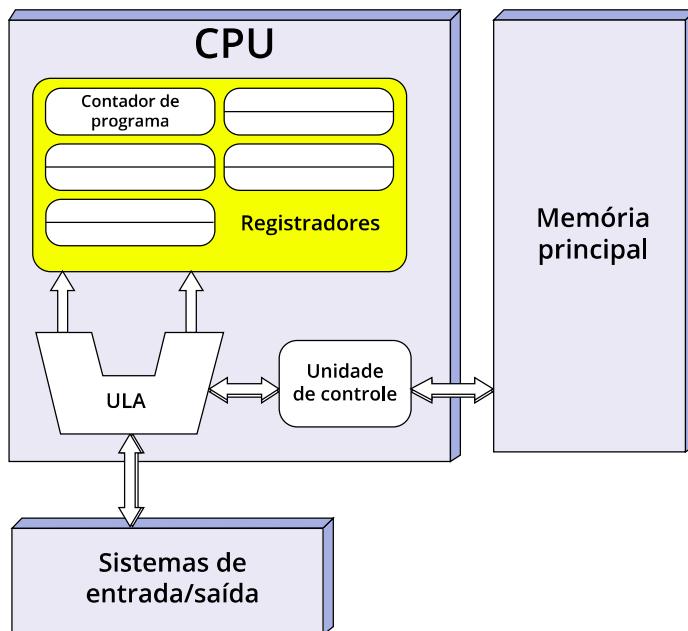


Fonte: elaborada pelo autor.

Para que você possa entender melhor a função da memória principal de um computador, é necessário também conhecer dois tipos específicos de memórias que auxiliam o gerenciamento dos dados, que são os registradores e a memória cache (FÁVERO, 2011). Os registradores são memórias de alta velocidade e baixa capacidade utilizadas nas operações da unidade lógica aritmética da UCP. Os registradores também armazenam endereços de instruções a serem executadas e em execução. Nesse sentido, os registradores têm a função de armazenar dados, instruções e endereços de dados que serão processados ou estão em execução pela UCP. Essas memórias têm o maior custo por bit.

Importa destacar que os registradores estão localizados na unidade central de processamento (processador), como podemos visualizar na Figura 2.6.

Figura 2.6 | Registradores em destaque dentro da estrutura de um processador



Fonte: adaptada de Wikimedia.

Outro conceito importante sobre memórias é que elas podem ser voláteis ou não voláteis. As memórias voláteis requerem energia para funcionar e armazenar dados, ou seja, só funcionam quando o computador está ligado, e os dados armazenados nelas são apagados quando o computador é desligado, em geral, são as memórias utilizadas para o processamento de instruções e dados (do processador). As memórias não voláteis gravam os dados de forma permanente em seus dispositivos, não sendo apagados quando se desliga o computador, podendo ser lidos e recuperados quando for necessário (PATTERSON, 2005).

Temos, assim, uma classificação hierárquica das memórias de acordo com suas características, que são descritas mais claramente de acordo com o Quadro 2.2, como segue:

Quadro 2.2 | Quadro das características básicas dos tipos de memória

Características básicas dos tipos de memória					
MEMÓRIA	LOCALIZAÇÃO/		VELOCIDADE	CAPACIDADE DE ARMAZENAMENTO	CUSTO
	É VOLÁTIL				
Registrador	Processador	Sim	Muito alta (opera na velocidade do processador)	Muito baixa (Bytes)	Muito alto
Cache	Processador	Sim	Alta (opera na velocidade do processador)	Baixa (KB)	Alto
Principal	Placa-mãe	RAM – sim RAM – não	Depende do tipo de memória instalada	Média (GB)	Médio (tem caído muito)
Secundária	HD, CDs, etc.	Não	Baixa (lenta)	Alta (GB)	Baixo (tem caído muito)

Fonte: adaptada de Murdocca (2001); Fávero (2011).

O registrador é um tipo de memória volátil. Por estar dentro do processador, proporciona uma velocidade de transferência bastante alta, mas sua capacidade de armazenamento é baixa, uma vez que divide espaço com as demais unidades do processador.

Como o processador é uma das peças mais caras de um computador, o custo desse tipo de memória é, por consequência, bastante caro (TANENBAUM, 2007).

Memória cache

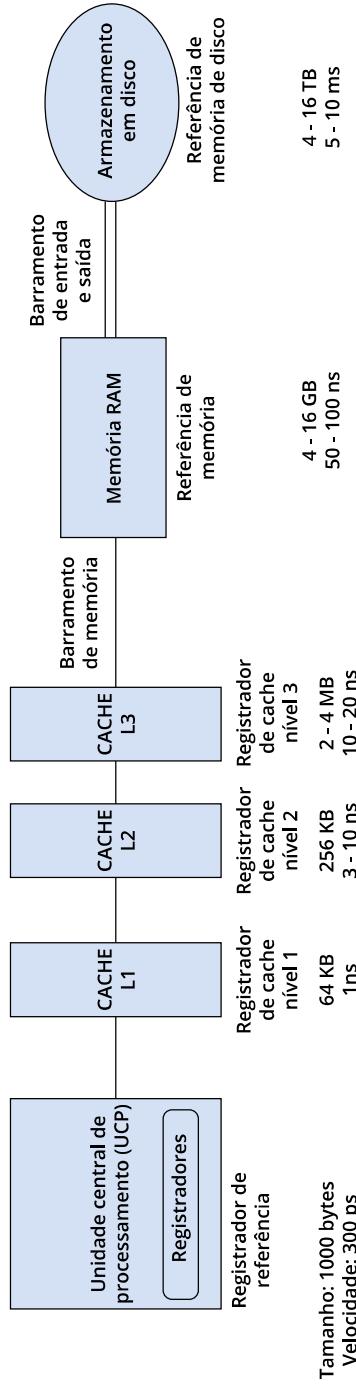
Quando entramos com os dados em um computador ou quando ele executa instruções vindas dos programas, o processador busca esses dados em uma memória externa, ou seja, uma memória que não está dentro do próprio processador, chamada de memória principal. Como a velocidade dos processadores é, em geral, muito maior do que a velocidade das memórias, gera-se uma fila de espera entre os dados encontrados na memória e o processador na hora da execução do processamento (FÁVERO, 2011). Temos a memória cache para poder solucionar essa limitação entre a velocidade de processamento e a velocidade da memória principal. A memória cache é uma memória com menor capacidade que a RAM, porém, possui velocidades mais altas.

Esse tipo de memória mantém uma cópia dos dados e das instruções frequentemente utilizadas pelo processador, evitando que ele busque os dados e as instruções na memória RAM, que é uma memória mais lenta em comparação à cache. Logo, a memória cache é uma memória intermediária entre a RAM e o processador; ela tem a função de criar condições que aumentam a velocidade de comunicação entre a memória principal e o processador, aumentando, assim, a velocidade final do processamento. Esse tipo de memória também é volátil e se apaga quando o computador é desligado (PATTERSON, 2014).

Devido ao alto custo na hierarquia de memória, ela é organizada em níveis: *level 1* (L1), *level 2* (L2) e *level 3* (L3), sendo a L1 com maior velocidade e capacidade mais baixa e a L3 com maior capacidade e velocidade inferior.

O intuito de ter vários níveis de cache é fornecer um sistema de memória com custo mais baixo e uma velocidade quase tão rápida quanto a de nível mais veloz (HENNESSY, 2014). A Figura 2.7 a seguir ilustra os níveis de hierarquia de memória em um servidor e em um computador pessoal.

Figura 2.7 | Hierarquia de memória com velocidades e capacidade



Fonte: adaptada de Hennessey (2014).

Observe que quanto mais distante dos processadores, mais lenta é a memória e com maior capacidade. Um exemplo é a memória de disco (HD), que tem a capacidade de terabytes e a velocidade na ordem de 5 a 10 milissegundos, enquanto que as memórias cache (L1, L2 e L3) têm velocidades na ordem de nanosegundos, e o registrador, memória mais rápida da hierarquia, na ordem de pico segundos e com menor capacidade (1000 bytes). É importante ressaltar que a memória cache de menor nível (L1) possui capacidade mais baixa e velocidade mais alta que a cache L3.

O princípio básico de funcionamento do processador e a cache é o seguinte: quando um processador requer alguma informação a ser processada, primeiro é realizado o acesso à memória cache, e se essa informação for encontrada, ocorrerá o cache hit (chamado de acerto de cache) e ela será transferida para a UCP. Caso contrário, se a informação não for encontrada na cache (chamado de cache miss), será realizada a busca na memória principal (RAM), que envia as informações ao processador, e, por fim, essa informação será copiada para a memória cache.

Assimile

Os registradores são as memórias com maior velocidade e menor capacidade de armazenamento. As memórias cache são memórias intermediárias entre o processador e a memória RAM e são utilizadas para armazenamento de dados mais acessados, evitando-se que a memória RAM, que possui uma velocidade mais baixa que a cache, seja acessada muitas vezes.

A memória cache é uma memória dita “estática” pois, uma vez colocado o dado, ele permanece enquanto a memória for alimentada. Esse tipo de memória é baseado em circuitos do tipo “*flip-flop*”. Essas memórias são muito rápidas porque os circuitos *flip-flop* são feitos com transistores, e a leitura é feita simplesmente medindo-se a tensão de saída, em que 0 (zero) volt gera um bit “0” e 5 volts gera um bit “1”. Apesar de mais rápida, são necessários muitos transistores e resistores para se fazer um *flip-flop* (1 bit), o que torna o custo dessa memória muito alto (PATTERSON, 2014).

A memória cache é uma memória que, atualmente, encontra-se internamente nos processadores, entre a CPU e a memória principal, que espelha parte dessa memória e torna o processamento mais rápido (FÁVERO, 2011). Atualmente, o tamanho dessa memória cache chega a 256 MB compartilhado por todos os núcleos do processador, como no caso do processador de 64 núcleos da série AMD Ryzen.

Memória principal

Como o próprio nome já diz, essa memória mais o processador são os elementos principais para o funcionamento de um computador. Não há computador se não houver um processador, e também não há um se não existir memória (MONTEIRO, 2007).

A memória principal é chamada de memória RAM – *random access memory*; em português, memória de acesso aleatório –, que faz o armazenamento dos dados inseridos no computador, dos dados dos programas e dos próprios programas. Ela é chamada de aleatória porque, para preservar os circuitos de deterioração, a cada acesso de escrita, um *bit* aleatório é escolhido, evitando-se que sempre os mesmos *bits* sejam usados, o que causaria fadiga no circuito. Além disso, a memória RAM permite ao processador ter acesso às memórias secundárias, disponibilizando os dados gravados nelas e o seu processamento (PATTERSON, 2014).

A memória RAM é do tipo volátil, ou seja, é apagada quando o computador é desligado. É por esse motivo que muitos usuários perdem os trabalhos que estão sendo feitos no computador quando a energia é interrompida de repente, pois, enquanto esses trabalhos não são gravados em um disco rígido, por exemplo, eles não são arquivos, são apenas dados que estão, naquele momento, sendo processados pelo computador (FÁVERO, 2011).

Figura 2.8 | Pente de memória RAM DDR4



Fonte: Wikimedia.

Segundo Fávero, a memória RAM é conhecida também por DRAM (*Dynamic RAM*) ou, traduzindo, RAM dinâmica. Ela é considerada dinâmica porque tem a necessidade de refrescamento de memória, um recurso que realimenta de energia as memórias e mantém os dados armazenados enquanto o computador estiver ligado, pois, sem esse recurso, a memória ficaria sem energia e seus dados seriam perdidos.

Isso ocorre porque as memórias dinâmicas, ao contrário das memórias estáticas, são feitas com capacitores. A leitura de um capacitor que esteja

descarregado gera o bit “0” (zero). A leitura de um capacitor carregado gera o *bit* “1”. Ocorre que o capacitor deve ser recarregado de tempos em tempos para que sua carga não se deteriore e, assim, o *bit* seja perdido. Esse processo de leitura por descarga de capacitores é lento, o que torna esse tipo de memória mais lento. O ponto a favor dele é que, por ser baseado em capacitores, seu custo torna-se menor (PATTERSON, 2014).

Por serem constantemente refreshadas, as memórias DRAM consomem muitos ciclos de processamento e muito mais energia que outros tipos de memória, o que as tornam mais lentas; em contrapartida, tem seu custo menor e uma maior capacidade de armazenamento de dados (MONTEIRO, 2007).

Conforme você pôde observar na Figura 1.3, a memória RAM tem o formato de pente; trata-se e um módulo composto por uma pequena placa com circuitos integrados que determinam sua capacidade e sua taxa de transferência.

Segundo Monteiro (2007), existem diferentes modelos de módulos de memória disponíveis no mercado, sendo, atualmente, mais comum o uso dos modelos DIMM – *dual inline memory module* ou, traduzindo, módulo de memória em linha dupla –, usados nas memórias tipo DDR, DDR2, DDR3, DDR4 e nas DDR5.

Exemplificando

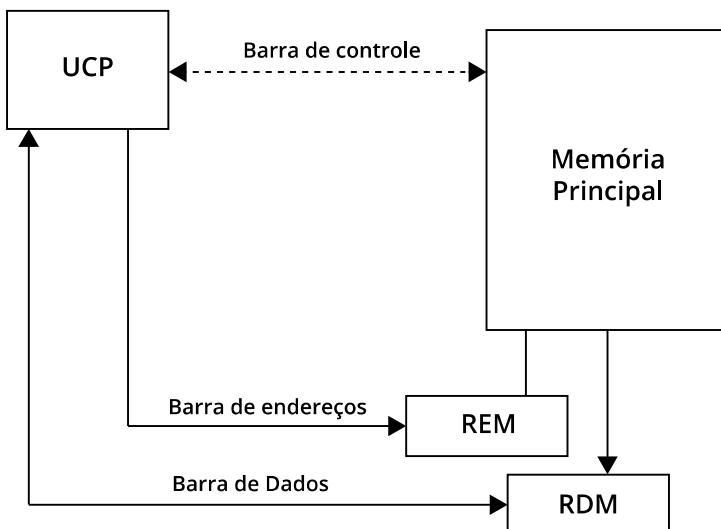
Podemos citar como exemplos de memórias do tipo DDR:

- DDR (*double data rate*): transfere dois lotes de dados entre processador e memória por ciclo de *clock*.
- DDR-2: transfere quatro lotes de dados por ciclo de *clock* e apresenta um consumo de energia menor do que a DDR.
- DDR-3: transfere oito lotes de dados por ciclo de *clock*, trabalha com *clocks* que vão de 800 a 2.133 MHz e consome, ainda, menos energia que a DDR2.
- DDR-4: trabalha com *clocks* entre 2.133 até 4.266 MHz. O que representa um ganho enorme de velocidade, já que temos uma quantidade muito maior de transferências num mesmo espaço de tempo, e ainda consome menos energia que a DDR3.
- Ainda temos novas tecnologias de memórias sendo desenvolvidas e lançadas no mercado com maiores velocidades e capacidades, como, por exemplo, as memórias GDDR5.

Importa destacar que qualquer tipo de memória possui dois tipos de operação a serem realizados, sendo eles: operações de leitura e escrita. A operação de leitura consiste na recuperação dos dados armazenados

na memória, enquanto que a operação de escrita consiste no armazenamento, em si, (a gravação dos dados) na memória. A operação de leitura é considerada não destrutiva, uma vez que ela apenas copia as informações de uma célula da memória principal para a UCP; já a operação de escrita permite a gravação da informação na memória principal, sendo, assim, uma operação destrutiva, pois o conteúdo anteriormente na memória é sobreescrito. A Figura 2.9 ilustra a organização da CPU e a memória principal com seis barramentos, que permitem a entrada e a saída de dados da memória principal para a UCP, a fim de que sejam realizadas as operações de leitura e escrita.

Figura 2.9 | Entrada e saída entre a UCP e MP



Fonte: elaborada pelo autor.

Observe que há três barramentos ligados aos registradores que interligam a MP e a UCP. Isso ocorre porque os registradores são memórias com maior velocidade se comparadas à MP, e isso permite a redução da ociosidade do processador, ou seja, o tempo de espera por informações a serem processadas. Como cada célula da memória é endereçada, são necessários dois registradores para o armazenamento, um para o armazenamento do endereço de memória e outro para os dados.

Memória ROM

A memória ROM – *ready only memory* – também é uma memória principal do computador, mas com função apenas de leitura cujo conteúdo é gravado apenas uma vez e não é alterado. Essa memória também tem como característica ser uma memória não volátil, ou seja, não é apagada quando desligamos o computador. Nela, são gravados os programas de inicialização de um computador, que são chamados também de “*Firmware*” (HARDWARE, 2015).

São três os principais programas gravados em uma memória ROM:

- BIOS (*basic input output system*): sistema básico de entrada e saída; é onde ficam gravadas as instruções para que o processador da máquina possa reconhecer os dispositivos básicos de entrada e saída.
- POST (*power on self test*): programa de autoteste que faz a verificação e o teste quando o computador é ligado; realiza diversas ações sobre o *hardware*, reconhecendo e contando a quantidade de memória, os dispositivos de entrada e saída conectados, entre outros.
- SETUP: programa que altera os parâmetros armazenados na memória de configuração (CMOS).

A memória ROM é classificada de acordo com os dados que são gravados e/ou regravados nela.

De acordo com Fávero, 2011, as memórias ROM podem ser classificadas em:

- PROM (*programmable read-only memory*): a gravação de dados nesse tipo é feita uma única vez, e os dados gravados na memória PROM não podem ser apagados ou alterados.
- EPROM (*erasable programmable read-only memory*): essas memórias permitem a regravação de dados, que é feita por meio de emissão de luz ultravioleta que apaga por completo os dados gravados, permitindo, após isso, uma nova gravação.
- EEPROM (*electrically-erasable programmable read-only memory*): permite a regravação de dados, feitos eletricamente, não sendo necessário mover o dispositivo para que a regravação ocorra.
- EAROM (*electrically-alterable programmable read-only memory*): os dados gravados nesta memória ROM podem ser alterados aos poucos, razão pela qual esse tipo é, geralmente, utilizado em aplicações que exigem apenas reescrita parcial de informações.

- Flash-ROM: as memórias Flash-ROM também podem ser vistas como um tipo de EEPROM, no entanto, o processo de gravação e regravação é muito mais rápido. Nesse tipo de memória, os dados devem ser totalmente apagados e não é permitida a sua gravação parcial.

Embora alguns tipos de memória ROM permitam que seus dados sejam apagados para que novos sejam gravados, eles têm um funcionamento diferente da memória RAM. Em uma memória ROM, o processo de gravação é lento e os dados gravados são basicamente *Firmwares* ou programas que executam determinadas funções. Na memória RAM, um novo dado é gravado imediatamente na memória e pode ser de qualquer tipo, por exemplo, os dados resultantes do processamento de cálculos.

Reflita

Observamos que existem dois tipos de memórias principais, a memória RAM e a memória ROM. A memória RAM é volátil, é apagada quando o computador é desligado, recebe os dados que são inseridos no computador e permite que seja executado o processamento dos dados. Já a memória ROM é não volátil, ou seja, não é apagada quando o computador é desligado, e nela, em geral, são gravados os “*Firmwares*”, que são os programas de inicialização de um computador.

Esta seção apresentou as memórias e as diferenças entre elas, destacando que existem memórias com diversos objetivos e velocidades. Cada vez mais, os processadores têm evoluídos e requerem dispositivos de armazenamento eficientes. Agora, você já é capaz de solucionar a situação-problema desta seção. Preparado? Então, boa sorte!!

Sem medo de errar

Como sugestão de resposta, deve-se pesquisar alguns fabricantes de memória e especificar o usuário e uma quantidade mínima para ser utilizada de acordo com a aplicação do usuário. Por meio das referências citadas pelo fabricante, no site **UserBenchmark** podemos ter acesso à listagem de memórias e seus tamanhos.

Para o melhor entendimento da RAM e suas velocidades de acordo com o processador, recomendamos a leitura do artigo no site da Intel: *Desmistificando a tecnologia de pcs: RAM versus processador: o que é mais importante para as tarefas da sua empresa?*

É possível estabelecer parâmetros para quantidades de memória RAM de acordo com a aplicação desejada. Como uma sugestão de tabela, pesquise a quantidade de memória de acordo com o perfil de um usuário, como segue:

Relatórios consolidados e testes encomendados pelo fabricante:

Como sugestão, é possível considerar um quadro de classificação de uso de quantidade de memória:

Quadro 2.3 | Recomendações de capacidade de memória de acordo com o uso

Requisitos de Memória (Ideal)	Usuário frequente	Jogador	Usuário Profissional
DESKTOP	8 GB+ (WINDOWS e MAC_OS)	16 GB+ (WINDOWS)	16 GB+ (WINDOWS e MAC_OS)
NOTEBOOK	8 GB+ (WINDOWS e MAC_OS)	16 GB+ (WINDOWS)	16 GB+ (MAC_OS)
Uso			
	Email, Internet	A maioria dos jogos atuais exige 8-16 GB de RAM	Design Gráfico, Modelagem 3D
	Baixar e organizar fotos músicas, filmes e TV		Suite completa Produtos Office
		Alto desempenho em Jogo	
			Software Corporativo
	Suite completa Office		CRM, Produção etc.
	Word, Excel, Power Point etc.		
			Programação de Softwares
	Software Corporativo		Engenharia de Som
	CRM, Produção etc.		Design
			Páginas WEB avançadas
			Desenvolvimento de Banco de Dados
			Edição de Imagens

			Edição de Vídeos
			Computação Gráfica/ Animação etc.

Fonte: elaborado pelo autor.

Faça valer a pena

- 1.** A memória RAM (*random access memory*) faz o armazenamento dos dados inseridos no computador, dados dos programas e os próprios programas.

Sobre a memória RAM, analise as afirmativas a seguir:

- I. Ela permite o acesso à memória secundária, disponibilizando os dados gravados.
- II. Ela permite gravar programas de inicialização do computador.
- III. Ela é uma memória com maior capacidade de armazenamento quando comparada a um SSD.

É correto o que se afirma em:

- a. I e II, apenas.
- b. I e III, apenas.
- c. II e III, apenas.
- d. I, apenas.
- e. III, apenas.

- 2.** Na hierarquia de memória, as memórias são organizadas de acordo com velocidades, custo e capacidade.

Quanto à hierarquia de memória, assinale a alternativa correta.

- a. A cache L2 é mais rápida que a memória RAM, porém, com mais baixa capacidade.
- b. Os registradores possuem baixa capacidade de armazenamento e um alto custo.
- c. A memória secundária possui capacidade de armazenamento menor que a RAM.

- d. Uma cache L1 é mais lenta que a L3, e a L2 é mais rápida que L1 e L3.
- e. A memória cache serve como intermediária entre a memória secundária e o processador.

3. As memórias são consideradas não voláteis quando os dados armazenados não são perdidos no desligamento de energia. Quando os dados são perdidos, denomina-se memórias voláteis.

Quanto às memórias voláteis e não voláteis, analise as asserções a seguir e a relação proposta entre elas.

- I. A memória RAM é uma memória não volátil.

PORQUE

- II. Os dados armazenados ficam disponíveis na memória RAM para a memória cache repassar para a ULA.

A respeito dessas asserções, assinale a alternativa correta.

- a. A asserção I é uma proposição verdadeira e a II é falsa.
- b. A asserção I é uma proposição falsa e a II é verdadeira.
- c. As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- d. As asserções I e II são proposições verdadeiras e a II justifica a I.
- e. As asserções I e II são proposições falsas.

Seção 3

Memória secundária

Diálogo aberto

Olá, caro aluno! Você já teve a oportunidade de conhecer a memória principal de um computador, os conceitos de memória volátil e não volátil, viu o que são os registradores, a memória cache, a memória principal e, em detalhes, as memórias RAM e ROM. Você também já viu que os computadores baseados na arquitetura de von Neumann são compostos de uma unidade central de processamento chamada de CPU, de memórias e de unidades de entrada e saída de dados.

Para que um processador possa executar o processamento nessa arquitetura é necessária a utilização de memória, e como você já teve oportunidade de ver, uma das grandes conquistas dessa arquitetura foi poder ter na mesma memória dados e o processamento de programas.

Essa tecnologia dispõe de dois tipos de memória: a memória principal, representada basicamente por memórias RAM e ROM, e as memórias secundárias.

As memórias secundárias são responsáveis por armazenar as informações para uso posterior, pois elas não se apagam quando o computador é desligado, são do tipo não voláteis. Além disso, elas podem ser alteradas e regravadas quantas vezes for necessário (OLIVEIRA, 2007).

Dando continuidade às questões vistas pela empresa de fabricação de microprocessadores, considerando o conceito de “cidades inteligentes”, pretende-se em uma cidade implantar um sistema de informação com os dados médicos de cada habitante. Esse sistema será interligado com o de atendimento de urgência e poderá ser acessado pela equipe médica dentro da ambulância, no momento do atendimento. Para isso, é necessário que o setor de pesquisa e desenvolvimento consiga identificar dispositivos com grande capacidade de armazenamento de dados e que possam ser acessados de forma rápida, por meio de sistemas que se interconectem entre si pela internet. O cruzamento de dados e informações disponibilizadas nesse sistema poderá salvar vidas e representa um grande avanço para a saúde da população. Estes dispositivos de armazenamento têm que oferecer capacidade aumentada de armazenamento e velocidade de acesso, além de eficiência no consumo de energia.

Faça uma descrição detalhada da capacidade de modelos SSDs de alguns fabricantes, apresentando suas respectivas capacidades de memória, velocidades de taxa de transferência de dados dessas memórias e demais capacidades técnicas. Veja como elas podem atender aos requisitos dos sistemas propostos na situação-problema, como o acesso rápido às informações médicas de um paciente pela equipe médica da ambulância no momento do atendimento.

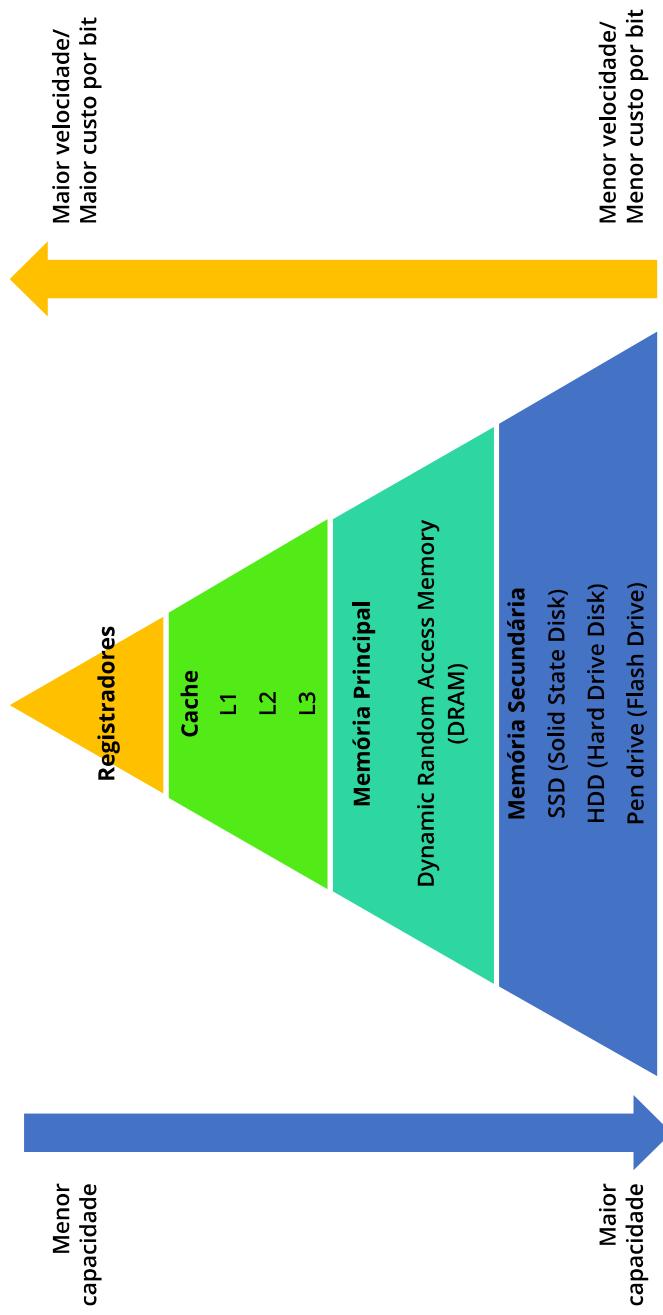
Vá em frente e pesquise estas novas tecnologias!

Não pode faltar

Nos computadores, como você pôde ver, a memória não é uma única peça isolada: existem vários tipos de memórias. A velocidade dos processadores, de suas CPUs e de suas estruturas requerem vários tipos de memórias, cada qual com sua função específica (FÁVERO, 2011). Você também viu que essas memórias são classificadas em memória principal e memória secundária. Além desses dois tipos de memória ainda temos a memória cache e os registradores da CPU (MONTEIRO, 2007).

Relembrando: as memórias de um computador podem variar também em sua tecnologia, capacidade de armazenamento, velocidade e custo, e são interligadas de forma estruturada, compondo um subsistema de memória. Este subsistema organiza os diversos tipos de memória hierarquicamente (FÁVERO, 2011). A Figura 2.10 ilustra a hierarquia de memória: pode-se observar que as memórias secundárias estão localizadas na base da pirâmide, as quais apresentando maior capacidade e menor velocidade de acesso quando comparadas às memórias principal, cache e registradores. Além disso, as memórias secundárias têm menor custo por bit; assim, são utilizadas para armazenamento de um maior volume de dados de modo permanente, porém apresentam maior tempo de acesso a esses dados (menor velocidade).

Figura 2.10 | Posição das memórias secundárias na hierarquia de memórias



Fonte: elaborada pelo autor.

Neste contexto, as memórias secundárias são responsáveis por armazenar dados para uso posterior, pois elas não se apagam quando o computador é desligado, são do tipo não voláteis. Além disso, podem ser alteradas e regravadas quantas vezes for necessário (OLIVEIRA, 2007).

Outra característica das memórias secundárias é que elas não são endereçadas diretamente pelo processador e, por este motivo, os dados armazenados nessas memórias precisam ser carregados na memória principal para serem processados. Em geral, elas são memórias não voláteis e permitem gravar os dados permanentemente. Estão nessa categoria de memórias os *solid state disk* (SSD), discos rígidos (HD), CDs, DVDs, pen drives e outros (VELLOSO, 2014). A memória secundária também é chamada de memória de massa, por apresentar uma capacidade de armazenamento muito superior à das outras memórias (FÁVERO, 2011).

A figura a seguir ilustra os dispositivos de memórias externa mais utilizados, o HD e o SSD.

Figura 2.11 | HD e SSD



Fonte: Shutterstock.

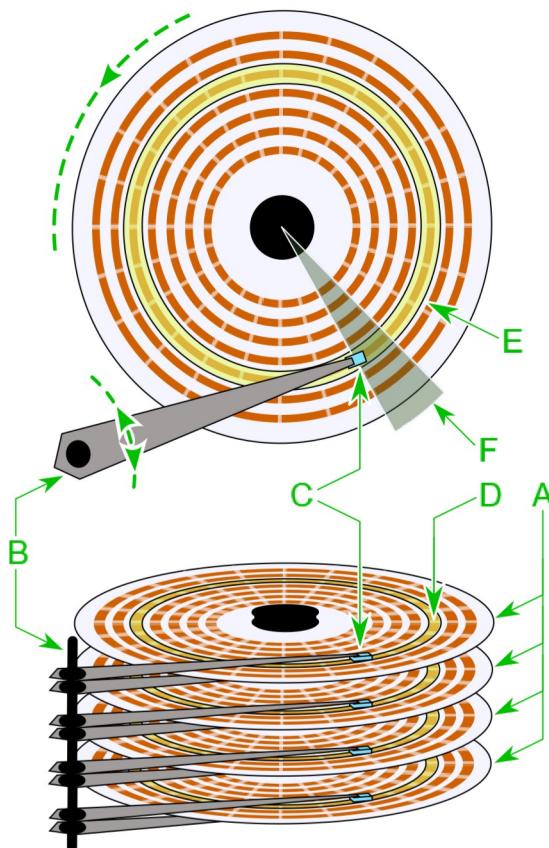
Os discos rígidos são dispositivos que utilizam o meio magnético para realizar o armazenamento dos dados. Eles são considerados a base de memória externa para um sistema de computação e são os mais utilizados atualmente. Em geral, os computadores de grande porte (computador pessoal – PC) apresentam um ou mais discos rígidos instalados.

O disco rígido (do inglês *hard disk* – HD) é composto de múltiplas superfícies circulares empilhadas também chamada de pratos. Essas superfícies têm camadas de material magnetizável, e são divididas basicamente em trilhas,

setores e cilindros. Existem, ainda, as cabeças de leitura e escrita (gravação), disposta em um suporte chamado de braço. A Figura 2.12 ilustra o esquema interno de um HD. A cabeça de leitura e escrita fica próxima ao disco (sem tocar a superfície); o braço (B) possibilita a movimentação da cabeça de leitura e escrita, movendo-se por meio de um sistema eletromagnético.

As trilhas (E) de um disco são divisões circulares no disco, e os setores (F) são divisões nas trilhas. Cada setor dispõe de 512 KB para armazenamento. O cilindro (D) é um conjunto de trilhas de todos os discos, em uma mesma posição nos diferentes pratos (A).

Figura 2.12 | Esquema simplificado de um HD. A: pratos; B: braço; C: cabeça de leitura e gravação; D: cilindro; E: trilhas; F: setor



Fonte: Wikimedia.

A partir dessas informações podemos calcular o tamanho de um disco, multiplicando os números de cilindros, cabeças, setores e trilhas por 512 bytes (cilindros × número de cabeças × números de setores por trilhas × 512 bytes).

As gravações em disco são realizadas via sinais elétricos que as cabeças de leitura e escrita emitem, convertendo esses sinais em bits (0s ou 1s).

Os discos rígidos, também chamados de HDs, são o tipo de memória secundária mais usado, pois acompanham praticamente todos os computadores e notebooks, desde os mais antigos até os dias atuais, e ainda são considerados o principal meio de armazenamento de dados. Em geral, é no disco rígido que são gravados os sistemas operacionais e demais arquivos de um computador.

Ele se comunica com o computador por uma interface, que é composta por conectores. Esses conectores podem ser de diferentes tipos e padrões, cada qual com sua característica específica, como você poderá ver (EQUIPE DIGERATI, 2009).

O disco rígido é fechado a vácuo, para evitar acúmulo de resíduos. Esses discos apresentam rápidos movimentos de rotação em torno de um eixo central. Por exemplo: alguns HDs de servidores podem realizar até 15 mil rotações por minuto (RPM) (Dell, 2020) e de computadores pessoais até 7200 RPM (SEAGATE, 2018). Quanto maior o número de rotações por minuto mais rápido a cabeça de leitura alcança os dados. Importante destacar que qualquer tipo de sujeira pode comprometer o funcionamento do disco, e por isso eles são fechados a vácuo.

Um outro conceito relevante relacionado aos discos é o RAID (*redundant array of independent disk*) ou conjunto redundante de discos independentes. O RAID consiste em um método muito utilizado principalmente em servidores, mas não somente neles. Esse método aumenta o desempenho e a confiabilidade dos HDs, combinando múltiplos discos rígidos e formando um virtual. Isso ocorre por meio de divisões e replicações (cópia) de dados em vários discos. Além disso, dependendo do tipo de RAID utilizado, é possível aumentar o desempenho do disco rígido, bem como dificultar a perda de dados (MONTEIRO, 2007). Alguns dos níveis de RAIDs são os seguintes:

- RAID 0 (*stripping* ou distribuição): no RAID 0, os dados são divididos em pelo menos dois discos. Esse nível permite realizar a leitura e a escrita (gravação) mais rapidamente do que uma configuração que não utiliza RAID. Nesse nível não há tolerância a falhas, os dados são apenas distribuídos entre os discos e não há redundância dos dados. Logo, se algum dos discos danificar ou falhar, os dados serão perdidos. O RAID 0 possibilita o aumento de desempenho, pois o

acesso aos dados nos discos é distribuído, minimizando as sobrecargas de acesso ao disco. Além disso, esse nível favorece a capacidade de armazenamento.

- RAID1 (*mirroring* ou espelhamento): os dados são espelhados, ou seja, são replicados em dois discos. A velocidade de gravação não é maior. A vantagem desse nível está na redundância dos dados, uma vez que eles permanecem disponíveis mesmo na ocorrência de falhas em um dos discos. O disco danificado pode ser substituído por um outro disco, e os dados do disco existente são copiados para o novo disco.
- RAID 5 (*parity* ou paridade): inclui características do RAID 0 com a paridade. Em outras palavras, utiliza a divisão dos dados com paridade, requerendo no mínimo três discos. Os dados não são perdidos na ocorrência de uma falha de algum dos discos. A velocidade de escrita é menor que a leitura devido à necessidade de se realizar cálculos de paridade e armazenamento.
- RAID 6 (divisão dos dados com duplo pareamento): combina a distribuição dos dados com a paridade dupla, requerendo no mínimo quatro discos. A paridade extra (segunda paridade) possibilita a perda de dois discos de uma única vez, sem haver perda dos dados. A redundância dos dados é a prioridade para o RAID 6.
- RAID 10 (divisão dos dados e espelhamento RAID 1 + 0): utiliza o espelhamento dos dados entre os discos para depois realizar a distribuição. São necessários ao menos quatro discos nesse nível. Primeiramente, os discos são espelhados (RAID 1) e, após isso, os dados são divididos (RAID 0). Apresenta melhor desempenho que os níveis RAID 5 e 6, uma vez que dispensa cálculos de paridade e armazenamento. Suporta a falha de um disco por espelhamento, porém, na ocorrência de falha no espelhamento de um dos discos, os dados de todos os discos serão perdidos.

Solid-State Drive (SSD)

O SSD, sigla do inglês *Solid-State Drive*, é um tipo de dispositivo para armazenamento de dados. Atualmente, muitos usuários já realizaram a atualização de seus HDs para esses dispositivos, devido ao grande ganho desempenho que ele proporciona, pois apresenta alta velocidade de acesso e consumo de energia reduzido. Isso acontece pela ausência de peças móveis, como bobinas e cabeças de leitura e gravação, encontrados nos HDs. Não há necessidade de rotacionar o disco em busca de arquivos, pois o SSD possibilita a busca de arquivos de forma mais rápida. Nesses dispositivos são usados chips de memória *Flash* para o armazenamento de dados, o que os torna mais econômicos no consumo de energia. Algumas desvantagens de SSDs são o custo maior por bit em relação aos HDs. A tecnologia SSD começou a ser empregada de forma ampla em dispositivos portáteis, tais como notebooks ultrafinais (ultrabooks) e tablets.

Figura 2.13 | Diferentes tipos de SSD, um SSD clássico e o SSD M.2 NVMe



Fonte: Shutterstock.

Os SSDs não utilizam peças móveis, por isso são considerados silenciosos. Essa tecnologia é baseada no uso de chips de memória *Flash* não voláteis. Além da ausência de ruídos, apresentam boa resistência física na ocorrência de trepidações e quedas, e são mais leves que os HDs. Por causa dessas características eles são aplicados cada vez mais em notebooks, ultrabooks e tablets, pois são portáteis e sujeitos a trepidações e quedas. Observe, na Figura 2.14, a parte interna de um HD (à esquerda) e um SSD (à direita).

Figura 2.14 | Disco rígido magnético (HD) e *Solid-State Disk* internamente



Fonte: Shutterstock.

Nas memórias *Flash*, os dados não são perdidos quando não há mais fornecimento de energia, pois são do tipo não voláteis (FÁVERO, 2011). Os fabricantes de dispositivos SSD utilizam memórias *Flash* para produzir este dispositivo.

Existem dois tipos de memória *Flash*: o **Flash NOR** (*Not OR*) e o **Flash NAND** (*Not AND*), como segue:

- O tipo NOR: este tipo de memória permite acessar dados em posições diferentes da memória de maneira rápida, sem necessidade de ser sequencial. É usado principalmente em chips de BIOS e em *firmwares* de smartphones.
- O tipo NAND: a memória NAND pode armazenar mais dados que a memória NOR, considerando blocos físicos de tamanhos equivalentes. É um tipo mais barato de memória e é mais utilizado em SSD. Esse tipo de memória também trabalha em alta velocidade, mas executa o acesso sequencial às células de memória e as trata em conjunto, isto é, em blocos de células.

Assimile

As memórias secundárias não podem ser endereçadas diretamente, e por isso seu conteúdo tem que ser transferido para a memória principal para poder ser processado. Existem vários tipos de dispositivos e

padrões de memória secundária, cada um com sua capacidade de transferência entre o próprio dispositivo e a memória principal.

Os SSDs são exemplos de dispositivos que apresentam muitas vantagens em relação aos HDs, como alta velocidade de transmissão de dados e baixo consumo de energia, itens muito importantes em sistemas computacionais.

Existem outros dispositivos de armazenamento de dados, tais como as mídias ópticas, os CDs, DVDs e Blu-ray. Outro dispositivo é o pen drive, um dispositivo portátil de armazenamento com memória *flash* do mesmo tipo das usadas em dispositivos SSD, acessados quando conectados a uma porta USB. Existem diversos modelos desses dispositivos, com diversos tamanhos de capacidade de armazenamento. Os mais atuais dispõem de terabytes (TB) de memória (KINGSTON, 2017). O pen drive tornou-se mais eficiente do que os CDs e até do que os DVDs, pois sua capacidade de armazenamento pode ser bem superior ao dessas mídias, o que o fez extremamente popular. Hoje, todavia, já existem os SSD portáteis, como o SSD portátil SanDisk (WESTERN DIGITAL TECHNOLOGIES INC., [s.d.]), com capacidade de armazenamento de 2TB. Essas mídias são consideradas mídias de alta velocidade na gravação e leitura dos dados. Em relação ao armazenamento de dados, há alguns anos vem aumentado significativamente essa prática pela nuvem.

Refita

Pensando que na atualidade temos o armazenamento de dados a partir de recursos computacionais – processamento, memórias e máquinas virtuais – na nuvem, por que hoje nos preocupamos com a capacidade de armazenamento e o desempenho das memórias secundárias em nossos computadores pessoais e notebooks?

Atualmente no mercado existem algumas variações de SSDs, principalmente os relacionados às interfaces (também chamadas de conectores), e protocolos utilizados para a transmissão e recepção dos dados. Existe uma gama de conectores que surgiram ao longo dos anos, os quais eram utilizados para a conexão de um HD à placa-mãe, ou até mesmo à conexão de outros dispositivos. Cada uma dessas interfaces utilizadas em HD ou SSD apresenta particularidades relacionadas à velocidade de transferência dos dados. A seguir, vamos listar alguns desses padrões de interfaces.

Padrão SCSI

O padrão SCSI, sigla do termo em inglês *Small Computer Systems Interface*, foi criado para permitir a comunicação entre dispositivos com confiabilidade de transmissão e velocidade rápida. Embora seu uso tenha sido mais comum em HDs, esse padrão foi usado também para conectar outros tipos de dispositivos, como impressoras, scanners e unidades de fita usadas em backups. Esse é um padrão antigo, desenvolvido no final da década de 1970 e lançado oficialmente em 1986. A tecnologia SCSI foi muito importante, pois permitia uma taxa alta de transferência de dados, dando, assim, suporte ao avanço da velocidade dos processadores. Essa tecnologia foi mais aplicada em servidores do que em computadores pessoais, e ainda hoje é utilizada devido a sua confiabilidade na transferência de dados.

Padrão IDE/ATA

Os padrões IDE (do inglês *Integrated Drive Electronics*) e ATA (do inglês *Advanced Technology Attachment*) foram lançados no ano de 1986 (no caso do IDE) e padronizado em 1990 (no caso do padrão ATA) (CARMONA, 2006). O padrão IDE foi o primeiro que integrou ao HD a controladora do dispositivo, o que representou uma grande inovação, reduzindo os problemas de sincronismo e tornando seu funcionamento mais rápido e eficiente. Seus cabos de conexão eram menores, o que facilitou sua aplicação em computadores pessoais (EQUIPE DIGERATI, 2009).

Quando os dispositivos IDE foram lançados não havia uma definição de padrão para esse dispositivo, o que gerou problemas de compatibilidade entre os diversos fabricantes. Para resolver este problema, o ANSI (*American National Standards Institute*) aplicou as correções necessárias para a padronização dessa tecnologia, e em 1990 foi criado o padrão ATA (*Advanced Technology Attachment*). Por ser o nome IDE o mais conhecido, ele permaneceu, surgindo o termo IDE/ATA (PEREIRA, 2009).

Esta tecnologia foi ainda renomeada para PATA (*Parallel ATA*) para ser diferenciada da sua sucessora, a tecnologia SATA (vide item a seguir). Na tecnologia ATA os dados são transmitidos por cabos de 40 ou 80 fios paralelos, o que resulta em cabos maiores, e os dados são transmitidos e recebidos por esses fios, ou seja: eles podem tanto transmitir como receber dados, o que torna o processo menos eficiente em relação a seu sucessor SATA (CARMONA, 2006).

Padrão SATA

O padrão SATA, do inglês *Serial Advanced Technology Attachment*, é o sucessor do padrão ATA e funciona de forma serial, diferente do IDE/ATA, que funciona de forma paralela. Como ele utiliza dois canais separados, um para enviar e outro para receber dados, isto reduz quase totalmente os problemas de sincronização e interferência, permitindo uma capacidade maior de transferência de dados. A maioria dos HDs utiliza a interface SATA para a conexão. Os SSDs também utilizam, porém existe vários tipos de conectores que podem influenciar as velocidades de transferência dos dados.

Seus cabos têm apenas um par de fios para envio de dados e outro par para o recebimento dos dados, que são transferidos em série. Também apresentam outros três fios para a alimentação de energia do dispositivo, totalizando apenas sete fios no cabo, o que resulta em cabos com diâmetro menor, auxiliando na ventilação e diminuição da temperatura dentro do computador (PEREIRA, 2009).

Além desses, existe uma classificação do padrão SATA de acordo com a capacidade de transferência de dados, que é medida em gigabits por segundo. O SATA, primeiro a ser lançado e chamado de SATA I, vai até 1,50 Gb/s (gigabits por segundo); o SATA II atinge 3 Gb/s; e o SATA III atinge 6 Gb/s. Cabe ressaltar que a cada dia novas capacidades são lançadas; além disso novas tecnologias são introduzidas, com o objetivo de alcançar cada vez mais capacidade de armazenamento, bem como de melhorar a velocidade de transmissão e eficiência no consumo de energia. O SATA express é a evolução do SATA e suporta também o barramento PCI express. Apresenta um conector híbrido composto de conectores SATA e PCI express (periféricos *Component interConnect Express*).

Existe, ainda, um protocolo *Non-Volatile Memory express* (NVMe), o qual possibilita a conexão de SSDs ao barramento PCI.

Tabela 2.1 | Capacidades dos padrões IDE/ATA, SATA, SATA II, SATA III, SATA express, PCI express

Padrão	Quantidade de Pinos	Velocidade de transferência (em Gbit/s)
IDE/ATA	40	1,33
SATA	7	1,5
SATA II	7	3
SATA III	7	6
SATA express	18	>18

Fonte: elaborada pelo autor.

Exemplificando

Mesmo com essa gama de variações de interfaces com diferentes velocidades, é importante sempre verificar a velocidade dos dispositivos. Por exemplo: de nada adianta você ter um dispositivo com maior velocidade que a velocidade de transferência da interface, ou vice-versa.

Esta seção tratou de questões importantes: as memórias secundárias e seu funcionamento. Assim, com o conhecimento adquirido, você já tem as informações para diferenciar um SSD de um HD e descrever as suas características. Agora você já é capaz de solucionar a situação-problema desta seção. Preparado? Bom trabalho!

Sem medo de errar

Para a resolução da situação-problema, você deve realizar um levantamento dos principais fabricantes de SSDs e verificar as velocidades dos SSDs e os tipos de interfaces utilizadas. Além disso, deve realizar uma análise das velocidades das interfaces, bem como das velocidades de leitura e gravação dos SSDs.

Para uma análise comparativa sobre memórias SSD você poderá acessar o site da Kingston (2017, [s.p.]).

Como uma sugestão, pode ser apresentado um quadro com as seguintes características:

Quadro 2.4 | Características de SSDs disponíveis no mercado

	Modelo	Capacidade	Velocidade leitura/gravação	Interface
SSD	Kingston - 480G DC500R	480 GB	555MBs/520MBs	SATA Rev. 3.0 (6Gb/s)
SSD	SanDisk SSD PLUS	480 GB	535 MBs / 445 MBs	SATA Rev. 3.0 (6Gb/s)
SSD	WDS100T2G0A	1TB	545MB/s / 430 MB/s	SATA III 6 Gb/s
SSD	SSD Intel 660P Series	1 TB	1800 MB / 1800 MB / s	PCIe NVMe 3.0 x4 (4GB/s)

Fonte: elaborado pelo autor.

Faça valer a pena

1. Estão na categoria de memórias secundárias dispositivos como discos rígidos e pen drives, entre outros.

Sobre as memórias secundárias, avalie as afirmativas a seguir:

- I. São memórias não voláteis, pois não se apagam quando o computador é desligado.
- II. São memórias consideradas de massa por apresentarem capacidade de armazenamento superior em relação a outros tipos de memória.
- III. Os SSDs têm maior capacidade de processamento e menor custo por bit na hierarquia de memória.
- IV. Essas memórias podem ser regravadas e alteradas de acordo com a necessidade.

É correto o que se afirma em:

- a. I, II e III, apenas.
- b. I, II e IV, apenas.
- c. I, III e IV, apenas.
- d. II, III e IV, apenas.
- e. I, II, III e IV.

2. O método RAID possibilita o aumento de confiabilidade, segurança e o desempenho dos discos. Dependendo de cada nível de RAID, há um resultado diferente.

Com base nos níveis de RAID, analise as afirmativas a seguir:

- I. O RAID 0 garante a tolerância a falhas, uma vez que os dados são replicados entre os discos.
- II. O RAID 1 possibilita a distribuição dos dados entre os discos, porém esses dados não são redundantes.
- III. O RAID 5 combina divisão de dados com paridade e requer no mínimo três discos.
- IV. O RAID 10 realiza o espelhamento dos dados para depois distribuí-los entre os discos, se o espelhamento falhar, os dados são perdidos.

É correto o que se afirma em:

- a. I e II, apenas.
- b. II, III e IV, apenas.
- c. III e IV, apenas.
- d. III, apenas.
- e. IV, apenas.

3. Os SSDs e HDs são dispositivos que estão na base na pirâmide da hierarquia de memória. Apesar de serem considerados dispositivos de armazenamento em massa, eles apresentam algumas particularidades.

Sobre os dispositivos de memória secundária SSD e HD, analise as asserções a seguir e a relação proposta entre elas.

- I. Os SSDs são considerados dispositivos com maior velocidade e silenciosos quando comparados ao HD.

PORQUE

- II. Os SSDs não têm partes móveis que requeiram rotações para alcançar os dados no disco, como no HD.

A respeito dessas asserções, assinale a alternativa correta.

- a. A asserção I é uma proposição verdadeira e a II, falsa.
- b. A asserção I é uma proposição falsa e a II, verdadeira.
- c. As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- d. As asserções I e II são proposições verdadeiras e a II justifica a I.
- e. As asserções I e II são proposições falsas.

Seção 4

Dispositivos de entrada e saída

Diálogo aberto

Você já aprofundou seus conhecimentos sobre a Unidade Central de Processamento – CPU, sua unidade lógica e aritmética, seus registradores e seus barramentos, o que são os bits de um processador, sua tecnologia CISC ou RISC, conheceu o que é memória principal, memória cache, os tipos de memória RAM, a memória ROM e o que significa memória volátil e não volátil. Você pôde conhecer também os conceitos de processamento e de memória principal, além das memórias secundárias e seus dispositivos, entre eles os HDs e os mais recentes dispositivos de SSDs, que proporcionam armazenamento com velocidade superior de acesso e baixo consumo de energia. Diante de tais conhecimentos, você está quase apto a descrever um computador completo e, para isso, você vai estudar agora os dispositivos de entrada e saída e como eles se comunicam com as demais partes do computador. Tais conceitos são fundamentais para que você possa dar continuidade às questões vistas pela empresa de fabricação de microprocessadores.

Considerando o conceito de “cidades inteligentes”, uma determinada cidade implantará um sistema de informação de atendimento médico, composto pelos dados médicos de cada habitante da cidade. Ele será interligado ao sistema de agendamento de consultas, acompanhamento de exames, encaminhamento para unidades de especialidades e emissão eletrônica de receita médica – que poderá ser acessada pelas farmácias públicas ou comerciais –, histórico médico do paciente e a interligação de aparelhos médicos colocados no local do paciente e monitorados pela equipe médica a distância, entre outros. Para isto, é necessário que o sistema funcione em um computador servidor que atenda à demanda de acessos e troca de informações entre os diversos sistemas. Os dispositivos que acessarão esse sistema, aparelhos médicos e de diagnóstico estarão conectados ao sistema, alimentando as informações médicas e retornando orientações para as pessoas, podendo até definir padrões de funcionamento desses dispositivos, comandados pela equipe médica de forma on-line em tempo real, com o uso da internet. Portanto, é de fundamental importância a capacidade de entrada e saída de informações nesse computador. Para isso, é necessário entender como funcionam a entrada e a saída de dados, os tradicionais dispositivos e as novas tecnologias disponíveis, que contribuem com sistemas complexos e interligados entre si.

O desafio consiste em apresentar as características de um computador servidor que permita o processamento com alta performance e baixo consumo de energia, e que seja capaz de atender à demanda de acessos às informações e retorno das solicitações com rapidez. Para isso, acesse as especificações sobre o tipo de processador do fabricante e verifique quantos núcleos apresenta, quais os tipos de memórias RAM, SSDs aplicados e aceitos, e taxas de transferência de entrada e de saída de dados.

Bom trabalho!

Não pode faltar

Como você já viu anteriormente, os computadores atuais são baseados na arquitetura von Neumann, que prevê a capacidade de uma máquina digital armazenar na memória dados e instruções em formato binário, necessários para a execução de uma tarefa. A CPU – Unidade Central de Processamento, busca esses dados e instruções na memória e executa o processamento, e o resultado desse processamento é disponibilizado na memória (FÁVERO, 2011).

Como também já foi visto, nessa arquitetura de computadores estão previstas as unidades de entrada e saída de dados. Essas unidades são compostas por diversos dispositivos e podem ser divididos em (SOUZA FILHO; SANTANA; MEDEIROS, 2014):

- **Dispositivos de Entrada** – no qual podemos inserir/entrar com dados no computador. Exemplo: teclado, mouse, telas sensíveis ao toque (*touch screen*).
- **Dispositivos de Saída** – em que os dados podem ser visualizados. Exemplo: telas e impressoras.
- **Dispositivos de Entrada/Saída** – podem enviar e receber dados, como o disco rígido, pen drives, conexões de internet via cabo e wi-fi, monitores e telas *touch screen*, entre outros (FONSECA, 2007).

Existem diversos dispositivos de entrada e saída que também são chamados de periféricos. A cada dia surgem novos equipamentos que fazem a entrada e saída de dados. Segundo Velloso (2014), os elementos de um computador que garantem interligação do processador com o mundo externo constituem um sistema de entrada e saída, no qual temos:

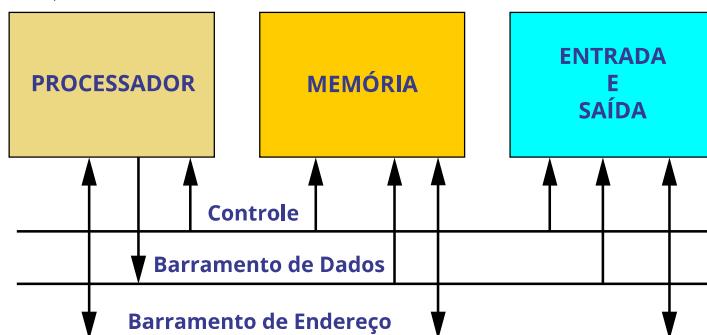
- Barramentos.
- Interfaces.
- Periféricos – dispositivos de entrada e saída (VELLOSO, 2014).

Um processador manipula dados executando ações com o objetivo de obter resultados. São ações comuns à execução de operações aritméticas simples: somar, subtrair, multiplicar e dividir; operações lógicas e as operações de movimentação de dados entre a CPU e a memória. Os componentes do processador são interligados pelos barramentos, que permitem essa movimentação de dados entre a CPU e a memória (MONTEIRO, 2007). Ainda segundo Monteiro (2007), um barramento é o caminho pelo qual trafegam todas as informações de um computador. Existem três tipos principais de barramentos:

- Barramento de dados.
- Barramento de endereços.
- Barramento de controle.

O conjunto desses três barramentos compõe um modelo de barramento de sistema, conforme a figura a seguir.

Figura 2.15 | Modelo de barramento de sistema



Fonte: adaptada de Souza Filho e Alexandre (2014, p. 54).

É preciso que você reforce o entendimento sobre os barramentos, pois eles desempenham uma função direta na entrada e saída de dados, no processamento desses dados e no retorno dos resultados desse processamento. Relembmando:

Barramento de dados

Esse barramento interliga a CPU à memória e vice-versa, para a transferência das informações que serão processadas. Ele determina diretamente o desempenho do sistema, pois quanto maior o número de vias de comunicação, maior o número de bits transferidos e, consequentemente, maior a

rapidez. Os primeiros PCs apresentavam barramento de 8 vias. Atualmente, dependendo do processador, o número de vias pode ser de 32, 64 e até de 128 vias (FÁVERO, 2011).

Barramento de endereços

Interliga a CPU à memória fazendo seu endereçamento e tem o número de vias correspondente à tecnologia de bits do processador, ou seja, nos computadores mais modernos, 32 bits ou 64 bits, permitindo endereçar até quatro GB (gigabytes) de memória em processadores 32 bits e cerca de 16 PB (petabytes) no caso de processadores 64 bits (SOUZA FILHO; SANTANA; MEDEIROS, 2014).

Barramento de controle

Interliga na CPU a Unidade de Controle aos componentes e dispositivos de um computador, componentes de entrada e saída, memórias auxiliares e de armazenamento, entre outros. (MONTEIRO, 2007).

Como visto, o barramento de controle faz a comunicação entre os periféricos de entrada e saída com a CPU do computador. Durante o processamento de um programa, cada instrução é levada à CPU a partir da memória, junto com os dados necessários para executá-la. A saída do processamento é retornada à memória e enviada a um dispositivo, como um monitor de vídeo. A comunicação entre a CPU, a memória e os dispositivos de E/S é feita sempre pelos barramentos (SOUZA FILHO; SANTANA; MEDEIROS, 2014).

Existem muitas diferenças de características entre os diversos periféricos de E/S. Por exemplo, a velocidade de transferência de um teclado ou de um mouse é muito menor do que a velocidade de um HD. Por esse motivo foram criados outros tipos de barramentos, com taxas de transferência de bits diferentes. Existem, atualmente, diferentes tipos de barramentos adotados pelos fabricantes desses dispositivos, e podemos citar:

- **Barramento local:** funciona na mesma velocidade do clock (relógio) do processador. Em geral, interliga o processador aos dispositivos com maior velocidade, memória cache e memória principal.
- **Barramento de sistema:** adotado por alguns fabricantes, faz com que o barramento local faça a ligação entre o processador e a memória cache, a qual se interliga com a memória principal (RAM). Dessa forma, não acontece acesso direto do processador à memória principal. Um circuito integrado auxiliar é usado para sincronizar o

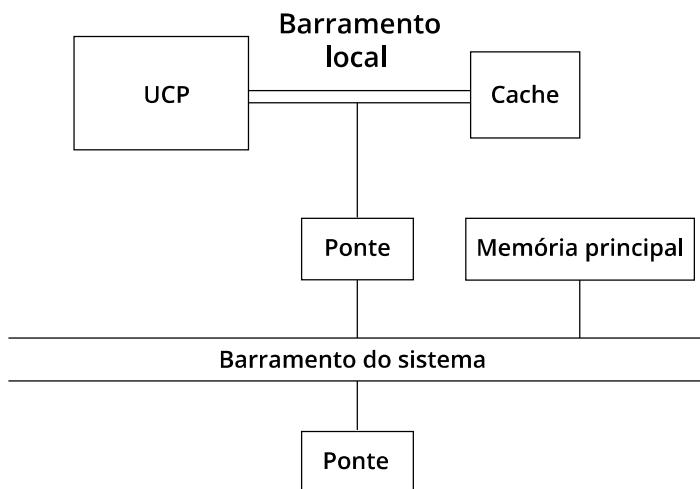
acesso entre a memória cache e a RAM, chamado de ponte e mais conhecido como *chipset*.

- **Barramento de expansão:** também chamado de barramento de entrada e de saída (E/S), é responsável por interligar os diversos dispositivos de E/S aos demais componentes do computador, tais como: monitor de vídeo, impressoras, CD/DVD. Neste caso também é usado um *chipset* para cada dispositivo poder se conectar ao barramento do sistema, e esses *chipsets* (pontes) sincronizam as diferentes velocidades dos barramentos. (FÁVERO, 2011).

Exemplificando

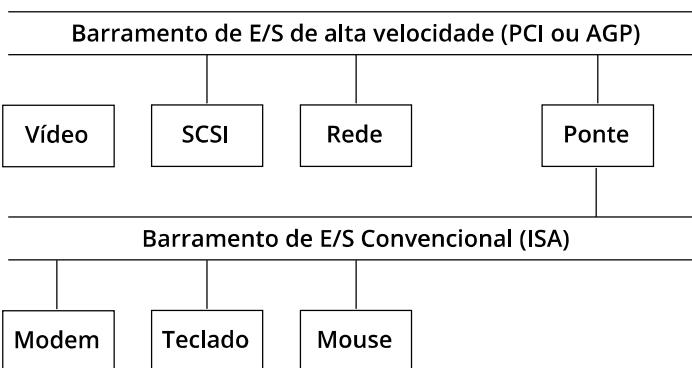
O nome barramento também é usado para identificar o tipo de conector de uma placa de interface de acordo com o número de pinos e números de vias utilizados na comunicação com a placa-mãe do computador. O tipo mais atual desse tipo de conector é o PCI Express, que apresenta algumas variações relacionadas ao número de vias e velocidades.

Figura 2.16 | Exemplo de barramentos utilizados atualmente



Fonte: adaptada de Monteiro (2007).

Figura 2.17 | Exemplo de barramentos utilizado atualmente



Fonte: adaptada de Monteiro (2007).

Assimile

Os três principais tipos de barramento de um computador são:

- Barramento de dados.
- Barramento de endereços.
- Barramento de controle.

Além deles, existem, atualmente, diferentes tipos de barramentos adotados pelos fabricantes desses dispositivos:

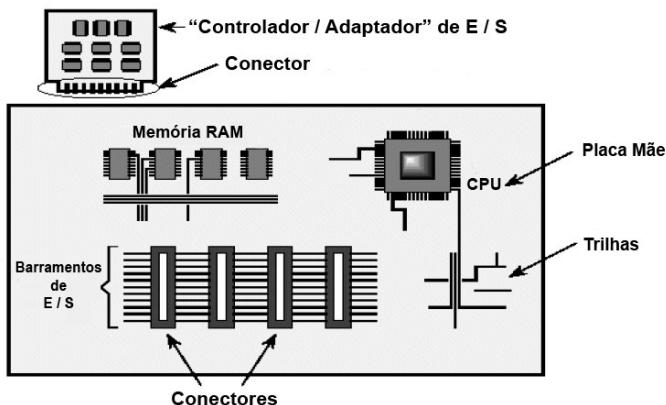
- Barramento local.
- Barramento de sistema.
- Barramento de expansão.

Os periféricos de E/S apresentam diferentes velocidades de transmissão e por isso não se conectam diretamente à CPU do computador. Dessa forma, os dispositivos são conectados à placa-mãe por meio de suas interfaces, normalmente placas que contêm diversos componentes, incluindo o *chipset*, responsáveis pela sincronização entre a velocidade dos dispositivos e a velocidade dos barramentos e da CPU do computador. Para que interfaces de fabricantes diferentes possam funcionar de maneira organizada, esses fabricantes têm procurado uma padronização na definição de protocolos de funcionamento. Assim, vários tipos diferentes de dispositivos podem funcionar adotando determinado padrão. Por exemplo: temos vários fabricantes de teclado, e todos os teclados funcionarão seguindo determinado protocolo, independentemente do modelo (FÁVERO, 2011).

Dessa forma, foram desenvolvidos vários padrões de barramentos para a conexão de placas de interfaces. Considere a interface uma placa adicional contendo um *chipset* e que proporcionará a sincronização dos dispositivos periféricos de E/S. (EQUIPE DIGERATI, 2009).

O nome barramento é usado nesse caso para identificar o tipo de conector de acordo com o número de pinos e números de vias utilizados na comunicação com a placa-mãe. Por esse motivo, o termo barramento é mais conhecido ao se referir a esses padrões de conectores da placa-mãe. Você já percebeu, no entanto, que existem vários tipos de barramento, e que esse termo abrange muito mais conceitos do que somente esse (ALMEIDA, 2007).

Figura 2.18 | Representação de tipos de barramentos de E/S dentro de uma placa-mãe



Fonte: adaptada de Tanenbaum (2007).

Os tipos mais conhecidos de padrões de barramentos de conectores são:

- **ISA (*Industry Standard Adapter*):** um dos primeiros padrões, desenvolvido pela IBM, apresentava uma taxa de transferência muito baixa e não é mais utilizado.
- **PCI (*Peripheral Component Interconnect*):** desenvolvido pela Intel, tornando-se quase um padrão para todo o mercado como barramento de alta velocidade. Permite transferência de dados em 32 ou 64 bits a velocidades de 33 MHz e de 66 MHz. Cada controlador permite cerca de quatro dispositivos.
- **AGP (*Accelerated Graphics Port*):** barramento desenvolvido por vários fabricantes liderados pela Intel, com o objetivo de acelerar as transferências de dados do vídeo para a memória principal

- especialmente dados em 3D, muito utilizados em aplicativos gráficos como programas CAD e jogos.
- **PCI Express (*Peripheral Component Interconnect Express PCIe*):** esse barramento foi construído por um grupo de empresas denominado PCI-SIG (*Peripheral Component Interconnect Special Interest Group*), composto por empresas como a Intel, AMD, IBM, HP e Microsoft. Esse barramento veio para atender às demandas por mais velocidade gerada por novos chips gráficos e tecnologias de rede, apresentando altas taxas de transferência. Assim, o PCI e o AGP foram substituídos pelo PCI Express. Até o momento existiram três versões desse barramento: 1.0 – lançado em 2004; 2.0 – lançado em 2007; e o 3.0 – lançado em 2010. Cada barramento tem um protocolo-padrão, utilizado pela indústria de computadores para a fabricação de todos os dispositivos de entrada e saída a serem conectados nos diferentes tipos de barramento.

Atualmente os barramentos PCIe evoluíram, e temos uma gama de utilização desses barramentos para interligar os mais diversos dispositivos. O Quadro 2.5 ilustra os tipos de PCI Express e suas velocidades. O mais recentemente criado é o PCI Express 4.0, que na versão de 16 vias tem a velocidade de 36 GB/s.

Quadro 2.5 | Barramento PCI Express e suas velocidades

Quantidade de vias	PCIe 4.0	PCIe 3.0
x1	2 GB/s	1 GB/s
x2	4 GB/s	2 GB/s
x4	8 GB/s	4 GB/s
x8	16 GB/s	8 GB/s
x16	32 GB/s	16 GB/s

Fonte: PCI-SIG.

USB (*Universal Serial Bus*): tem a característica particular de permitir a conexão de muitos periféricos simultaneamente ao barramento e por uma única porta (conector), conecta-se à placa-mãe. Grande parte dos dispositivos USB é desenvolvida com a característica de serem conectados ao computador e utilizados logo em seguida, o que é chamado de *plug-and-play*. Existe várias versões do padrão USB: o USB 1.1, USB 2.0 e USB 3.0, e o USB-C, também chamado de USB 4.0. A diferença entre essas versões são as velocidades de transferência de dados. O Quadro 2.6 ilustra os padrões USB, bem como suas velocidades.

Quadro 2.6 | Padrões USB

Versão	Velocidades
USB 1	12 Mbps
USB 2.0	480 Mbps
USB 3.0	5 Gb/s
USB 3.1	10 Gb/s
USB 3.2	20 Gb/s
USB 4	40 Gb/s

Fonte: elaborado pelo autor.

Refletá

Com a evolução dos dispositivos como placas de vídeo, memórias RAM, externas como SSDs e mesmo os HDs, houve a necessidade de desenvolver barramentos com altas velocidades para que seja aproveitada ao máximo a velocidade de transferência desses dispositivos. Porém, se você utilizar um barramento com menor velocidade que a velocidade de leitura e gravação de uma memória, qual seria o resultado?

Além dos barramentos, para que os usuários possam inserir dados no computador e obter as informações nele contidas, são necessários dispositivos/periféricos que permitam a comunicação do usuário com o computador, tanto para dar a entrada de dados e instruções quanto para proporcionar a saída de resultados ao usuário, no formato adequado ao solicitado.

Esses dispositivos/periféricos devem ser capazes de realizar duas funções:

- Receber ou enviar informações ao meio exterior.
- Converter as informações de entrada para a linguagem da máquina e as de saída para a linguagem usada pelo usuário (MONTEIRO, 2007).

Em um computador, há a necessidade de que a CPU se comunique com a memória principal (RAM) e com os dispositivos de E/S para a transferência de dados. Semelhante ao que ocorre com a comunicação entre CPU e memória principal, na qual são definidos endereços para cada posição de memória, referenciados pela CPU, quando se trata de comunicação entre CPU e dispositivos, torna-se necessário que a CPU indique um endereço que corresponda ao periférico em questão.

Diversas formas de comunicação entre CPU e memória principal foram propostas, as quais passaram por melhorias ao longo do tempo, buscando sempre alcançar a melhor utilização da CPU e o melhor desempenho para o

sistema como um todo. Murdocca e Heuring (2001) destacam três métodos para gerenciar a entrada e saída:

Entrada e saída programada

Nesse método, a CPU precisa verificar continuamente se cada um dos dispositivos necessita de atendimento. Esse método não é mais utilizado (MURDOCCA; HEURING, 2001).

Entrada e saída controladas por interrupção

Esse método possibilita que a CPU não fique presa em espera ocupada até que um dispositivo esteja pronto para realizar a transferência de dados propriamente dita. Embora tenha passado por melhorias, esse método não é mais utilizado (MURDOCCA; HEURING, 2001).

Acesso direto à memória (DMA – *Direct Memory Access*)

A função do controlador (ou interface) é controlar seu dispositivo de E/S e manipular para ele o acesso ao barramento. Quando um programa quer dados do disco, por exemplo, ele envia um comando ao controlador de disco, e este controlador emitirá comandos de busca e outras operações necessárias para que ocorra a transferência (TANENBAUM, 2007).

Dessa forma, a CPU solicita a transferência para um dispositivo denominado controlador de acesso direto à memória principal (DMA Controller), o qual se responsabiliza totalmente pela transferência. A CPU é avisada apenas no início e no final da operação de transferência entre dispositivo e memória principal. Esse é o tipo de acesso utilizado atualmente pelas interfaces de E/S (FÁVERO, 2011).

Nesta seção foram apresentados os barramentos e entrada e saída, bem como as interfaces utilizadas para controlar os dispositivos. Neste momento, com o conhecimento adquirido, você já tem as informações para analisar e diversificar as características principais de uma máquina. Agora você já é capaz de solucionar a situação-problema desta seção. Preparado? Boa sorte!

Sem medo de errar

Descrição do Servidor	PowerEdge T140
Fabricante	DELL
Processador	Intel® Xeon® E-2224 3.4GHz, 8M cache, 4C/4T, turbo (71W)
Chipset	Intel C246
Memória RAM	16GB UDIMM DDR4 de 2666 MT/s

HD	1TB SATA cabeado, 6 Gbps, 7200 RPM
Slots	PCIe 1 slot de 3ª geração (x16) 2 slots de 3ª geração (x8) 1 slot de 3ª geração (x1)
Portas de E/S e legadas	Placa de vídeo 1 VGA Portas frontais 1 micro-USB dedicada para iDRAC 1 USB 3.0 Portas traseiras 1 serial 2 USB 3.0 4 USB 2.0 1 VGA Portas internas 1 USB 3.0
Comunicações (Placas possíveis de redes contidas neste servidor)	Placa de rede integrada Broadcom 5720 com duas portas de 1Gb Placa de rede Broadcom 5719 com quatro portas de 1Gb

Fonte: adaptado de Dell (2020, [s.p.]).

Faça valer a pena

1. Em um computador há a necessidade de que a CPU se comunique com a memória principal (RAM) e com os dispositivos de E/S para a transferência de dados.

Sobre os métodos de gerenciamento de entrada e saída de dados, analise a seguir:

- I. Entrada e saída programada.
- II. Entrada e saída controladas por interrupção.
- III. Entrada e saída controladas por processamento.
- IV. Acesso direto à memória.

São métodos para gerenciar a entrada e saída de dados apenas:

- a. I e II.
- b. II e III.
- c. I, II e III.

- d. I, II e IV.
 - e. I, II, III e IV.
- 2.** Os barramentos são um conjunto de vias que realizam a comunicação entre os dispositivos, tais como a UCP, memória, discos rígidos e dispositivos de entrada e saída, entre outros.
- O barramento que realiza a comunicação entre a UCP e a memória e para a transferência das informações que serão processadas é o:
- a. Barramento de endereço.
 - b. Barramento de dados.
 - c. Barramento de expansão.
 - d. Barramento de sistema.
 - e. Barramento local.
- 3.** Existem muitas particularidades entre os diversos periféricos de E/S, por exemplo: a velocidade de transferência de um teclado ou de um mouse é muito menor do que a velocidade de um SSD. Nesse caso, foram criados vários tipos de barramentos com diferentes velocidades de transferência. Atualmente, existem diversos tipos de barramentos para suprir essa necessidade.

Sobre os barramentos, analise as asserções a seguir e a relação proposta entre elas.

- I. A comunicação entre os componentes de um processador é realizada via barramentos, os quais possibilitam a movimentação de dados entre a UCP e a memória e interligação entre outros dispositivos, placas de vídeo e HDs.

PORQUE

- II. Os barramentos têm a capacidade de realizar a leitura e a gravação dos arquivos das memórias para a UCP com a mesma velocidade dos dispositivos.

A respeito dessas asserções, assinale a alternativa correta.

- a. A asserção I é uma proposição verdadeira e a II, falsa.
- b. A asserção I é uma proposição falsa e a II, verdadeira.
- c. As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- d. As asserções I e II são proposições verdadeiras e a II justifica a I.
- e. As asserções I e II são proposições falsas.

Referências

- ADVANCED MICRO DEVICES INC. **Processador AMD Ryzen™ Threadripper™ 3990X.** AMD, 2020, [s.p.]. Disponível em: <https://www.amd.com/pt/products/cpu/amd-ryzen-threadripper-3990x>. Acesso em: 24 mar. 2020.
- ALECRIM, E. **O que é SSD (SOLID State Drive).** Infowester, 19 maio 2019. Disponível em: <http://www.infowester.com/ssd.php>. Acesso em: 27 mar. 2020.
- ALECRIM, E. **Processadores:** clock, bits, memória cache e múltiplos núcleos. Infowester, 26 jul. 2012. Disponível em: <http://www.infowester.com/processadores.php#cache>. Acesso em: 27 mar. 2020.
- ALECRIM, E. **Tecnologia SCSI (Small Computer Systems Interface).** Infowester, 12 maio 2012. Disponível em: <http://www.infowester.com/scsi.php>. Acesso em: 27 mar. 2020.
- ALMEIDA, M. **Curso de montagem e manutenção de micros.** São Paulo: Digerati Books, 2007.
- BROOKSHEAR, J. G. **Ciência da computação:** uma visão abrangente. 11. ed. São Paulo: Bookman, 2013.
- CARMONA, T. **Universidade VBA.** São Paulo: Digerati Books, 2006.
- DELL INC. **Servidor em torre PowerEdge T140.** Eldorado do Sul, 2020. Disponível em: <https://www.dell.com/pt-br/work/shop/productdetailstxn/poweredge-t140>. Acesso em: 25 mar. 2020.
- EQUIPE DIGERATI BOOKS. **Guia técnico de montagem e manutenção de computadores.** São Paulo: Digerati Books, 2009.
- FÁVERO, E. M. B. **Organização e arquitetura de computadores.** Pato Branco: Universidade Tecnológica Federal do Paraná, 2011.
- FONSECA FILHO, C. **História da computação:** o caminho do pensamento e da tecnologia. [recurso eletrônico]. Porto Alegre: PUCRS, 2007.
- GERALDI, L. M. A. **Elucidando os sistemas operacionais:** um estudo sobre seus conceitos. Taquaritinga: Clube de Autores, 2013.
- HARDWARE 2. Site Clube do. **Dicionário de termos.** Disponível em: <http://www.clubedohardware.com.br/dicionario/termo/rom/239>. Acesso em: 5 dez. 2015.
- HAUTSCH, O. **O que é pen drive?** TecMundo, 30 out. 2008. Disponível em: <http://www.tecmundo.com.br/pendrive/844-o-que-e-pendrive-.htm>. Acesso em: 27 mar. 2020.
- HENESSY, J. L.; PETTERSON, D. A. **Arquitetura de computadores:** uma abordagem quantitativa. 5. ed. Rio de Janeiro: Elsevier, 2014.
- JORDÃO, F. **DDR4:** será que as novas memórias vão fazer diferença no desempenho do PC? TecMundo, 6 mar. 2015. Disponível em: <http://www.tecmundo.com.br/memoria-ram/76135-ddr4-novas-memorias-fazer-diferenca-desempenho-pc.htm>. Acesso em: 27 mar. 2020.

- KINGSTON TECHNOLOGY CORPORATION. **DataTraveler Ultimate GT**. Califórnia, 2017. Disponível em: https://www.kingston.com/datasheets/dtugt_br.pdf. Acesso em: 25 mar. 2020.
- MEIRELLES, A. **Hardware, o guia definitivo**. Hardware.com.br., 1 out. 2007. Disponível em: <http://www.hardware.com.br/livros/hardware/processador.html>. Acesso em: 27 mar. 2020.
- MONTEIRO, M. A. **Introdução à Organização de Computadores**. 5. ed. Rio de Janeiro: LTC, 2007.
- MURDOCCA, M. J.; HEURING, V. P. **Introdução à arquitetura de computadores**. Rio de Janeiro: Elsevier, 2001.
- OKUYAMA, F. Y. **Desenvolvimento de Software**: conceitos básicos. [recurso eletrônico]. Porto Alegre: Bookman, 2014.
- OLIVEIRA, R. A. **Informática**. Rio de Janeiro: Elsevier, 2007.
- PATTERSON, D. A.; HENNESSY, J. L. **Arquitetura de computadores**: uma abordagem quantitativa. Rio de Janeiro: Elsevier, 2014.
- PATTERSON, D. A.; HENNESSY, J. L. **Organização e projeto de computadores**: a interface hardware/software. São Paulo: Campus, 2005.
- PEREIRA, A. P. **Quais as diferenças entre IDE, SATA e SATA II?** TecMundo, 12 ago. 2009. Disponível em: <https://www.tecmundo.com.br/placa-mae/2580-quais-as-diferencias-entre-ide-sata-e-sata-ii-.htm>. Acesso em: 27 mar. 2020.
- RAINER, K. **Introdução a sistemas de informação**. Tradução Multinet Produtos. [recurso eletrônico]. 3. ed. Rio de Janeiro: Elsevier, 2012.
- SEAGATE TECHNOLOGY. **Barracuda Pro Compute**. Folha de especificações do HDD de 3,5 polegadas. Califórnia, 2018. Disponível em: https://www.seagate.com/www-content/datasheets/pdfs/barracuda-pro-14-tb-DS1901-9-1810BR-pt_BR.pdf. Acesso em: 24 mar. 2020.
- SOUZA FILHO, G.; SANTANA, E.; MEDEIROS, A. **Introdução à computação**. 2. ed. João Pessoa: UFPB, 2014.
- TANEMBAUM, A. S. **Organização estruturada de computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013.
- VELLOSO, F. C. **Informática**: conceitos básicos. 9. ed. Rio de Janeiro: Elsevier, 2011.
- WESTERN DIGITAL TECHNOLOGIES INC. **SSD Portátil SanDisk Extreme®**. Califórnia, [s.d.]. Disponível em: https://documents.westerndigital.com/content/dam/doc-library/pt_br/assets/public/sandisk/product/portable-drives/extreme-pro-usb-3-1-ssd/data-sheet-extreme-pro-usb-3-1-ssd.pdf. Acesso em: 24 mar. 2020.
- ZAMBARDA, P. **Internet das Coisas**: entenda o conceito e o que muda com a tecnologia. TechTudo, 16 ago. 2014. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html>. Acesso em: 27 mar. 2020.

Unidade 3

Maurício Acconcia Dias

Bases numéricas, representação dos dados e instruções de máquinas

Convite ao estudo

O processo de desenvolvimento de um processador inicia-se com o projeto de seu conjunto de instruções, ou seja, um conjunto de operações que podem ser lidas e executadas por um determinado *hardware*. Estas instruções são o coração do processador e são compostas por números, sendo necessário, portanto, um conhecimento de sistemas numéricos e bases numéricas para que seja possível entender e projetar um processador. O conteúdo apresentado nesta unidade de ensino é a parte que faltava do conteúdo que permite entender por completo o funcionamento de um processador e, também, o seu projeto.

Todo sistema de representação numérico possui uma base associada. Por exemplo, o sistema decimal, como o próprio nome diz, utiliza a base 10 (escolhida devido à facilidade de se contar até 10 com os dedos das mãos). Já os computadores precisam de um sistema que esteja relacionado à sua tecnologia de fabricação, ou seja, os sistemas eletrônicos. Em sistemas eletrônicos é relativamente fácil chavear uma corrente, ou seja, permitir ou não a passagem de elétrons por um determinado fio. Neste caso, conclui-se que o sistema mais interessante para os computadores é o sistema binário que, como o próprio nome diz, possui base 2, ou seja, apenas dois algarismos: o ‘0’ que representa a falta de corrente e o ‘1’ que representa a presença de corrente.

Dentre os principais sistemas numéricos utilizados em sistemas computacionais, podemos citar os números binários, octais e hexadecimais. O crescente desenvolvimento do *hardware* resultou em um aumento no número de *bits* trabalhados pelo processador e, desta forma, os números hexadecimais estão ainda mais importantes atualmente para o cenário do desenvolvimento de *hardware*.

Assim, esta unidade tem como principal objetivo conhecer e compreender os conceitos e desenvolvimentos históricos de arquitetura e organização de computadores, assim como a estrutura de computadores. Para isso, serão apresentados todos os conceitos relacionados à representação binária de informação, a composição de outras representações relacionadas

ao sistema binário e a classificação de números binários. Após apresentar os números binários, também serão abordados os números octais e hexadecimais, bem como as conversões de base entre todos estes diferentes sistemas de numeração (Seção 3.1). Todo este conhecimento será necessário para que seja possível projetar instruções de máquinas, que nada mais são do que um conjunto de números binários que será o coração do processador que iremos projetar mais adiante (Seções 3.2 e 3.3) e as linguagens de máquina e montagem (Assembly) para projetar as instruções (Seção 3.4). Curioso para saber como? Vamos descobrir juntos!

Seção 1

Sistemas numéricos: conceitos, simbologia e representação de base numérica

Diálogo aberto

Os sistemas de numeração são importantes em diversos processos no projeto de processadores. Dominar estes sistemas e saber aplicá-los corretamente influencia diretamente no desempenho do produto final. Mesmo assim, em diversos casos, esta questão é ignorada e vários projetos precisam ser refeitos justamente porque a representação escolhida não era suficiente para englobar todos os aspectos necessários para a realização do projeto do processador. Um exemplo seria uma representação que não apresenta suporte a números negativos em situações em que a maioria dos cálculos realizados envolve o conjunto de números inteiros.

Seria então possível aplicar o conhecimento abordado em sistemas numéricos para corrigir ou até mesmo verificar se uma arquitetura realmente está de acordo com os requisitos do projeto? Seria necessário ter algum profissional que dominasse bem a parte de conversões numéricas em uma equipe de desenvolvimento de *hardware*? Apesar de ser um conhecimento simples, ele não deve e não pode ser ignorado em diversas situações.

Considere que você faz parte de um time de desenvolvimento de *hardware* de uma empresa X que necessita de uma verificação em sua arquitetura. Após adquirir o conhecimento desta seção, você é a pessoa indicada para realizar os testes necessários. A empresa vai apresentar um conjunto de números binários trabalhados pelo processador e você terá de apresentar um parecer de número de acertos e erros de forma que o projeto seja avaliado e as decisões de correção possam ser devidamente tomadas. Nesse sentido, você deverá avaliar os números apresentados na tabela apresentada a seguir:

Tabela 3.1 | Conversão de números binários e validação

Resultado esperado binário	Resultado obtido octal	Resultado obtido hexadecimal	Validação
0001 1110		1E	
1101 0111	315		
1111 1101		FF	
1010 0101		A5	
0111 1111	154		

1100 0011	303		
1111 1111		AA	

Fonte: elaborada pelo autor.

Você, então, deverá fazer a conversão, preencher as linhas em branco e, depois, preencher a coluna de *validação*, com o valor CORRETO ou ERRADO.

Aproveite, portanto, o conteúdo da seção e prepare-se para seu primeiro desafio com sistemas numéricos! Boa sorte!

Não pode faltar

Neste momento inicia-se o estudo sobre sistemas numéricos, base para o completo entendimento de arquitetura de computadores. Você deve observar que este conteúdo é teórico e inicial, portanto, mesmo que pareça repetitivo, lembre-se que solidificar o aprendizado deste conteúdo neste momento será crucial para o sucesso no projeto de seu processador!

Os sistemas de numeração são um conjunto de algarismos e regras que, quando aplicadas, permitem que um conhecimento quantitativo seja representado e possa ser devidamente trabalhado. Este material apresenta quatro sistemas, sendo três deles diretamente relacionados com o conteúdo de arquitetura de computadores: o sistema decimal, o sistema binário, o sistema octal e o sistema hexadecimal. O nome de cada sistema remete à base utilizada para a representação numérica: o sistema decimal utiliza base dez (10), o sistema binário utiliza base dois (2), o sistema octal utiliza base oito (8) e o sistema hexadecimal utiliza a base dezesseis (16).

O sistema mais utilizado pelo ser humano atualmente é o sistema decimal, já que é intuitivo considerando o ato de contar nos dedos (IFRAH, 1997). Porém, ao utilizarmos o sistema decimal diversas operações são realizadas pelo nosso cérebro sem que o pensamento seja direcionado diretamente para elas. Pense na seguinte situação, ao considerar o número 567 (quinhentos e sessenta e sete) nosso cérebro subdivide este número até mesmo para que seja possível falar seu “nome”. E como ele faz isso?

O número 567 possui três casas decimais antes da vírgula certo? Portanto este número possui unidades, dezenas e centenas. Você já parou para analisar o motivo de falarmos desta maneira? Vamos conferir então: inicialmente, é preciso observar que um número pode ser representado por potências de

sua base. No caso dos números decimais, a base é 10, portanto, o número apresentado pode ser representado como na Equação 3.1.

$$567 = 5 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0$$

3.1

A Equação 3.1 apresenta uma soma de potências de 10 (que é a base do número decimal) multiplicadas pelos algarismos do número em questão. A soma irá resultar em $500 + 60 + 7$ que é o número que estamos considerando. Apesar de não parecer, inicialmente seu cérebro faz isso sem que você perceba. Agora que você percebe, é possível entender que uma representação numérica também é influenciada por seus algarismos. No caso dos números decimais temos 10 algarismos, do 1 ao 9 e o 0. Sendo assim, é possível definir outra regra: os algarismos utilizados para representação numérica são compostos segundo a Equação 3.2, onde b seria a base do sistema numérico em questão e Q o conjunto dos algarismos.

$$Q = 0 \rightarrow (b - 1)$$

3.2

Vamos trabalhar estes conceitos também com exemplos das outras bases a serem estudadas nesta seção. Lembre-se que, conforme comentado no início da seção, vamos estudar os números binários, octais e hexadecimais. As bases destes números são respectivamente 2, 8 e 16. Vamos compor, então, o conjunto de algarismos necessários para cada base.

Considerando $b = 2$, temos o conjunto $Q = \{0, 1\}$

Considerando $b = 8$, temos o conjunto $Q = \{0, 1, 2, 3, 4, 5, 6, 7\}$

O problema é encontrado, no nosso caso, para $b = 16$. Conhecemos apenas 10 algarismos então é necessário incorporar outros símbolos. Neste caso o conjunto de algarismos seria $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Agora que definimos alguns conceitos básicos sobre sistemas numéricos é possível apresentar os sistemas desta seção junto dos conceitos de conversão de bases. Um conceito importante é a representação destes números agora que iremos tratar de várias bases diferentes. Como pôde ser constatado no quadro, os conjuntos de algarismos possuem elementos em comum. O conjunto dos algarismos dos números binários $Q = \{0, 1\}$ está contido nos conjuntos de algarismos de quaisquer outras bases. Portanto, como é possível diferenciar o número 10 na base binária do número 10 na base decimal? Para efeito de representação, neste material, iremos adotar o padrão de colocar a base do número em seu canto inferior direto: número 10 na base binária será representado como 10_2 e o mesmo número 10 na base octal será representado por 10_8 .

A conversão de bases é o que permite que seja possível interpretar um número de determinada base em outra base. Vamos iniciar as conversões das

outras bases para a base decimal. Por exemplo, quanto vale o número 10_2 , na base decimal? O resultado desta conversão é facilmente obtido com uma soma das potências da base em questão como demonstrado na Equação 3.3.

$$10_2 = 1 \cdot 2^1 + 0 \cdot 2^0 = 2 + 0 = 2_{10} \quad 3.3$$

Portanto, para que se converta qualquer número para a base 10 é necessário que se decomponha os algarismos dos números multiplicando suas respectivas potências e some-se o resultado. Fazendo o mesmo exemplo anterior, porém para o número 10_{16} , temos o resultado apresentado na Equação 3.4:

$$10_{16} = 1 \cdot 16^1 + 0 \cdot 16^0 = 16 + 0 = 16_{10} \quad 3.4$$

Para entender como funcionam as letras (de A a F) nos números hexadecimais é necessário convertê-las para a base decimal. O número seguido de 9 seria o 10, como não temos o algarismo para 10, utiliza-se a letra A. Portanto A vale 10, B vale 11, C vale 12, e assim por diante até que F vale 15. Sendo assim o número FF_{16} convertido para a base decimal é apresentado na Equação 3.5:

$$FF_{16} = F \cdot 16^1 + F \cdot 16^0 = 15 \cdot 16 + 15 \cdot 1 = 255 \quad 3.5$$

A conversão inversa, ou seja, de um número da base 10 para qualquer outra base é feita realizando-se sucessivas divisões do número em questão pela base desejada e considerando seus restos. Vamos converter o número 47_{10} para o sistema binário. Iniciamos a conversão dividindo o número pela base, que no caso é 2, e verificando o quociente e o resto. Portanto:

- $47 \div 2 = 23$, temos então o quociente 23 e o resto 1. Como o quociente ainda é divisível pela base continuamos o processo.
- $23 \div 2 = 11$ com resto 1
- $11 \div 2 = 5$ com resto 1
- $5 \div 2 = 2$ com resto 1
- $2 \div 2 = 1$ e o resto é 0.
- O último quociente, que é 1, não é divisível pela base sendo que deve ser considerado o último número.

Analizando apenas os restos das divisões, e o último algarismo que não é divisível pela base, temos 111101. Porém este método nos apresenta o número de forma invertida, portanto o resultado correto é 10111_2 . Fazendo o mesmo exemplo, porém convertendo 47_{10} para a base hexadecimal temos:

- $47 \div 16 = 2$ sendo que o resto neste caso é F (representando 15)

- 2 já não é divisível pela base 16, sendo, portanto, o último número considerado.

Adotando, então, a regra utilizada anteriormente, o resultado seria F2 que está invertido, sendo corrigido para $2F_{16}$.

Após uma análise cuidadosa dos procedimentos de conversão adotados até o momento é possível notar a dificuldade da conversão entre bases que não envolvam a base 10. Para resolver este problema sempre **iremos converter qualquer número primeiro para a base 10 e, em seguida, convertemos para a base desejada.**

Assimile

Devido à importância das etapas de conversão, vamos explorar mais a fundo a afirmação anterior. Foi apresentado que caso seja necessário converter um número entre bases é sempre recomendável que o número a ser convertido seja inicialmente passado à base 10 e, em seguida, convertido para a base desejada.

Este procedimento visa facilitar a conversão entre bases de modo que independentemente da situação a conversão seja uma operação simples para nós que temos como base de aprendizado o sistema decimal. Portanto, lembre-se na conversão de um número N da base A para a base B:

- Converter N_A para a base 10 gerando C_{10}
- Converter C_{10} para R_B .

Aplicando o conceito apresentado vamos converter o número 541_{16} para a base 8, passando a utilizar o sistema octal. Sendo assim temos $A=16$, $B=8$ e $N=541$. O passo a passo das operações não será detalhado aqui, porém lembre-se de considerar os restos das divisões e de inverter o número no final. A primeira conversão resulta no número 1345_{10} . O resultado da segunda conversão é 2501_8 , conferido na Equação 3.6.

$$2501_8 = 2 \cdot 8^3 + 5 \cdot 8^2 + 0 \cdot 8^1 + 1 \cdot 8^0 = 1024 + 320 + 0 + 1 = 1345_{10} \quad 3.6$$

Agora que sabemos converter números entre as principais bases numéricas utilizadas em aplicações da computação vamos voltar nosso foco para as bases binária e hexadecimal apresentando algumas de suas características e relações.

A primeira característica a ser abordada é a forma como chamamos os algarismos de um número binário. Cada um dos algarismos em um número binário é chamado de ***BIT***. Um grupo de 8 *bits* representa um ***BYTE***. Por fim, um conjunto de *bits* que representa algo é chamado de **PALAVRA**.

Assimile

Vamos considerar alguns casos para que seja possível entender melhor como esta nomenclatura de números binários influí na maneira como obtemos informações.

Iniciando com o conceito de palavra, é muito comum atualmente falar que um determinado processador trabalha em 32 ou 64 bits. Quando fazemos esta classificação estamos falando sobre o tamanho da PALAVRA que determinado processador é capaz de interpretar.

Isso quer dizer que um processador com uma palavra de 32 bits só entende 32 algarismos binários de uma vez, e um processador com uma palavra de 64 bits só entende 64 algarismos binários de uma vez. Por isso que, para executar um programa feito em uma arquitetura de 32 bits em um processador de 64 bits, é necessário algum tipo de compatibilidade (retornaremos neste assunto mais a diante).

Quando vamos comprar um HD novo, notamos que as unidades estão expressas em GB ou TB. Cuidado! Quando o B é maiúsculo está indicando *bytes*, quando é minúsculo está indicando *bits*. A sigla GB, por exemplo, se refere a *gigabytes*, e a sigla mbps (normalmente utilizada para medir conexões de internet) se refere a *megabits* por segundo.

Vamos analisar então um caso que irá exemplificar a diferença de nomenclatura. Um HD de 1TB possui, então, 1 ***terabyte*** de capacidade total. Para descobrir quantos *bits* disponíveis este HD possui é necessário realizar alguns cálculos:

- 1 ***terabyte*** é equivalente a:
 - 1,024 ***gigabytes***, que são equivalentes a:
 - 1,048,576 ***megabytes*** que são equivalentes a:
 - 1,073,741,824 ***bytes***. Uma vez que 1 byte possui 8 bits, temos (multiplicando o valor diretamente por 8):
 - 8,589,934,592 ***bits***. Ou seja, mais de 8 bilhões de bits para armazenamento.

Repare que as conversões realizadas entre *bytes* foram feitas multiplicando-se por 1024 e não por 1000. Cuidado, este é um erro comum pois estamos acostumados com as grandezas de mil, milhões, bilhões, que são feitas multiplicando-se por potências de 10. Como estamos tratando do sistema binário é necessário tratar de potências de 2 e não de 10. Pense da seguinte maneira: para aumentar de grandeza um número multiplicamos por 1000 (de 5 mil para 5 milhões é só multiplicar por 1000 correto?). Então, na base 2 não temos como multiplicar por 1000 já que 1000 não é potência de 2. Então qual o primeiro número potência de 2 próximo ao 1000? A resposta é 1024, que seria 2^{10} .

A exceção de conversão ocorre no caso dos *bits* que são a unidade básica. Portanto, quando adquirimos um pacote de internet de 10 mbps, seriam 10 **megabits** por segundo que são convertidos diretamente para 10,000,000 *bits*.

A segunda característica é a forma como fazemos a leitura do número binário. Os números decimais são lidos da esquerda para direita, ou seja, da mesma forma como é realizada a leitura textual. A interpretação de números decimais é feita desta maneira pois temos um padrão para leitura que foi aceito e adotado.

Assim como no sistema decimal, é necessário que se adote uma forma de ler um número binário para que o valor considerado não seja diferente do esperado. Veja o exemplo do número 1011₂. Da esquerda para a direita o número é lido como 11₁₀, porém da direita para a esquerda o número é lido como 13₁₀ (Dica: exercite a conversão de base conferindo os resultados da conversão do número binário apresentado para a base decimal). Para evitar este problema foram criadas nomenclaturas para indicar como o número deve ser lido e evitar o problema de interpretação.

Uma definição de arquitetura de um processador deve vir acompanhada, obrigatoriamente, da indicação de como o número binário é considerado, sendo que existem duas possibilidades. Para ler o número binário da esquerda para a direita (maneira como lemos um texto) dizemos que o MSB é o primeiro e o LSB é o último. **MSB** é uma sigla para o *bit* mais significativo (do inglês, *Most Significant Bit*), e **LSB** é o *bit* menos significativo (do inglês, *Less Significant Bit*). Se o primeiro é o MSB, ele é o que representa o maior valor, sendo que o número deve ser lido da esquerda para direita. Entretanto, existem arquiteturas que o primeiro bit é o LSB e o último é o MSB, fazendo com que o número seja interpretado de maneira invertida.

Exemplificando

Vamos considerar que duas empresas, A e B, possuem duas arquiteturas cujo tamanho da palavra é o mesmo, 12 bits, porém cada uma delas possui um tipo de arquitetura diferente. Na arquitetura da empresa A, o MSB é o primeiro e na arquitetura da empresa B, ele é o último *bit*. No caso do número 10011101_2 ser parte de alguma operação matemática, qual o seu valor para cada uma das arquiteturas?

Na arquitetura da empresa A, o número deve ser convertido da forma que resulte em 157_{10} , porém na arquitetura da empresa B, o número deve ser convertido como 10111001_2 resultando em 185_{10} .

Encerrando o conteúdo desta seção, vamos fazer uma análise de como os números hexadecimais podem nos ajudar a escrever números binários maiores. A base dos números hexadecimais é 16. Para representar um número decimal N_d em base binária precisaremos de um número mínimo de bits N_{\min} . Para sabermos qual o número utilizar, vamos à Equação 3.7:

$$N_{\min} = \lceil \log_2 N_d \rceil \quad 3.7$$

No caso de cada algarismo hexadecimal, o valor possível é de 0 a $F(15)$. Para representar cada número hexadecimal em binário precisamos utilizar a Equação 3.7 com $N_d = 15$. O resultado será 4 porque é o teto do número 3,90 que é o logaritmo de 15 na base 2. Sabendo deste fato, é possível concluir que cada algarismo hexadecimal representa 4 bits. Então, como fazemos esta representação? Vejamos um exemplo:

- O número binário 100111010001111 possui 4 conjuntos de 4 *bits* sendo;
- $1001\ 1101\ 1000\ 1111$ cada um desses grupos possui um correspondente decimal;
- Seriam respectivamente 9;14 (E); 8; e 15 (F), convertendo-os para hexadecimal temos;
- $9E8F$ que é exatamente o mesmo número que 100111010001111 .

Desta forma fica muito mais simples escrever número binários complexos.

Refita

Agora que você já possui o conhecimento sobre as bases binária, octal e hexadecimal seria interessante pensar: qual seria uma aplicação direta para as bases octal e hexadecimal? Seriam as bases octal e hexadecimal uma representação de algum outro tipo de arquitetura? Ou podem ser utilizadas diretamente em alguma situação? Caso não sejam, alguma outra base é indicada? Uma dica, pesquise um pouco sobre computadores ópticos!

Encerramos aqui a seção sobre sistemas numéricos. Foram apresentados os conteúdos básicos para que o aluno possa identificar corretamente os números, fazer conversões de base e resolver problemas mais complexos que utilizam esta teoria como base. Gostou? Ótimo, então vamos para o próximo passo que é o início do projeto do processador! Boa sorte!

Sem medo de errar

A seção apresentada demonstrou os aspectos importantes de sistemas de numeração e conversão de bases. Agora é possível retomar a questão da situação-problema e resolver a questão da verificação dos resultados do processador da empresa X.

Ao ser informado do trabalho a ser realizado, você foi apresentado a uma tabela que continha em cada linha o resultado de uma operação feita pelo processador e o resultado que era esperado. Como são diversos testes realizados e cada um por uma pessoa diferente cada resultado estava em uma base diferente dificultando seu trabalho. A tabela apresentada, como informado no início desta unidade, foi a seguinte:

Tabela 3.1 | Conversão de números binários e validação

Resultado esperado binário	Resultado obtido octal	Resultado obtido hexadecimal	Validação
0001 1110		1E	
1101 0111	315		
1111 1101		FF	
1010 0101		A5	
0111 1111	154		
1100 0011	303		
1111 1111		AA	

Fonte: elaborada pelo autor.

Para fazer a validação dos números é necessário fazer a conversão entre as bases (da mesma maneira que foi apresentado na seção) e encontrar as lacunas que faltam, conferindo os números existentes. Caso o resultado obtido no processador seja diferente do resultado esperado, a questão da representação numérica está errada. O resultado é apresentado na tabela seguinte.

Tabela 3.2 | Resultado da conversão de números binários e validação

Binários	Conversão decimal	Resultado obtido octal	Resultado obtido hexadecimal	Validação
0001 1110	30	36	1E	Correto
1101 0111	215	315 (deveria ser 327)	CD (deveria ser D7)	Errado
1111 1101	253	377 (deveria ser 375)	FF (deveria ser FD)	Errado
1010 0101	165	245	A5	Correto
0111 1111	127	154 (deveria ser 177)	F7 (deveria ser 7F)	Errado
1100 0011	195	303	C3	Correto
1111 1111	255	358 (deveria ser 377)	AA (deveria ser FF)	Errado

Fonte: elaborada pelo autor.

Se você converteu os números corretamente entre as bases, o resultado encontrado está apresentado nesta tabela. É possível ver que existem mais resultados errados do que corretos. Portanto, o seu parecer com relação a estes resultados é que o processador não está projetado de acordo com as bases numéricas necessárias e está apresentando erros na saída de dados. Agora é com a equipe de projeto que deverá solucionar o problema.

Avançando na prática

Proposta de algoritmo de solução para problema de conversão de bases

Você foi contratado por uma empresa que tinha problemas nos projetos de *hardware* pois não havia nenhum *software* para ser utilizado na conversão de bases entre binário e hexadecimal, sendo que os próprios projetistas de *hardware* faziam as conversões e esta situação estava causando erros nos testes. Portanto, seu trabalho é propor um algoritmo (passo a passo) para a conversão de um número binário em um número hexadecimal. Este algoritmo será enviado à equipe de desenvolvimento da empresa para a criação de um *software*.

Resolução da situação-problema

Este algoritmo tem como base os exemplos que fizemos na parte teórica do material. Vamos colocar o procedimento em forma de linguagem natural para facilitar o entendimento da equipe de desenvolvimento de *software*:

1. Obter o número desejado na conversão.
2. Separar o número binário em grupos de 4 *bits* para facilitar a conversão.
3. Converter separadamente cada um dos 4 *bits* em um número hexadecimal.
4. Agrupar o resultado formando um número hexadecimal completo.

Agora que você resolveu o problema, teste seu conhecimento com este desafio! Converta, utilizando o algoritmo projeto, o seguinte número binário:

101001000101110111111011100010010011

Boa sorte!

Faça valer a pena

1. Os HDs são medidos de acordo com seus números de *bytes* e não de *bits*, e a taxa de transmissão em *bits* por segundo. Por exemplo, um HD de 128 GB que possui transferência de 3mbps possui as seguintes características (lembrando que 1 *byte* = 8 *bits*, 1Kb = 1024 *bytes*, 1 MB = 1024 Kbytes, ou seja, ao invés de multiplicar por 1000 multiplica-se por 1024):

- 128 GB = $(128 * 1024) \rightarrow 131.072 \text{ Kbytes} = (131.072 * 1024) \rightarrow 134.217.728 \text{ bytes}$.
- Transmite 3000 bits em um segundo.

Seu computador estava com problemas e quando você verificou seu HD havia parado de funcionar. Ao buscar alternativas encontrou um HD de 480 GB com taxa de transmissão de 4mbps. Você acredita ser uma boa opção, porém seria interessante aplicar seus conhecimentos para verificar como fica a transferência de *bits*. Escolha a opção que se aproxima mais, respectivamente, do valor de *bytes* que existe em seu HD e de quantos *bytes* ele transmite em 27 segundos.

- a. $503,3 \cdot 10^6 \text{ bytes}$ e 13500 bytes .
- b. $503,4 \cdot 10^6 \text{ bytes}$ e 13500 bytes .
- c. $503,3 \cdot 10^6 \text{ bytes}$ e 108000 bytes .

- d. $503,4 \cdot 10^6$ bytes e 108000 bytes.
 - e. $503,4 \cdot 10^3$ bytes e 13500 bytes.
- 2.** Sabe-se que independentemente das bases numéricas existentes, a operação mais simples que se pode realizar é a soma. A única diferença é o momento que ocorre o “vai um”. No caso dos decimais o “vai um” ocorre quando a soma ultrapassa 9, no caso do binário quando ultrapassa 1, no caso do octal quando ultrapassa 7 e no caso do hexadecimal quando ultrapassa 15. É facilmente perceptível o padrão neste contexto.

Considere o número hexadecimal 4F3B7. Qual seria o número hexadecimal somado a ele para que na conversão numérica para binário todos os algarismos do número resultante sejam 1?

Dica

No caso desta questão, o número hexadecimal possui 5 dígitos, sendo assim 5 dígitos multiplicados por 4 bits seriam 20 bits no total. O número de 20 bits todos em 1, ao ser convertido para hexadecimal, seria o número FFFF.

- a. C1D59.
- b. B0C48.
- c. ABCDE.
- d. 00011.
- e. 84C0B.

3. Um determinado processador foi projetado para executar aplicações gráficas. Neste caso, os projetistas, após muitos estudos, concluíram que o número de instruções possíveis para o processador seria 498. Além disso, sabe-se que um dos requisitos do projeto é liberar 9 *bits* da instrução para os sinais de controle.

Considerando o cenário apresentado, julgue as asserções a seguir:

- I. Serão necessários, no mínimo, 17 bits para tudo que é solicitado pelos projetistas

PORQUE

- II. Para representar o número 498 são necessários 8 *bits* que, somados aos outros 9 previamente solicitados, resultam em 17.

A respeito dessas asserções, assinale a alternativa correta.

- a. A asserção I é uma proposição falsa e a II, verdadeira.
- b. As asserções I e II são proposições falsas.
- c. A asserção I é uma proposição verdadeira e a II, falsa.
- d. As asserções I e II são proposições verdadeiras, mas II não justifica a I.
- e. As asserções I e II são proposições verdadeiras e a II justifica a I.

Seção 2

Introdução a instruções de máquinas

Diálogo aberto

Considere que você, profissional de computação, foi contratado para desenvolver um novo núcleo para um processador e precisa agora resolver diversos problemas relacionados à arquitetura do processador. Apesar de o processador já existir, é necessário que você, como projetista, crie todo um conjunto novo de instruções que será utilizado como base para a construção do processador. Você, sabiamente, aceita o desafio, porém ao encará-lo detalhadamente nota que alguns questionamentos são pertinentes e até mesmo críticos para o bom desenvolvimento de seu projeto. Vamos, então, propor alguns questionamentos que possivelmente podem ter surgido.

Qual seria o tamanho desta instrução? Esta questão é complicada já que uma instrução muito grande pode precisar de muito espaço tanto para ser armazenada, quanto para ser transmitida. O que é necessário representar nesta instrução? A instrução tem o papel de agrupar *bits* de modo que contemplem toda a informação necessária para que o processador seja capaz de processá-la. Como apresentado anteriormente, uma instrução é nada mais que um conjunto de algarismos binários, sendo assim, o que cada um destes algarismos representa? Qual parte da informação deve colocada em cada parte dos *bits*? Existe uma regra? Como projetar esta instrução ocupando poucos algarismos e representando tudo que for necessário?

Após a reflexão, você, como um hábil projetista e profissional, relembra que a melhor forma de resolver um problema complexo é quebrando-o em partes menores e solucionando os pequenos problemas (técnica chamada de dividir para conquistar). Sendo assim, o primeiro subproblema a ser resolvido por você será: qual o tamanho e qual será a forma da instrução escolhidos para o projeto do processador?

Além disso, a instrução referencia espaços na memória que serão utilizados para recuperar informações para o processador e, também, para salvar os resultados de processamento. Estas informações deverão também ser incluídas no projeto da instrução, porém, como fazer isso da melhor forma possível?

Utilize os conhecimentos que serão abordados a seguir para entender melhor o problema e sua complexidade e ser capaz de resolvê-lo da melhor forma possível. Boa sorte!

Não pode faltar

Esta seção é uma das mais importantes tanto da unidade quanto de toda a disciplina da arquitetura de computadores. Para entender e poder realizar o projeto de um processador é muito importante que saibamos o conceito de instrução. O conceito que vamos trabalhar não é diferente de um conjunto de instruções para qualquer finalidade, como o passo a passo para fazer um bolo, que deve ser executado em sequência. Porém, a análise desta seção será específica, já que buscamos aprender como o processador entende cada um dos passos de uma sequência. Vamos lá?

Porém, antes de falar de instruções propriamente ditas, será necessário apresentar alguns conceitos que irão compor as nossas definições. São eles: o conceito de armazenamento e os registradores, cada um dos blocos que compõe um processador e suas finalidades, início da ideia de uma unidade de controle que irá comandar todo o processamento e, por fim, o conceito de instrução. Após entender o que é de fato uma instrução na ótica de um processador será possível analisar o que é o conjunto de instruções e seu impacto no projeto do *hardware*.

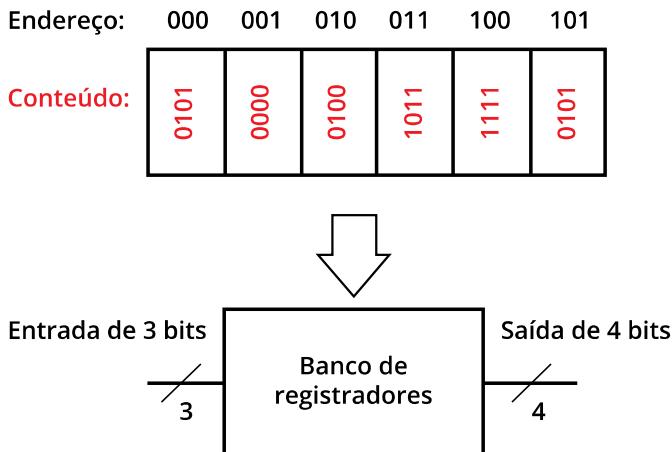
Toda a informação utilizada em um processador deve ser armazenada, por isso sempre que compramos um computador verificamos algumas informações tais como: sua capacidade de memória RAM, seu HD, se o processador possui um, dois ou três níveis de memória cache. Embora não saibamos neste momento exatamente o que significa cada parte na hierarquia de memória, sabemos que quanto mais memória um computador possui melhor seu processamento, certo? Mas não é tão simples assim...

O tamanho da memória está diretamente relacionado à complexidade de seu gerenciamento. Pense por exemplo em uma fita muito longa enrolada e que você precisa achar uma informação nessa fita. Você vai ter de ler parte por parte, à medida que desenrola, para achar o que precisa, e a este tipo de acesso damos o nome de **sequencial**. Agora considere que esta fita tenha 1 km de tamanho e que a informação está ao final dela (mas não sabemos disso), quanto tempo você vai demorar para encontrar? Seria mais fácil se a fita fosse menor? Sim, portanto é preciso lembrar que não adianta apenas uma memória maior, é necessário que seu acesso seja feito de forma inteligente.

Nosso interesse na hierarquia de memória está voltado para a unidade básica de armazenamento, o registrador. Em outras disciplinas, o funcionamento básico desta unidade será detalhado, mas, por enquanto, pense que existe no processador um banco de registradores. Este banco de registradores funciona da seguinte maneira: você passa um endereço de um determinado

registrador e o bloco de *hardware* retorna seu conteúdo. A Figura 3.1 apresenta um exemplo de banco de registradores.

Figura 3.1 | Exemplo de banco de registradores



Fonte: elaborada pelo autor.

O banco de registradores apresentado na Figura 3.1 possui uma entrada de endereços de 3 *bits* e retorna um número de 4 *bits*. Os fios que ligam os blocos do processador são chamados de **barramentos**, justamente porque possuem conjuntos de fios. No caso do nosso exemplo, um barramento de 3 *bits* recebe o endereço e o bloco de *hardware* retorna no barramento de 4 *bits* o dado correspondente. O acesso ao dado é feito por meio do endereçamento **direto**, ou seja, o endereço enviado é o próprio endereço do registrador e ativa apenas o setor da memória correspondente, tornando este tipo de acesso extremamente rápido e permitindo que seja implantado dentro do processador para o armazenamento de informações.

A manipulação dos registrados é realizada por meio de instruções dos processadores, as quais as principais são do tipo I, tipo R e tipo J. As instruções do tipo I manipulam endereços de memória ou constantes, como exemplo pode-se citar a adição de uma constante a um registrador ou um pulo (salto) para um outro endereço. As instruções do tipo R podem realizar a manipulação e as operações entre registradores, como a soma de um valor de um registrador a outro. As instruções do tipo J realizam desvios de instruções a serem executados.

Exemplificando

Vamos entender melhor o projeto de um banco de registradores utilizando um exemplo real. Em seu livro, Hennessy e Patterson (2017) propõem um processador chamado MIPS, cuja arquitetura foi utilizada como base de diversos processadores comerciais. O MIPS possui 32 registradores (do registrador 0 ao registrador 31) que armazenam dados de 32 bits de tamanho. Sendo assim, como deve ser o projeto deste banco de registradores considerando os bits de entrada e de saída?

A resposta é que o projeto deve ser similar ao que foi apresentado na Figura 3.1, porém a diferença é que para representar os 32 registradores é necessário um barramento de endereços de 5 bits, uma vez que $2^5 = 32$, ou seja, (registrador 0 – 00000_b, ao registrador 31 – 11111_b) e um barramento de saída de dados de 32 bits.

Agora que entendemos um pouco de registradores, é importante mencionar os dois tipos básicos de arquiteturas para computadores, a arquitetura de Von Neumann e a arquitetura de Harvard. Nas duas arquiteturas, o computador possui quatro componentes básicos: memória, unidade de controle, unidade lógica aritmética (ULA) e entrada/saída de dados. A diferença básica entre estas duas arquiteturas é o fato de que na arquitetura de Von Neumann a memória de programa e a memória de dados estão fisicamente no mesmo chip, porém separadas pelos limites de endereçamento de cada uma (por exemplo, a memória de programa é do endereço x ao y, e a memória de dados do endereço y+1 a z). No caso da arquitetura de Harvard existe uma separação física real da memória de dados e da memória de programas o que gera um custo maior de implantação (já que dobram-se os barramentos e o número de memórias necessárias), porém, a proteção é maior visto que se um vírus for executado, ele irá afetar apenas a memória de programa sem que os dados sejam perdidos.

Após entender os registradores e os tipos de arquiteturas, o próximo passo é entendermos a unidade de controle e a parte de entrada/saída de dados. Sem este entendimento seria complexo demais compreender como é formado um conjunto de instruções.

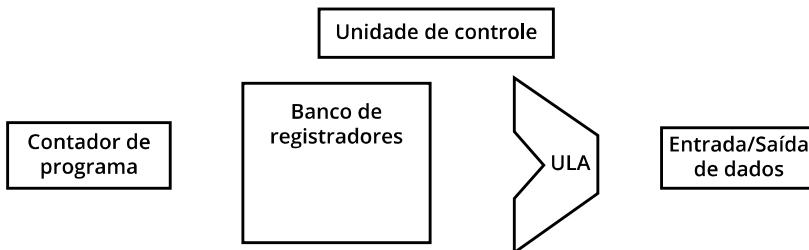
Ainda considerando a parte da hierarquia de memória, quando um dado não está armazenado em um registrador é possível que esteja ou na memória principal (representada em nosso sistema por níveis secundário de cache e também pela memória RAM) ou na memória secundária (representada pelo HD). Em uma instrução é possível também acessar endereços de memória que não são endereços de registradores e, nestes casos, é necessário que se tenha o endereço propriamente dito do dado ou então que se passe o valor

numérico a ser trabalhado pelo processador. Este tipo de endereçamento é chamado de **imediato** e irá gerar um tipo específico de instruções.

Após apresentar alguns conceitos básicos, vamos à definição de instrução. Segundo Hannessy e Patterson (2017), uma instrução é um conjunto de *bits* (0's e 1's), entendidos pelo processador como sinais eletrônicos, que podem ser representados por um conjunto de números. Cada um destes números representa algo específico para o processador e a junção destes números em uma palavra única é o que chamamos de **instrução**.

Para entender o papel de uma instrução vamos analisar os blocos que compõem o processador e o que cada um deve receber e como se dá o caminho dos dados através dos blocos. A Figura 3.2 apresenta os blocos que compõem um processador simples ainda sem conexão.

Figura 3.2 | Blocos utilizados para compor um processador ordenada conforme o caminho dos dados



Fonte: elaborada pelo autor.

O contador de programa é o registrador que irá armazenar a instrução que será executada pelo processador. Para simplificar a análise neste momento iremos considerar que o contador de programas sempre recebe uma instrução correta. Os outros blocos são configurados pelos *bits* que compõem a instrução. O banco de registradores deve receber os endereços dos dados e retornar os operandos corretos para a unidade lógica aritmética. Uma vez contendo os dados corretos, a ULA irá realizar o cálculo solicitado pela instrução e irá direcionar o resultado para um bloco de entrada/saída de dados. O bloco de entrada/saída de dados será responsável por colocar o resultado em seu local correto (seja na memória principal, seja no banco de registradores, seja direcionada para algum dispositivo específico).

Como cada bloco é capaz de saber qual a operação correta a ser realizada? Todos os blocos apresentados realizam diversos tipos de operações, então é necessário que sejam configurados para as operações corretas. O bloco que tem a função de configurar os outros blocos do processador é a unidade de controle (UC). A UC recebe, da instrução, *bits* de configuração e a partir

destes *bits* configura todo o restante dos blocos para que seja obtido o resultado correto da execução da instrução. Agora é possível considerar a seguinte questão para continuar nosso estudo: como devemos então projetar a nossa instrução para que todos estes requisitos sejam satisfeitos?

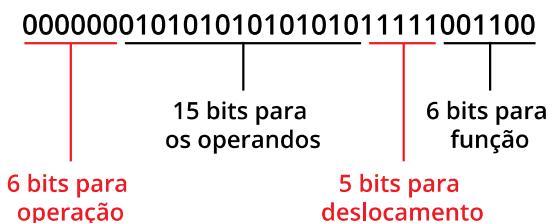
É necessário entender neste momento que a instrução é uma palavra que terá um determinado número de *bits*. Estes *bits* devem ser organizados de forma a conter todas as informações necessárias para a execução da instrução. Considerando projetos de processadores reais e complexos, é possível listar características básicas para as instruções:

- A instrução deve conter um campo para seu tipo de operação. Este campo deve ter um número de *bits* suficiente para mapear todos os tipos de operação contidos no projeto das instruções.
- Outro campo ser considerado é o campo da função dentro da operação. Por exemplo: em uma operação aritmética pode ser realizada uma soma, uma subtração, uma multiplicação. Então também é necessário que se tenha *bits* para este propósito.
- Usualmente utiliza-se no máximo 3 endereços para realizar uma operação sendo 2 operandos e um resultado. Portanto é interessante que uma instrução contenha espaço suficiente para que 3 operandos sejam descritos.
- Por fim, uma operação utilizada frequentemente é o deslocamento de bits (HENNESSY; PATTERSON, 2017). Neste caso, um campo de deslocamento é interessante para indicar em quantos *bits* deve ser deslocada uma cadeia de *bits*.

Assimile

O conceito de instrução deve ser assimilado de forma completa. Portanto vamos utilizar uma imagem que representa uma instrução de 32 *bits* com a mesma divisão proposta por Hennessy e Patterson (2017) para visualizar um projeto real.

Figura 3.3 | Exemplo de projeto de instrução



Fonte: elaborada pelo autor.

Nesta instrução, o campo de operações possui 6 *bits*, sendo que podemos então mapear 64 operações diferentes. Considerando que para cada operação temos ainda 6 *bits* de função é possível mapear 64 funções para cada operação. O campo de deslocamento possui 5 *bits*, isto indica que podemos deslocar os bits de uma palavra até 32 vezes (lembra que deslocar significa movimentar um algarismo para a esquerda ou para a direita se considerarmos que 1 *bit* pode ser o sinal + ou -). Por fim restam os 15 *bits* para os operandos. Estes *bits* podem representar três registradores de cinco *bits* cada, ou então representar no mínimo um registrador e o restante dos *bits* ser utilizado como endereço. Veremos mais à frente quando este modelo é utilizado.

Veja: este projeto considera uma palavra relativamente grande, de 32 *bits*, e os campos são divididos para permitir o projeto de diversas instruções (64 operações com 64 funções cada). Este projeto pode não ser necessariamente uma necessidade, porém é um modelo que pode ser vir de base para qualquer projeto de instruções.

Agora que sabemos como deve ser uma instrução é possível entender como os diferentes tipos de endereçamentos são implementados. Os campos da instrução podem ser interpretados de formas diferentes de acordo com o que é apresentado à unidade de controle (UC) do processador. Em uma arquitetura básica são necessários no mínimo 3 tipos de instruções (HENNESSY; PATTERSON, 2017):

- As instruções regulares que interpretam os campos exatamente como a descrição base do projeto (como no exemplo da Figura 3.3).
- As instruções de acesso imediato que necessitam de uma mudança de interpretação para permitir que mais *bits* sejam utilizados para representar um número direto, ou mesmo um endereço em hierarquias superiores de memória. Um exemplo deste caso seria interpretar os 5 últimos *bits* do campo de endereços, o campo de *shift* (deslocamento) e o campo de função da instrução da Figura 3.3 como um número de 16 *bits*.
- Por fim, um conjunto de instruções executado em um processador possui situações de condição e desvio para outras instruções que não sejam necessariamente a próxima (o famoso “goto” das linguagens de programação). Este caso gera um tipo de instrução especial de desvio que possui apenas os 6 *bits* de operação e os outros todos para o endereço do desvio (no caso da instrução da Figura 3.3 são 26).

Refita

O projeto do processador que estamos abordando nesta seção é um projeto simples que não requer muitos desdobramentos do projeto de *hardware*. Agora considere um processador mais complexo como o processador que você tem em seu *notebook*. As instruções deste processador seriam mesmo tão simples quantas estas? Ou seriam necessários outros tipos de instrução? Lembre-se que as instruções devem permitir que o *hardware* seja todo utilizado, portanto as diversas plataformas de memória e periféricos encontrados em seu computador podem exigir modificações.

Após entender os tipos de instrução é necessário que sejam revistos os modos de endereçamento que existem em processadores para sintetizar os conhecimentos. Os tipos são:

- Endereçamento imediato – este tipo é utilizado nas instruções do tipo I em que um conjunto de *bits* irá representar um valor constante.
- Endereçamento em registrador – quando colocamos o endereço do registrador direto na instrução. Este modo de endereçamento ocorre nas instruções **de tipo R e I**, em que são indicados três e dois endereços de registradores respectivamente.
- Endereçamento de base ou deslocamento – neste caso para encontrar o endereço desejado é necessário somar o conteúdo de um registrador com a parte constante da instrução. Normalmente utilizada nos desvios condicionais.
- Endereço relativo ao contador de programa (PC) – é o mesmo caso da base e deslocamento, porém o que é considerado como base é o endereço do PC.
- Endereçamento pseudodireto – é o tipo de instrução de desvio condicional, ou seja, são utilizados 26 *bits* para endereçar a próxima instrução. Seria a instrução de *jump*.

Exemplificando

Como exemplo para encerrar esta seção iremos executar um código de máquina simples em um simulador e verificar seu resultado. O simulador escolhido é chamado de MARS (você o encontra para download ao consultar a referência seguinte).

MISSOURI STATE UNIVERSITY. **MARS (MIPS Assembler and Runtime Simulator)**. An IDE for MIPS Assembly Language Programming). 2014.

Trata-se de um simulador da arquitetura MIPS proposta por Hennessy e Patterson (2017). Ele é capaz de executar códigos em linguagem de máquina, demonstrar as mudanças nas instruções e na memória.

O simulador é um programa gratuito gerado em java (.jar) que deve ser apenas executado. Ao iniciar o programa (clicar duas vezes em seu ícone principal) devemos ir em File-> new.

Nesse exemplo iremos carregar os valores nos registradores e realizar a sua soma.

Assim, no campo de digitação o código a seguir deve ser digitado.

```
.text  
li $s2, 10      # carrega o valor IMEDIATO 10 em s2 (li = load  
(carregar)  
li $s3, 15      # carrega o valor IMEDIATO 15 em s3  
Add $s1, $s2, $s3 # Executa a soma de s2 com s3 e armazena o  
resultado em s1
```

É importante ressaltar que após o símbolo # temos comentários.

Li é a instrução load (ou carregar).

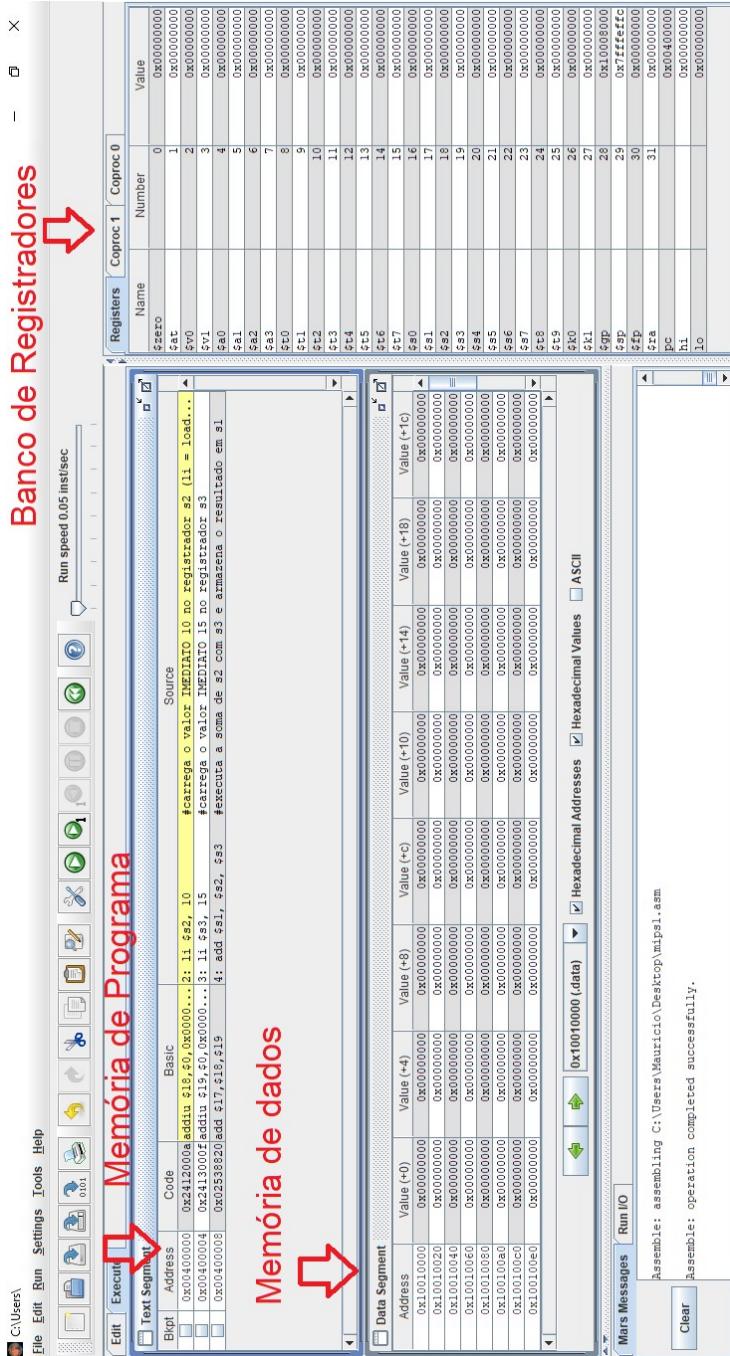
Add é a instrução para realizar somas.

Sub realiza as subtrações.

Em seguida clicamos em Run -> Assemble para que o código seja montado. Ao fazer isso, a tela irá mudar e o código terá sido colocado na memória de programa. A coluna Address indica o endereço na memória de endereço, a coluna code indica o hexadecimal da instrução, a coluna basic é o código com os imediatos convertidos em hexadecimal. A coluna na janela à direita é a memória. Antes de executar, a barra Run speed que fica no canto superior esquerdo deve ser modificada para 0.05 inst/sec para que seja possível ver a execução.

Após feito isso executamos o código em Run -> Go. O código irá carregar representado pela instrução li (load ou carregar) os números para os dois registradores e o resultado será armazenado em s1. Observe com calma e interprete os números alterados nas memórias do simulador. Com esse simulador podemos testar outras instruções que serão apresentadas nas próximas seções. Fique à vontade para explorar o programa neste momento seguindo os mesmos passos.

Figura 3.4 | Indicação das memórias no simulador



Fonte: captura de tela do simulador MARS.

Esta seção tratou de questões importantes sobre as instruções e o projeto do processador. Nesta seção, você aprendeu o que é uma instrução, os tipos diferentes de instrução e como os diferentes tipos fazem referências à memória. Neste momento, você já tem os conhecimentos para elaborar os projetos iniciais de conjuntos de instruções, com os tamanhos de cada uma e identificar a importância de projetar instruções com tamanhos adequados. Agora, você já é capaz de solucionar a situação-problema desta seção. Preparado? Então, boa sorte!!

Sem medo de errar

Como apresentado anteriormente, o problema que iremos resolver é o problema da criação de um modelo para um conjunto de instruções. O processador que iremos projetar deve ser simples, porém pode ser pensado para um tipo de operações específicas. Os processadores possuem sempre uma unidade que deve ser especializada na execução de operações aritméticas como soma, subtração, divisão, multiplicação, dentre outras (HENNESSY; PATTERSON, 2017).

A segunda questão importante é que este conjunto de instruções deve ter definido um tamanho de palavra que o processador será capaz de aceitar. Considerando que o processador terá de se comunicar com outros blocos é possível escolher um tamanho de palavra padrão. Os processadores mais novos estão operando em *64 bits* (que é o tamanho da sua palavra), porém, o nosso processador pode ser projetado considerando que um dado de *64 bits* irá precisar de toda uma estrutura de suporte maior como o barramento de transmissão de dados, os registradores para armazenar os dados, dentre outras estruturas. Sendo assim, para manter um tamanho de arquitetura pequeno e compatível com os projetos já executados no mercado vamos optar por uma **arquitetura de *32 bits***.

As duas primeiras etapas foram então cumpridas e justificadas. Veja: você como projetista poderia escolher qualquer número de *bits* e um núcleo especializado em outras funções. Porém, modificar estas características do problema exige normalmente conhecimentos mais avançados sobre as aplicações para as quais o processador seria desenvolvido, e um tamanho maior de palavra deve ser justificado com a necessidade de mais *bits* para representação. Um número muito pequeno destes *bits* pode causar um problema caso sua arquitetura do processador precise passar por um *upgrade* e isso também deve ser considerado.

Agora, para cada um dos *bits*, ou conjunto de *bits*, devemos escolher uma funcionalidade correspondente. Como ainda não conhecemos a fundo as funcionalidades necessárias, é possível tentar sem medo definir este modelo de instrução. Mais à frente no estudo de arquitetura de computadores poderemos analisar a resposta que demos aqui e entender o motivo de ser boa ou ruim para determinados aspectos.

O processador lida normalmente com operações numéricas, sendo que os números envolvidos estarão armazenados em uma memória. Assim existe a necessidade de dois tipos de campos: um onde acessamos o número em um registrador (como nas instruções do tipo R apresentadas) e outro que o número é obtido diretamente na instrução (como nas instruções do tipo I apresentadas).

Existem processadores inteiros como o MIPS (HENNESSY; PATTERSSON, 2017) que são definidos com 32 instruções e outros como os processadores que possuem centenas. O número de *bits* que reservarmos para o tipo de instrução deve ser capaz de identificar unicamente cada uma delas. Sendo assim podemos deixar 1 *byte* para a **identificação de instruções, ou seja, 8 bits** como visto nesta unidade. Também é comum reservar *bits* para controle da arquitetura que no nosso caso serão **6 bits**. Os outros **18 bits** podem ser divididos da seguinte maneira: para instruções do **tipo R**, três grupos de **6 bits**, e para instruções do **tipo I** dois campos um de **6 bits** e um de **12 bits**.

Vamos analisar esta resposta. Reservando 1 *byte* para instruções poderemos identificar unicamente instruções, ou seja, 256. Este número é razoável já que estamos projetando um processador apenas para cálculos matemáticos, pois são poucas as possíveis operações matemáticas. Os outros 6 *bits* para controle do sistema são importantes já que podem ser utilizados para configurar outras partes do processador. Neste caso permitimos, com 6 *bits*, que existam opções de configurações totalizando 64 o que é suficiente também para o controle dos blocos lógicos.

O tipo R possui 3 campos de 6 *bits* cada. Um campo de 6 *bits* para o endereço de um registrador (lugar onde fica armazenado um valor na memória) permite endereçar 64 registradores (do 0 ao 63). Normalmente, um projeto de processador simples não possui mais que 40 registradores sendo um número aceitável. Por fim, para a instrução do tipo I, temos novamente os 6 *bits* para endereçar um registrador mantendo o padrão anterior. No caso da parte imediata deixamos 12 *bits*, isso quer dizer que quando o programador do processador quiser passar um número direto para o processador poderá usar apenas do 0 ao 4095. Isso é uma limitação que podemos discutir mais adiante.

Pronto, projetamos assim a nossa instrução de 32 *bits* com:

- 8 *bits* para identificação da instrução.
- 6 *bits* para controle do processador.
- 18 *bits* para informações sobre os operandos.
- No tipo R são três endereços de 6 *bits* para os registradores.
- No tipo I é um conjunto de 6 *bits* para um endereço e 12 *bits* para um número imediato.

Faça valer a pena

1. A arquitetura de um processador é um segredo industrial já que irá determinar o quanto eficiente é o processador em relação a consumo de energia, tempo de execução, tolerância a falhas. Entretanto existem alguns componentes que são base para todas as arquiteturas.

Assinale a alternativa que contém um componente que não faz parte de um processador de computador.

- a. ULA.
- b. UC.
- c. Registradores.
- d. Memória RAM.
- e. Unidade de deslocamento.

2. Registradores são pequenas porções de memória dentro do processador usados para armazenamento temporário de dados. Os registradores podem ter propósito geral e armazenar qualquer tipo de dado a ser utilizado para a execução de instruções, como ter propósito específico e armazenar dados específicos definidos na arquitetura do computador.

Dentre os tipos de registradores há o contador de programa que:

- a. também é conhecido como cache de memória por fazer a intermediação entre a unidade de busca do processador e a memória RAM.
- b. armazena o endereço da próxima instrução que será carregada na memória.
- c. conta quantas vezes o programa foi executado.

- d. possui apenas um *bit* para indicar que uma situação particular ocorreu, como *overflow*.
- e. envia pedidos ao processador para que pare a execução de um programa e atenda outro processo incondicionalmente.

3. Diversos aspectos devem ser considerados ao se projetar o conjunto de instruções de um processador já que o impacto causado pelas instruções no caminho de dados sobre o projeto do processador é alto.

Sobre a relação entre o caminho de dados de um processador e seu conjunto de instruções, analise as afirmações abaixo:

- I. O caminho de dados de um processador influencia no projeto do conjunto de instruções.

PORQUE

- II. Já que não se pode definir um conjunto de instruções sem saber como é o caminho de dados do processador.

A respeito dessas asserções, assinale a alternativa correta.

- a. A asserção I é uma proposição verdadeira e a II, falsa.
- b. A asserção I é uma proposição falsa e a II, verdadeira.
- c. As asserções I e II são proposições verdadeiras, mas a II não justifica a I.
- d. As asserções I e II são proposições verdadeiras e a II justifica a I.
- e. As asserções I e II são proposições falsas.

Seção 3

Pipeline de instruções

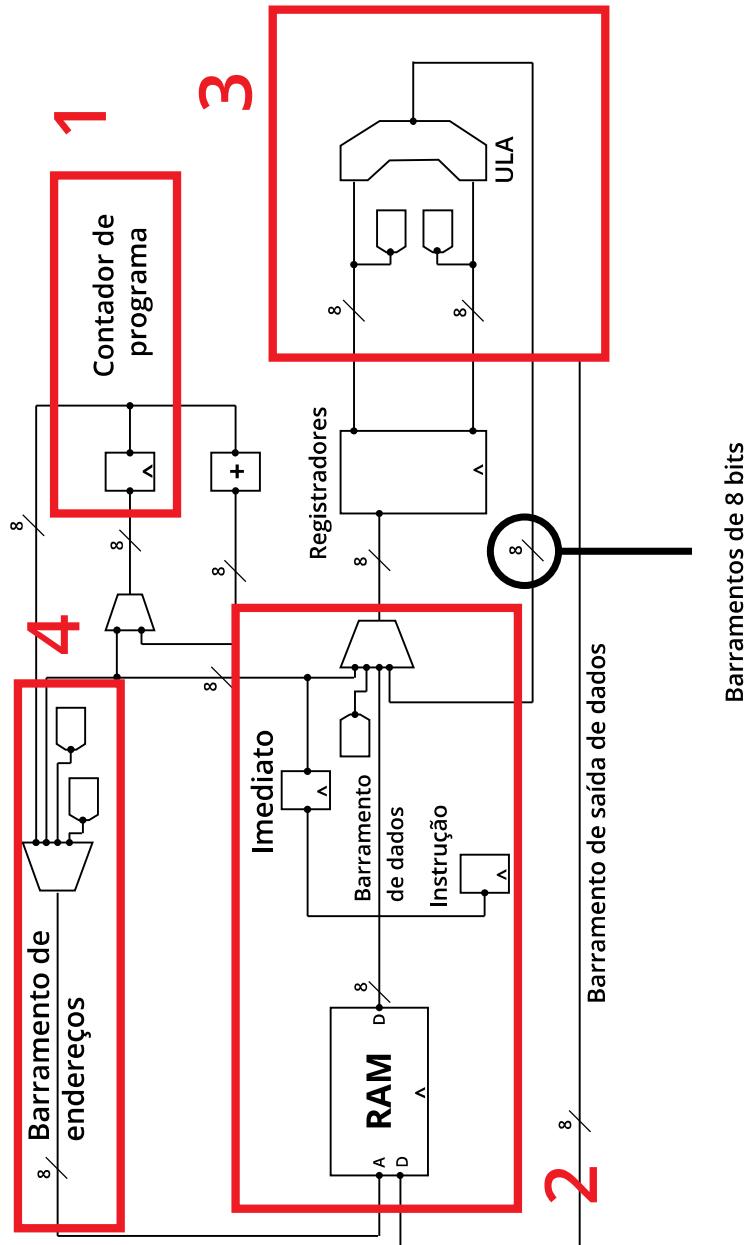
Diálogo aberto

Os processadores atualmente possuem diversos níveis de otimização buscando executar os programas de forma cada vez mais rápida, porém o mais comum em todos eles é o *pipeline* que é assunto desta seção. Desde as primeiras versões de processadores, a possibilidade de se executar mais de uma instrução ao mesmo tempo, em paralelo, parecia uma boa medida para acelerar a execução de programas. Os projetos dos primeiros processadores eram limitados pelas técnicas de projeto e fabricação da época que não permitia uma grande utilização de transistores, nem uma complexidade muito elevada do *hardware* final.

O avanço das tecnologias de fabricação e das ferramentas de projeto permitiu o desenvolvimento dos processadores e este fato deu margem ao início dos estudos de otimização na estrutura básica do *hardware*. A implementação da execução de uma instrução sem que a anterior tenha terminado completamente tem base no fato de que dependendo do estágio de execução em que se encontra uma instrução, várias partes do *hardware* ficam ociosas e poderiam ser aproveitadas. A divisão das etapas de execução da instrução permite que as áreas que já foram utilizadas possam executar a próxima instrução e este conceito é o que chamamos de *pipeline*. Quanto maior o número de estágios no *pipeline*, maior o número de instruções que podem ser executadas.

Visualizando o caminho de dados de um determinado processador com barramentos de 8 *bits*, ou seja, a quantidade de dados que flui é de 8 *bits*, é possível não só identificar as etapas do processador, como também propor modificações e verificar possíveis otimizações. Você, como pertence ao grupo de projetistas de *hardware* da X recebeu uma nova tarefa. A equipe de engenheiros recebeu a imagem do caminho de dados de um processador que será fabricado pela empresa concorrente Y apresentado na Figura 3.5.

Figura 3.5 | Arquitetura do processador da empresa X



Para que a sua empresa não seja prejudicada, sua tarefa é identificar neste diagrama as etapas de execução da instrução e se este processador tem potencial para otimizações. Com o conhecimento adquirido na seção será possível realizar a tarefa e proporcionar uma análise detalhada do caminho de dados do processador da empresa Y. Você está preparado para aprender como fazer isso? Vamos lá!

Não pode faltar

O conceito de paralelismo atualmente é muito difundido, principalmente, pelo aumento de núcleos dos processadores. Esses núcleos são utilizados em paralelo para a execução de diversas tarefas do computador. Porém, a tecnologia de se colocar vários núcleos em um processador foi desenvolvida após terem se esgotado as possibilidades de otimização em apenas um núcleo. Esta otimização ocorreu por muito tempo considerando o aumento no *clock* do processador e no número de instruções que se conseguia executar em um mesmo ciclo de *clock*. As limitações encontradas no desenvolvimento foram físicas.

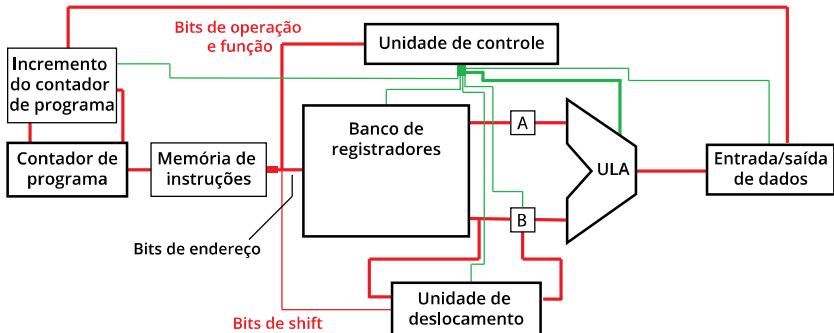
A primeira delas, a frequência de *clock*, indica de quanto em quanto tempo um pulso é liberado realizando as operações do processador. O problema é que este valor é definido pelo que se chama de caminho crítico do *hardware*, que é o tempo de execução da instrução mais demorada do conjunto de instrução. Vamos pensar o seguinte: se a instrução mais demorada do processador demorar 1 segundo, e o processador executar uma instrução por vez, a frequência do processador deve ser de 1 Hz, ou seja, 1 instrução por segundo. Mesmo tendo instruções mais rápidas, este tempo tem de ser respeitado para que a execução ocorra corretamente. A diminuição do tamanho dos componentes e a otimização dos processadores permitiu uma diminuição grande dos caminhos críticos, porém, ao executar muito rápido as instruções, o processador esquenta e sem a dissipação de calor adequada, ele “queima”. Porém uma frequência maior que 3.2 GHz aproximadamente faz com que o processador esteja muito suscetível a queimar ou ter sua vida útil diminuída. A saída neste caso foi a utilização de mais de um núcleo de processamento no mesmo *chip* melhorando o tempo de processamento sem prejudicar o *hardware* (já que a frequência de *clock* de cada núcleo não atinge mais que 3Ghz). Para entender toda esta evolução vamos iniciar de um processador mais simples.

O caminho de dados de um processador é composto pelos blocos apresentados na Figura 3.6, algumas estruturas auxiliares para a correta execução da instrução e as ligações entre todos estes elementos de *hardware*. Vamos

analisar então o caminho de dados necessário para executar uma instrução do tipo regular.

A Figura 3.6 apresenta o caminho de dados do processador proposto neste material implementando apenas o tipo de instrução regular. Vamos então analisar agora em detalhes cada uma dessas estruturas e suas conexões para entender como apenas um tipo de instrução gerou este caminho de dados.

Figura 3.6 | Caminho de dados para a instrução do tipo regular



Fonte: elaborada pelo autor.

A primeira questão a ser abordada é o contador de programa. Este é um registrador especial que guarda o endereço na memória da instrução a ser executada. No fluxo normal de programa, a unidade de incremento do contador de programa apenas faz com que o endereço atual seja substituído para o endereço da próxima instrução. O problema neste caso é que quando temos um desvio é necessário que este endereço seja atualizado. Por isso existe um sinal de controle (em verde) que chega neste bloco. Este sinal de controle irá dizer se o contador de programa receberá a próxima instrução ou o resultado do cálculo do desvio que vem do bloco de entrada/saída de dados.

O endereço da próxima instrução é passado por um barramento para a memória de instruções. Esta memória é a responsável por armazenar o conjunto de instruções que será executado. Na Figura 3.6 ela está representada de forma separada, porém pode ser representada junto do banco de registradores em uma memória só ocasionando algumas mudanças no diagrama. Em sua implementação real, estas memórias todas representam o conjunto memória RAM + memória cache do computador.

A saída do bloco da memória de instruções são os *bits* em barramento, sendo que o tamanho deste barramento é o mesmo do tamanho escolhido para a instrução. Neste momento, os *bits* da instrução se dividem. Os *bits* de operação e função irão para a unidade de controle que irá distribuir os sinais para todo o

processador (representados em verde). Os *bits* de endereço vão para o banco de registradores para identificar os dados necessários para o cálculo a ser realizado. Os *bits* indicando o deslocamento são direcionados a uma unidade de deslocamento que irá apresentar o resultado caso houver deslocamento (indicado pelo sinal de controle que chega na unidade de deslocamento).

Em uma instrução regular saem dois operandos do banco de registradores e é padrão nas arquiteturas que o primeiro operando (armazenado no registrador A) saia direto do banco de registradores (HENNESY; PATTERSON, 2017), porém no caso do segundo operando é necessário trabalhar as possibilidades. No caso da instrução regular, ele pode vir direto do banco de registradores ou de um deslocamento de *bits*. A escolha é feita pelo sinal de controle que chega no registrador B que armazena o segundo operando.

Os sinais de controle são o conjunto de *bits* que podem acionar determinada estrutura, desativar determinada estrutura ou então selecionar alguma resposta dentre um grupo de possibilidades. O acionamento ou a desativação de estruturas é necessário devido ao fato de que a estrutura do *hardware* executa todas as possibilidades paralelamente, então uma instrução de um determinado tipo está sendo executada como se fosse de todos os tipos ao mesmo tempo. Para selecionar corretamente o tipo é preciso controlar quais partes do *hardware* devem ser consideradas e esse é o papel da unidade de controle e dos sinais de controle. No caso do registrador B, imagine que existem duas possibilidades de entrada: uma que vem do banco de registradores e outra que vem da unidade de deslocamento. Seria então necessário escolher de onde vem a informação. Neste caso podemos colocar um sinal de controle de 1 *bit* que quando for 0 escolhe como origem o banco de registradores e quando for 1 escolhe como origem a unidade de deslocamento.

Com os dois operandos prontos, a operação matemática escolhida pela instrução através dos *bits* de controle (sinal verde que chega na ULA) é executada e seu resultado é armazenado temporariamente em uma unidade de entrada/saída de dados. O sinal de controle verde que chega nesta unidade irá indicar para onde o resultado deve ir. No caso das instruções regulares, ele sempre é armazenado em um registrador também, por isso do caminho de dados em negrito retornando ao banco de registradores. A outra saída do bloco de entrada/saída de dados é para os casos em que algum desvio resultar em um incremento no contador de programas.

Para a questão da implementação de instruções imediatas, temos o diagrama abaixo da Figura 3.6, onde foram adicionados um bloco de *hardware* logo abaixo do banco de registradores chamado de complemento e um sinal de controle que chega neste bloco. Este novo bloco, complemento, é responsável por fazer o complemento binário do número que possui um número de *bits* menor que a palavra do processador.

Para relembrar como é feito o complemento binário, imagine uma cadeia de 16 *bits* que precisa ser processada pelo nosso processador de 32 *bits*. É necessário que se adicionem 16 *bits* na cadeia para que o número possa ser utilizado para cálculos na ULA. O processo de complemento simplesmente replica o *bit* mais significativo quantas vezes for necessário para que seja atingido o número de *bits* necessário. Veja o exemplo abaixo:

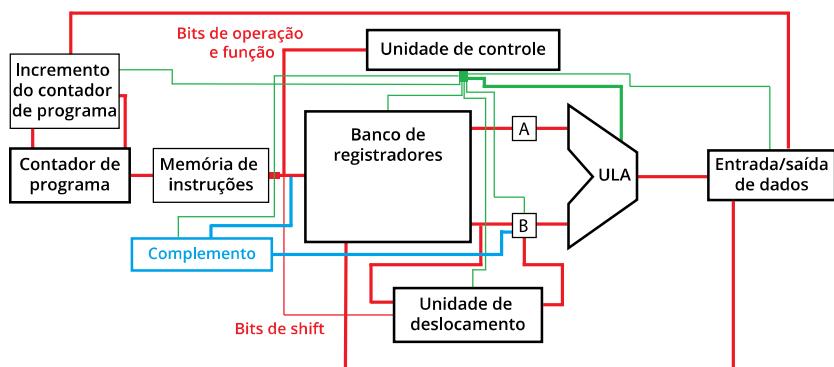
Exemplo de entrada de 16 bits:

0001110001101010

Resultado do complemento: **00000000000000000001110001101010**

No caso das instruções imediatas ou de desvio incondicional o número de *bits* que indica o endereço (no nosso estudo de caso, 16) é sempre menor do que o número de *bits* da palavra como um todo (no nosso estudo de caso, 32). Em seguida, este novo número será enviado ao operando B e então é mais uma escolha possível para ser utilizado na operação aritmética a ser executada. Este caminho novo permite que as instruções que possuam parte imediata, seja de endereço ou numérica, sejam executadas.

Figura 3.7 | Caminho de dados com estrutura para instruções imediatas



Fonte: elaborada pelo autor.

Você, neste momento, conhece o projeto de um processador. Este projeto é um projeto chamado de monociclo e possui alguns problemas. Está em dúvida do que seria isso? Bem, para entender melhor todos estes conceitos iremos iniciar com o mais simples de todos, o sinal de *clock*.

Quando vamos comprar um processador, as principais informações que procuramos são: a quantidade de núcleos, a quantidade de memória cache e

a frequência do processador (aquele número 1.8 GHz, 2.4 GHz). Apesar de sabermos que quanto maior a frequência, melhor, não é comum o completo entendimento do que isto representa para o *hardware*. Este número é a frequência de *clock* em que o processador trabalha. Calculamos a frequência como o inverso do período, ou seja, se o caminho crítico do nosso processador (tempo maior de execução de uma instrução) for 1 nanosegundo teremos a Equação 3.8:

$$f = \frac{1}{T} = \frac{1}{1 \cdot 10^{-9}} = 10^9 = 1GHz \quad 3.8$$

Certo, então isso quer dizer que a cada $1 \cdot 10^{-9}$ s um pulso é liberado e ativa todas as estruturas do processador conectadas a este sinal de *clock*. Isso significa que todas as informações que estavam na entrada de cada um dos blocos da arquitetura que construímos serão processadas e colocadas nas respectivas saídas dos blocos. Caso até o próximo pulso de *clock* essa informação não tenha sido processada, tudo que foi feito se perde e uma nova operação é iniciada com as informações de entrada atuais. Por este motivo é tão importante respeitar o tempo do caminho crítico na construção do processador.

Agora que sabemos o que é um sinal de *clock* já podemos passar para a próxima etapa. A arquitetura que fizemos até o momento é classificada como **monociclo**. Isto significa que todas as instruções são processadas em apenas um pulso de *clock*, tanto as mais demoradas quanto as mais rápidas. Este tipo de projeto, apesar de simples, não é muito inteligente porque se uma instrução termina muito rápido, a próxima só será executada no próximo ciclo e todo este tempo o processador fica ocioso. Uma das formas de melhorar esta situação é quebrar as instruções em partes menores e executá-las cada uma em um ciclo de *clock*. Desta forma diminui-se o tempo ocioso do processador e é possível executar mais de uma instrução no mesmo pulso de *clock*. Um pouco confuso não? Vamos entender melhor como funciona este processo.

Vamos inicialmente dividir as instruções em pequenos blocos que realizam operações. Esta divisão é apresentada por diversos autores, porém privilegiaremos a abordagem proposta por Hennessy e Patterson (2017). Todas as instruções possuem ao menos quatro etapas:

- A etapa de busca: quando o endereço armazenado no contador de programa é utilizado para buscar a instrução na memória de programa.
- A etapa de decodificação: após a recuperação da instrução, os bits são divididos e enviados às principais unidades do processador. Neste momento que a unidade de controle envia os sinais de controle ao processador.

- A etapa de execução: com o processador configurado corretamente a instrução propriamente dita é executada pelos blocos de *hardware*.
- A etapa de armazenamento do resultado: após a execução, o resultado é direcionado ao local indicado para seu armazenamento.

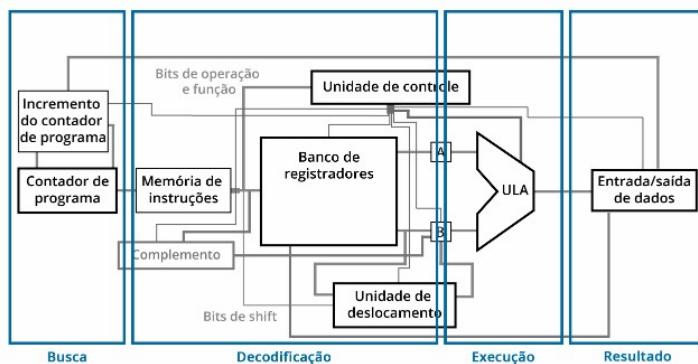
O tipo de instrução que gera um passo a mais é a instrução utilizada para carregar o conteúdo da memória para um registrador ou do registrador para a memória. Nestes casos, o cálculo é realizado para obter o endereço da memória onde o dado vai ser salvo, então é necessário um último passo quando o dado é realmente salvo na memória ou no banco de registradores.

Assimile

Para entender as etapas da divisão da instrução, vamos apresentar um diagrama do processador monociclo para, então, ser possível modificá-lo a fim de receber os pulsos de *clock* tornando-se assim uma arquitetura multiciclo.

Lembre-se que dividir as instruções em partes é uma abstração, ou seja, no caso vai facilitar a implementação da arquitetura multiciclo.

Figura 3.8 | Caminho de dados com representação da divisão



Fonte: elaborada pelo autor.

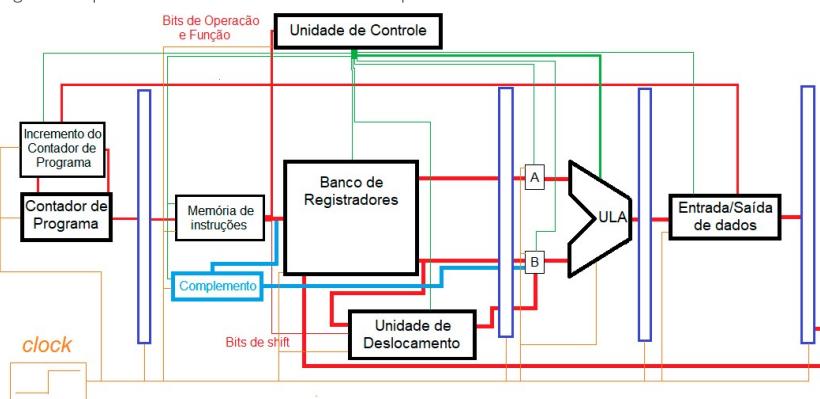
Como é possível visualizar na Figura 3.8, as etapas de execução da instrução podem ser divididas, sem maiores problemas, como proposto. As etapas são divididas em busca da instrução na memória de instruções, depois decodificação, execução e armazenamento do resultado. Estas quatro etapas vão ajudar na construção do processador multiciclo.

Certo, agora que foi possível compreender melhor como é feita a divisão das etapas da execução da instrução, podemos continuar nossa análise. Foi dito anteriormente que, caso uma execução não tenha terminado e outro pulso de *clock* seja acionado, tudo que foi feito é reiniciado sem o resultado da primeira execução. Para que resultados sejam armazenados entre os ciclos de *clock* é necessário acrescentar algumas estruturas de *hardware* em nosso diagrama.

Os registradores são estruturas que são acionadas por pulsos de *clock*, ou seja, independentemente de o dado estar disponível no barramento para ser acessado, ele só irá ser efetivamente armazenado após o próximo pulso de *clock*. Sendo assim, registradores criam uma “barreira” entre pontos de execução. Portanto, as duas modificações básicas a serem colocadas no diagrama a fim de proporcionar uma execução de instruções multiciclo são: o sinal de *clock* e os registradores.

Outra pequena modificação que deve ser feita é um sinal chamado “enable”. Este sinal irá acionar o registrador juntamente com o *clock* para armazenar o resultado. Vamos entender seu propósito então com uma situação exemplo: imagine que uma soma demore apenas 1 ciclo de *clock* para ser realizada uma multiplicação demore 10 ciclos, se os registradores ficarem sensíveis apenas ao sinal de *clock* a cada ciclo o resultado que está na saída será armazenado. Se for uma multiplicação o armazenamento estará errado já que o resultado só sai após 10 ciclos. Sendo assim, precisamos fazer alguns registradores serem sensíveis ao sinal de “enable” também que só é gerado após o término de uma operação.

Figura 3.9 | Caminho de dados multiciclo completo



Fonte: elaborada pelo autor.

No diagrama da Figura 3.9 é possível ver como seria o caminho de dados de um processador multiciclo. Neste caso, as mudanças principais são: a existência de um sinal de *clock* conectado a todos os blocos de *hardware* do processador (representado por uma onda quadrada que varia entre 0 e 1 em um período que respeita o caminho crítico), a inclusão de 4 registradores que irão armazenar os estados entre as etapas (previamente definidas na Figura 3.8) e, por fim, a ligação das saídas dos blocos de *hardware* diretamente nos registradores para que seja possível salvar o estado de execução.

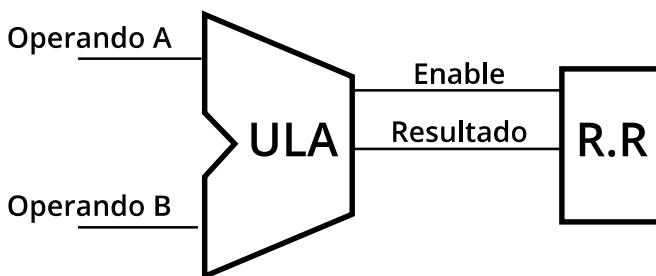
A estrutura do processador agora permite que cada parte seja executada e armazenada nos novos registradores que iremos chamar de **registradores de estado**. Também é necessário considerar que para todos os registradores de estado é necessário ter um sinal de “enable” que foi omitido deste diagrama por ser comum a todas as estruturas e fazer parte de todos os barramentos. Neste momento, basta que ao analisar o diagrama consideremos a existência de um sinal de “enable” em cada saída dos blocos, e que estes sinais em conjunto irão ativar o respectivo registrador de estado para armazenar o resultado correto.

Exemplificando

Vamos compreender melhor quando devemos utilizar o sinal de “enable” e qual a forma correta de projetar um registrador utilizando este conceito.

Considere um registrador que armazena uma informação que deve estar disponível apenas quando o final de um cálculo terminar, caso contrário a informação no registrador será inconsistente. Sendo assim vamos propor uma estrutura e explicar seu funcionamento.

Figura 3.10 | Unidade lógica aritmética com sinal de enable



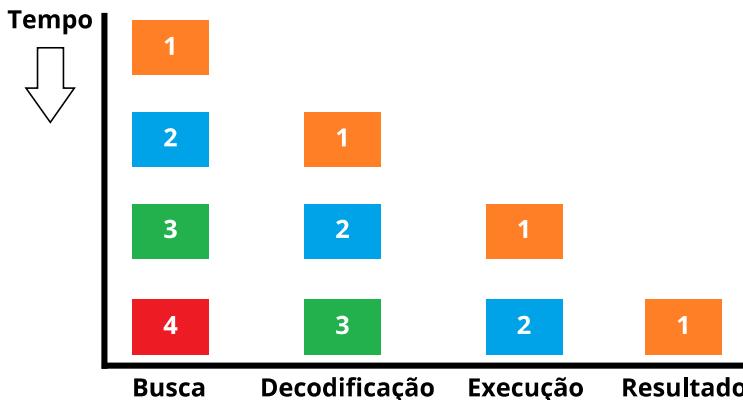
Fonte: elaborada pelo autor.

Na Figura 3.10 temos uma ULA simples que recebe dois operandos e realiza determinada operação aritmética. Na saída desta ULA estão dois sinais: o sinal de resultado conecta a saída da ULA ao registrador de resultado (R.R) para que o resultado seja corretamente armazenado, e o sinal de enable que neste caso é um sinal de 1 bit que só recebe o valor '1' quando a operação termina. Este procedimento faz com que o registrador R.R só receba o dado da ULA quando a operação terminar. Este exemplo demonstra a importância do sinal de *enable* no hardware.

Obviamente que ao se evoluir uma arquitetura novas possibilidades de otimização são criadas. O fato de se ter registradores de estado sugeriu que uma otimização poderia ser feita para acelerar a execução do processador. Anteriormente, a tecnologia de colocar 2 ou mais núcleos em um mesmo processador tinha um custo muito alto devido não só ao projeto, mas também à área de *chip* utilizada. Este impedimento sugeriu outro tipo de mudança. Por volta de 1939, o projeto ILLIAC II da IBM já explorava o que seria o início da técnica de *pipeline* (ROJAS, 1997). Vamos entender a ideia central desta otimização.

Inicialmente dividimos as etapas de execução da instrução (Figura 3.7), em seguida colocamos registradores para guardar os estados entre as etapas (Figura 3.8). Isto indica que uma parte da execução que já tenha executado seu cálculo pode iniciar a execução da próxima instrução. Para entender na prática a mudança vamos analisar o gráfico da Figura 3.11.

Figura 3.11 | Exemplo de pipeline de execução



Fonte: elaborada pelo autor.

A ideia é a seguinte: em um tempo inicial a instrução 1 encontra-se na fase de busca. Quando esta fase termina entramos no tempo seguinte quando a instrução 1 passa para a fase de decodificação e a instrução 2 entra na fase de busca. Continuando a execução das instruções no tempo seguinte, a instrução 1 estará na fase de execução, a instrução 2 estará na fase de decodificação e a instrução 3 estará na fase de busca. Na última fatia de tempo mostrada no gráfico temos 4 instruções sendo executadas ao mesmo tempo, cada uma em um estágio de execução diferente.

Esta possibilidade de manter todas as estruturas do processador em funcionamento e executar mais de uma instrução ao mesmo tempo é chamada de **pipeline de instrução**. A última linha da Figura 3.11 representa a situação que dizemos que o *pipeline* está cheio (ou seja, todas as partes do processador estão em execução). Em um *pipeline* de 4 estágios como o da Figura 3.11, o tempo para “encher” o *pipeline* é exatamente igual ao número de estágios. Em um *pipeline* de instruções, o ideal seria que o tempo total de execução fosse a divisão do tempo sem *pipeline* pelo número de estágios. Porém existe a demora para o *pipeline* começar a funcionar e depois terminar todas as instruções.

Assimile

Vamos entender agora como o *pipeline* melhora o tempo de execução e como calcular numericamente esta melhora. Aumentar o número de instruções que se pode executar ao mesmo tempo gera um aumento no número de instruções processadas pelo processador em um determinado período. Portanto, o maior ganho que temos na implantação do *pipeline* de instruções é no número de instruções processadas por segundo, métrica chamada de *throughput*. Existe também uma forma de calcular o tempo necessário para executar um conjunto de instruções em um processador com *pipeline*.

Segundo Hennessy e Patterson (2017), considerando um conjunto de instruções *I* em um *pipeline* com *E* estágios e com clock *C* (representado em segundos) temos o tempo de execução considerando a relação expressa pela Equação 3.9.

$$\text{Tempo total} = [E + (I - 1)] \cdot C \quad 3.9$$

Esta equação permite calcular para um determinado conjunto de instruções tanto o tempo total de execução quanto o *throughput*.

A utilização do *pipeline* como medida de otimização é interessante, porém é preciso analisar os problemas que ocorrem com a utilização desta técnica e como evitá-los. O fluxo da execução de instruções em um *pipeline* considera que todas as instruções da sequência podem ser executadas sem problemas, porém, em alguns casos esta afirmação não é verdadeira.

Iniciamos a análise considerando três tipos diferentes de conflitos que podem ocorrer (chamados de *hazards*) (HENNESSY; PATTERSON, 2017):

- Conflitos estruturais – ocorrem quando a estrutura do processador não está preparada para lidar com 2 instruções precisando do mesmo recurso em estágios subsequentes do *pipeline*. Neste caso, uma instrução fica parada até que o recurso seja liberado.
- Dependência de dados – ao se executar um conjunto de instruções é possível que o resultado de uma instrução seja necessário para realizar a etapa de execução da segunda instrução. Neste caso, a instrução que depende do dado deve esperar o término da instrução anterior.
- Dependência de controle – os programas possuem desvios que mudam o fluxo das instruções. Portanto pode ser que a próxima instrução não seja a que será executada e sim outra instrução.

Além dos *hazards* é possível também que a memória não consiga suprir a demanda de dados em um *pipeline* muito rápido, fazendo com que o processador fique parado esperando a disponibilidade dos dados.

Todos estes casos geram o que chamamos de **bolha** no *pipeline*, ou seja, até que se resolvam os problemas de execução não é possível continuar a execução das instruções. Este tempo que o processador fica parado pode esvaziar o *pipeline* ocasionando um atraso na execução e a perda de toda a otimização de *throughput* que seria a vantagem do *pipeline*. Portanto é preciso evitar ao máximo que as bolhas ocorram.

Dentre as soluções possíveis estão:

- A reorganização de instruções do programa para que os recursos sejam utilizados por instruções não sequenciais.
- A disponibilidade de mais de uma estrutura no processador para que mais instruções possam ser executadas sem prejuízo. Existem processadores que possuem mais de uma unidade de memória, ou mais de uma ULA a fim de auxiliar nestes casos.
- Existe também a previsão de desvio que pode auxiliar no caso das dependências de controle. Uma determinada arquitetura pode ser projetada para sempre considerar que o desvio será tomado ou que

sempre será ignorado. Nestes casos, a próxima instrução será a do desvio ou a próxima, respectivamente. Se a previsão obtiver alta taxa de acertos é possível melhorar muito o desempenho do processador, porém se errar muito é melhor não utilizar pois o efeito é até pior visto que será realizado um processamento desnecessário.

Refletá

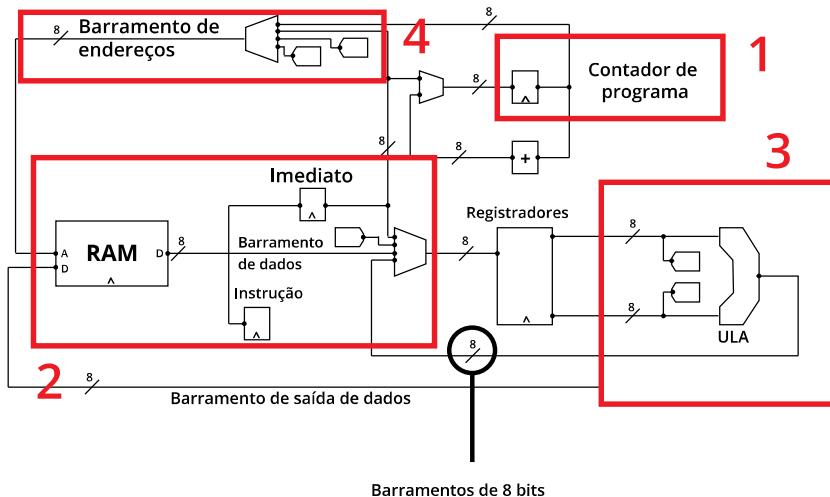
Ao definirmos o *pipeline*, apresentamos a presença do sinal de *enable* para permitir que as etapas da instrução sejam realizadas no tempo certo. Porém será que existe uma forma mais simples de definir o tempo de cada etapa do *pipeline*? Será que é possível adotar um padrão? Quais seriam as implicações de se adotar um padrão de duração de uma etapa? Para refletir sobre estes assuntos lembre-se que o projeto do processador deve sempre minimizar o tempo que o processador fica sem executar instruções.

Chegamos então a mais um final de seção. Foram apresentados aqui os conceitos de arquitetura multiciclo, otimização da execução de instruções com o *pipeline*, problemas deste método e suas soluções. Esta otimização é um conceito primitivo de paralelismo, visto que temos mais de uma instrução sendo executada ao mesmo tempo (ainda que cada uma em um estágio diferente). Este conceito foi primordial para o surgimento dos processadores multicore que temos atualmente, já que iniciou a linha de pesquisa de paralelização da execução de instruções. Agora você já tem conhecimento para solucionar seu próximo desafio! Boa sorte!

Sem medo de errar

Após ser apresentado aos conceitos de implementação multiciclo e *pipeline* é possível propor uma solução para a situação-problema. Como apresentado no início da seção, abaixo está o diagrama do processador da empresa Y e você deve analisá-lo a fim de descobrir como foi implementado e quais alternativas de otimização a empresa poderá abordar.

Figura 3.5 | Arquitetura do processador da empresa Y



Fonte: An Example (2010, [s.p.]).

Em um primeiro momento parece um processador totalmente diferente do que estudamos até aqui, porém é só procurar por estruturas conhecidas para identificar quais partes do processador estamos vendo. Identificando as partes presentes na Figura 3.5 temos:

O bloco 1 representa a fase de busca, identificável pela presença do contador de programa (PC), o bloco 2 seria a decodificação pois além de receber informações do bloco 1 está antes da unidade lógica aritmética (em inglês, ALU – *arithmetic logic unit*), o bloco 3 por conter a ALU é o bloco de execução e, por fim, o bloco 4 é o responsável pela configuração da gravação em memória. Sendo assim, o processador proposto possui as 4 etapas implementadas podendo conter, portanto, os 4 estágios básicos de *pipeline*. O diferencial principal deste processador é que todos os barramentos são de 8 bits indicando que este é o tamanho da palavra do processador. Com isso temos a análise esperada da arquitetura do processador da empresa Y.

Projeto de *pipeline*

Considere que você está projetando para a X o *pipeline* de um processador e tem que executar um conjunto de 6.999.993 instruções em 14ms já que se trata de um sistema que controla o freio de um veículo autônomo e que um tempo de resposta maior que 14ms poderia ocasionar um acidente. Quantos estágios de *pipeline* seu processador precisa ter para resolver este problema considerando que o *clock* tem período de 2ns?

Resolução da situação-problema

O resultado seria calculado da seguinte forma:

$$\begin{aligned}Tempototal &= [E + (I - 1)] \cdot C \\E &= \frac{tempototal}{C} - (I - 1) \\E &= \frac{14 \cdot 10^3}{2 \cdot 10^{-9}} - (6,999,993 - 1) \\E &= 7 \cdot 10^6 - 6,999,992 \\E &= 8\end{aligned}$$

Portanto seriam necessários 8 estágios de *pipeline*.

Faça valer a pena

1. Os processadores atuais são compostos de diversos núcleos que em seu projeto adotam a otimização chamada de *pipeline*. Este método de otimização prevê a divisão das instruções em etapas de execução e otimização do *hardware* para permitir que sejam executadas mais de uma instrução ao mesmo tempo, sendo que cada uma estará em um estágio diferente.

Considerando o *pipeline*, quais são as duas principais questões a serem abordadas no projeto de uma arquitetura que utiliza esse tipo de otimização?

- Número de registradores e ULA.
- Número de estágios de *pipeline* e número de bancos de memória.
- Número de estágios de *pipeline* e dependência de controle.

- d. Dependência de dados e unidade e exceções.
 - e. Tamanho do processador e consumo e energia.
- 2.** O início do desenvolvimento de processadores privilegiou as arquiteturas monociclo por serem mais simples de ser projetadas e mais adequadas às características da época. Com a evolução das técnicas de projeto e fabricação os processadores passaram ser projetados como multiciclo permitindo vários tipos de otimização como o *pipeline*.

Sobre a evolução da arquitetura de processadores, assinale a alternativa que apresenta a principal continuação após o *pipeline*.

- a. Arquiteturas 2 ciclos.
- b. Arquiteturas 8 ciclos.
- c. Arquiteturas MIPS.
- d. Arquiteturas mais de um núcleo.
- e. Arquiteturas monociclo.

- 3.** O *bypassing* (atalho) é uma solução implementada em *hardware* que prevê a criação de caminhos de dados ligando *hardware* para que seja possível obter informações fora do fluxo normal de processamento. Por exemplo, o mais comum é colocar um entre a saída da ULA e o registrador B, pois quando existir uma dependência de dados entre as instruções, a instrução da frente poderá acessar diretamente o resultado sem ter que esperar o mesmo ser salvo em bancos de registradores. Portanto, considerando este fato, é possível modificar a arquitetura do processador para englobar as modificações necessárias para o projeto.

Sobre as mudanças no caminho de dados, assinale a alternativa correta.

- a. Uma desvantagem de se alterar o *hardware* é o aumento significativo no custo.
- b. É possível otimizar a execução de mais de uma instrução ao otimizar o caminho de dados.
- c. Sempre existe melhora com a alteração no caminho de dados.
- d. Todas as modificações do caminho de dados resolvem problemas de execução.
- e. Alterar o caminho de dados é sempre a melhor saída para a solução de problemas com instruções.

Seção 4

Introdução à linguagem assembly

Diálogo aberto

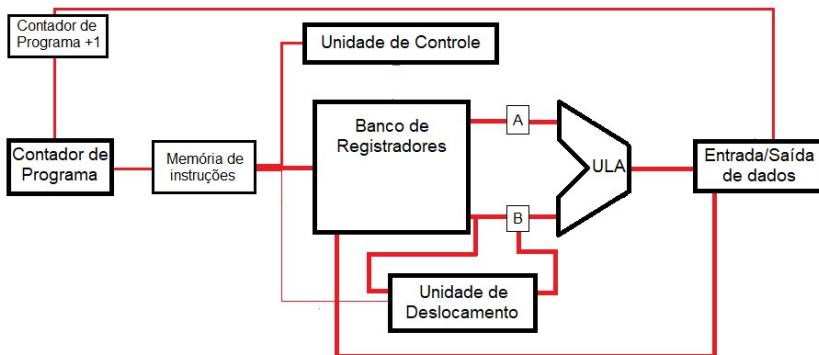
A ligação principal entre o programador que utiliza o hardware e a arquitetura do computador é o código do software. Não é possível programar bem um computador sem entender as implicações de se utilizar determinados comandos, sem se preocupar com a memória que será consumida por um determinado programa, sem entender as ferramentas que o hardware coloca à disposição de um programador.

Um bom profissional da área de computação necessita de conhecimentos de hardware para que possa desenvolver seu trabalho de forma otimizada, atingindo requisitos de projeto com alta performance. Como sabemos, um software pode ser compilado, interpretado e traduzido ou pode utilizar uma técnica híbrida para que as instruções sejam executadas (ASCENCIO; CAMPOS, 2012). Nesse processo, o código escrito em uma linguagem de programação de alto nível, como Java, C++, C#, será convertido em um código intermediário e, no caso dos programas compilados, convertido novamente para gerar os comandos que o processador é capaz de executar. Esse código intermediário é chamado de linguagem de montagem ou **assembly**. Essa linguagem foi criada com objetivo de facilitar o trabalho para os programadores, que anteriormente deveriam saber os códigos binários de cada instrução para poder programar (HENNESSY; PATTERSON, 2017).

Agora, você, como parte da equipe de desenvolvimento da X, teve uma nova atribuição. Considerando os problemas solucionados das arquiteturas anteriores, agora você foi designado para a função de elaboração dos mnemônicos da linguagem de montagem do processador que está sendo desenvolvido pela empresa. Nesta etapa você será apresentado ao diagrama do processador e aos tipos de instrução que ele irá suportar. Baseando-se nessas informações, e considerando as otimizações já discutidas em seções anteriores, você deverá criar as instruções a serem utilizadas para o desenvolvimento dos programas em linguagem de montagem e, futuramente, dos compiladores de linguagem de alto nível para a arquitetura do processador da X, mostrado na Figura 3.12.

O processador possui apenas instruções do tipo R e do tipo de desvio incondicional (j). Dessa forma, não será possível ter instruções imediatas. O diagrama do processador apresentado é mostrado a seguir:

Figura 3.12 | Diagrama para o desenvolvimento da linguagem



Fonte: elaborada pelo autor.

Considerando que o hardware está pronto, crie uma tabela com possíveis instruções – os ajustes a serem feitos na arquitetura serão realizados pelos outros idealizadores do projeto. Lembre-se de que buscamos uma arquitetura otimizada.

Lembre-se de verificar qual seria o conjunto mínimo de instruções necessárias para a construção de programas, pois, caso falte alguma instrução em seu conjunto, pode ser que não seja possível escrever programas corretamente utilizando sua linguagem de montagem. Preparado para mais este desafio? Boa sorte e bons estudos!

Não pode faltar

Esta seção apresenta os conceitos de linguagem de montagem, linguagem de máquina, teoria e simulação de instruções. Esses conteúdos são essenciais para o entendimento do efeito do projeto do conjunto de instruções para uma determinada arquitetura de computadores. Vamos relembrar alguns conceitos de programação antes de iniciar o nosso estudo propriamente dito. Para facilitar a abordagem e evitar que divaguemos sobre o assunto, vamos considerar o processo apenas de linguagens de programação compiladas.

O processo de geração de um programa que será executado em uma determinada linguagem, chamado de compilação, exige uma série de etapas para que seja capaz de gerar o melhor código possível dentro das capacidades do compilador. O compilador é um programa que possui como entrada um arquivo de determinada linguagem de programação e retorna um programa executável de uma determinada linguagem, para uma determinada

arquitetura. Existem dois grandes grupos de operações que são executadas: as operações de análise, que englobam operações específicas do compilador, e as operações de síntese, que são voltadas para a geração do código em si em uma arquitetura-alvo e englobam a geração de um código intermediário, a otimização desse código e a geração de um código-objeto, que será executado pelo processador. Vamos entender agora a relação entre essas áreas.

Estudamos, na seção anterior, o *pipeline* e seus *hazards*, sendo que uma das soluções para o problema da bolha (ou *stall*) é a reordenação de instruções para que a dependência de dados seja minimizada ou, no melhor caso, eliminada. Então, quem faz a reordenação destas instruções? Como é feita? Quando é feita?

Seria um erro pensar que atualmente, após anos de evolução das ferramentas computacionais, os programadores ainda programem em linguagem de máquina. Porém, seria um erro ainda maior considerar que tal conhecimento é desnecessário. Consideremos algumas situações para entender a aplicação deste conhecimento em situações reais atuais.

O entendimento da necessidade de programação em linguagem de máquina pode ser demonstrado em duas situações: a primeira é a do desenvolvimento de um software para gerência de relatórios em uma empresa (basicamente um sistema de informação gerencial); a segunda é a do desenvolvimento do software de controle para um veículo autônomo, ou seja, que é dirigido por um sistema embarcado de hardware e software (que “anda sozinho”, como os veículos de várias empresas atualmente).

No primeiro caso, as prioridades são a interface gráfica, o banco de dados que irá armazenar a informação, a disponibilidade do sistema (ou seja, esse sistema deve estar disponível 24 horas todos os dias da semana) e a segurança do sistema. Considerando as prioridades de um sistema como o proposto, é possível entender que são tarefas extremamente complexas, que envolvem a junção de tecnologias de hardware e software, como a programação da interface gráfica que se relaciona diretamente com instruções da placa de vídeo do computador. A menos que o programador utilize diversas ferramentas prontas (como softwares, bibliotecas de programação, códigos já desenvolvidos para partes básicas) esse trabalho irá tomar um tempo que inviabiliza sua execução. Pense, então, que em vez de utilizar uma linguagem de mais alto nível, como C++, Python ou Java, ele resolva utilizar uma linguagem de máquina e programar cada uma das instruções do processador. O processo fica inviável e torna-se desnecessário.

O segundo caso é mais específico e justifica a programação, que chamamos de baixo nível. Um veículo autônomo tem uma responsabilidade muito grande em não causar danos materiais ou físicos, já que é um programa que dirige o

carro. Se, por acaso, este programa demorar para responder, o veículo pode atropelar uma pessoa, bater em outro veículo, dentre outras situações que podem envolver risco de vida. Para que isso não ocorra, é preciso garantir que esse programa seja executado o mais rápido possível, da forma mais otimizada possível e sem erros. Além disso, o programa que aciona o freio é simples, tendo que analisar determinados comandos enviados pelo software que dirige o veículo e, de acordo com a necessidade, enviar comandos de acionamento do freio. Nesse caso, uma simples modificação no código intermediário gerado pode resultar em uma otimização do tempo de execução, tornando mais seguro e confiável o sistema.

Nos dois casos apresentados, não se considera construir o código completamente em linguagem de máquina, mas saber como analisar o código gerado e alterá-lo quando for necessário. Após a descrição do papel da linguagem de máquina na solução de problemas reais, podemos passar para a próxima etapa.

O conjunto de instruções de um processador é a base para a construção de sua arquitetura, como visto anteriormente. Não é possível adicionar uma instrução ao conjunto de instruções sem que exista o caminho de dados necessário para sua execução. Caso seja uma instrução básica necessária para o correto funcionamento dos programas, é necessário modificar o hardware. A vantagem de se definir tipos de instrução, como fizemos anteriormente, é que qualquer instrução que utilize um hardware existente e possa ser expressa em um dos formatos básicos pode ser adicionada ao conjunto sem problemas.

É importante discutirmos algumas questões de implementação de programas em linguagem de máquina. Para que seja possível entender como programar, considerando as estruturas básicas possíveis, vamos analisar cada um dos elementos de uma linguagem separadamente:

O primeiro conceito que aprendemos em uma linguagem de programação são as variáveis. No caso da linguagem de máquina, temos como armazenar informações diretamente em registradores (que são poucos e podem não ser suficientes) e na memória a partir de um endereço. Para acessar um registrador, utilizamos, em uma instrução, o cífrão \$ seguido do nome do registrador. No caso de um valor armazenado na memória, é necessário carregá-lo para um registrador, utilizando a instrução lw. O resultado de uma operação, que deve ser gravado na memória, tem que inicialmente ser salvo em um registrador e, em seguida, carregado para a memória com a instrução sw.

Agora, vamos explicar os tipos de instruções, iniciando com as instruções do tipo R, que possuem o formato: nome da instrução, registrador de destino, registrador de origem 1, registrador de origem 2. Os campos de

deslocamento (*shamt*) e de função, que vimos na Seção 3.2, são preenchidos automaticamente quando o código binário é gerado a partir da linguagem de máquina. Vejamos alguns exemplos de instruções e de sua utilização, como apresentados na Quadro 3.1.

Quadro 3.1 | Exemplos de instruções do tipo R

	Formato da Instrução	Operação Realizada
1	add \$rd, \$r1, r2	Soma o conteúdo de r1 com r2 e armazena em rd.
2	addu \$rd, \$r1, \$r2	Soma o conteúdo de r1 com r2, sem considerar o sinal dos números (só números positivos), e armazena em rd.
3	sub \$rd, \$r1, \$r2	Subtrai o conteúdo de r2 do conteúdo de r1 e armazena o resultado em rd.
4	or \$rd, \$r1, \$r2	Realiza a operação de “ou” lógico entre os bits armazenados em r1 e r2 e coloca o resultado em rd.
5	and \$rd, \$r1, \$r2	Realiza a operação de “e” lógico entre os bits armazenados em r1 e r2 e coloca o resultado em rd.
6	slt \$rd, \$r1, \$r2	Se o conteúdo de r1 for menor que o conteúdo de r2, armazena 1 em rd; caso contrário, armazena 0.
7	sll \$rd, \$r1, shamt	Armazena em rd os bits que estão armazenados em r1, deslocados para a esquerda na quantidade expressa por shamt (um inteiro).

Fonte: elaborado pelo autor.

As instruções do tipo R apresentam, em sua maioria, as operações que envolvem dados que estão armazenados nos registradores. Uma das exceções é apresentada na linha 7 do Quadro 3.1, que apresenta a instrução que desloca os bits, de modo que o terceiro operando será o valor do deslocamento e não mais um registrador de origem.

Refita

Após ser introduzido aos conceitos de linguagem de máquina, é preciso pensar em alguns pontos. Seria difícil programar utilizando bits? Ou seja, em vez de utilizar linguagem de máquina, seria possível utilizar sua tradução para binário diretamente? Qual seria o efeito nos programas criados? Essas perguntas foram as mesmas feitas pelas pessoas responsáveis por criar a linguagem de máquina de processadores, cuja palavra começava a ter um tamanho que geraria códigos grandes de alta complexidade. Tente responder a essas perguntas para sedimentar a noção sobre a importância da linguagem assembly.

Quando os dados ainda não estão carregados nos registradores ou quando se deseja fazer operações entre registradores e valores constantes, é necessário utilizar instruções do tipo I que recebem dois registradores (o registrador de resultado e um registrador de origem), seguido por uma constante. O Quadro 3.2 apresenta alguns exemplos.

Quadro 3.2 | Exemplo de instruções do tipo I

	Formato da Instrução	Operação Realizada
1	addi \$rd, \$r1, imediato	Armazena em rd o resultado da soma entre o conteúdo de r1 e o número colocado em imediato.
2	beq \$r1, \$r2, LABEL	Se r1 for igual a r2, o contador de programa é desviado para a linha onde se encontra o label indicado.
3	bne \$r1, \$r2, LABEL	Se r1 for diferente de r2, o contador de programa é desviado para a linha onde se encontra o label indicado.
4	slti \$rd, \$r1, imediato	Se o conteúdo de r1 for menor que o número colocado como imediato, o resultado em rd é 1; caso contrário, é 0.
5	lw \$rd, imediato(\$r1)	O número imediato colocado é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato (na memória de dados) de um dado que é salvo em rd.
6	sw \$rd, imediato(\$r1)	O número imediato colocado é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato de um endereço (na memória de dados), onde vai ser salvo o conteúdo de rd.

Fonte: elaborado pelo autor.

Em linguagem de programação, trabalhamos com as condicionais (if-else). Nesse caso, as instruções beq e bne são utilizadas. É preciso cuidado para manipular corretamente os valores, já que essas instruções apenas retornam 0 ou 1 para valores iguais ou diferentes nos registradores. Então deve ser pensada uma maneira de transformar uma expressão de teste de um if em um cálculo cujo resultado é comparado com o conteúdo de outro registrador.

E, por fim, as instruções de desvio incondicional são apresentadas no Quadro 3.3.

Quadro 3.3 | Exemplo de instruções do tipo J

	Formato da Instrução	Operação Realizada
1	j LABEL	O contador de programa é desviado para a linha de código onde se encontra o label.
2	jal LABEL	Mesma operação do anterior, com a diferença que, neste caso, o endereço da próxima instrução sem o desvio é armazenado no registrador reservado \$ra.

Fonte: elaborado pelo autor.

Assimile

As funções podem ser implementadas utilizando labels e instruções do tipo j. Porém, existe uma diferença, a chamada ao label da função deve ser feita com a instrução jal para que o endereço do programa seja salvo no registrador \$ra. Também, ao final da execução, deve-se utilizar o comando jr \$ra. A instrução jr desvia para o endereço contido em um registrador e, quando se utiliza o registrador \$ra, é devolvido ao contador de programa a próxima instrução do programa principal.

Algumas instruções apresentadas nos quadros anteriores são intuitivas, outras nem tanto. Vamos analisar alguns casos específicos. No Quadro 3.1, a instrução slt é utilizada para verificar se o conteúdo de um registrador é menor que o conteúdo de outro registrador. Sendo assim, caso o conteúdo for menor, o registrador de resultado recebe 1, caso contrário, recebe 0. A instrução sll é utilizada para deslocar os bits de um registrador, sendo assim, quando temos no registrador \$r1 o valor 000011100001111000011110000 1111 e executamos o comando **sll \$rd, \$r1, 3**, o resultado no registrador \$rd será 01111000011110000111100001111000. Repare que, nessa operação, os bits da esquerda são descartados e os bits da direita são completados com 0, como demonstrado a seguir:

Valor original - 0000111000011110000111100001111

Deslocamento - 01111000011110000111100001111000

As instruções do Quadro 3.2 possuem campos que são utilizados ora como LABEL, ora como imediato. Um LABEL é um rótulo colocado ao longo do programa em linguagem de máquina e um imediato é um número inteiro que será utilizado no cálculo. As instruções beq e bne são instruções de desvio de execução. Ao serem executadas, irão desviar a execução

do código caso o conteúdo dos registradores seja igual (beq) ou diferente (bne). O desvio ocorre para a linha que está sendo indicada pelo LABEL; esse conceito ficará mais simples assim que apresentarmos um exemplo. A instrução slti é uma variação da instrução slt do Quadro 3.1, que, quando utilizada, compara o conteúdo do registrador de origem com um número imediato e não mais com o conteúdo de um segundo registrador de origem.

Duas instruções importantes para qualquer programa em linguagem de máquina são apresentadas no final do Quadro 3.2. A instrução lw carrega uma informação da memória para um registrador. Esse procedimento é feito da seguinte maneira: o programador passará para a instrução o registrador de destino do dado, um endereço de base e um deslocamento; o endereço de base é o endereço em que começa a área de memória de dados e é definido pelo programador entre os endereços permitidos; o deslocamento refere-se a quantos espaços de memória devem ser considerados, desde a base, para chegar no dado. Um exemplo: considere que a memória se inicia no endereço 0 e o dado está no endereço 5, portanto 0 seria a base e 5 seria o deslocamento. Porém, repare que, no Quadro 3.2, é indicado que o valor de deslocamento é multiplicado por n , isso ocorre pois a memória nos computadores é dividida em espaços de k bits e, na hora de gerar o código binário, é feito o seguinte cálculo (Equação 3.10):

$$n = \frac{n^{\text{º}} \text{bitspalavra}}{k} \quad 3.10$$

No caso do MIPS, que estamos usando como exemplo, a palavra possui 32 bits e a memória é dividida em espaços de 8 bits, portanto n será 4. O mesmo ocorre com a instrução sw, porém, em vez de salvar no registrador, estamos armazenando o valor na memória.

As instruções apresentadas no Quadro 3.3 são de desvio incondicional, ou seja, quando executadas irão desviar o fluxo do programa. A diferença é que, no caso da instrução j, o desvio é feito e nada é armazenado; já na instrução jal, o endereço da próxima instrução é armazenado em um registrador especial \$ra. Esse procedimento é utilizado para criar subprogramas dentro de um programa em linguagem de máquina. Vejamos um exemplo que utiliza algumas destas instruções.

Exemplificando

Neste exemplo vamos considerar a seguinte situação: temos três vetores, a , b e c (estruturas que armazenam números em lugares sequenciais da memória), de 100 elementos cada e devemos desenvolver um programa que some cada elemento do vetor a e b e coloque o resultado no vetor c :

```
1. int i = 0; //declara um inteiro começando com 0
2. para ( i = 0; i < 100; i++){  
//o comando para executar as operações dentro das chaves por um  
número x de vezes, sendo que esse número vai do valor inicial de i (que  
definimos como 0) até a expressão não ser mais verdadeira (ou seja, no  
nosso caso quando i for menor que 100).
```

```
3. c[i] = a[i] + b[i];
```

//a posição i no vetor c deve receber a soma dos números do vetor a e b na posição i também.

```
4. }
```

Porém, como é necessário otimizar o código, foi solicitado que o código fosse feito em linguagem assembly do MIPS. Vejamos como você poderia resolver esse problema:

- Inicialmente vamos considerar que os endereços de base dos vetores *a*, *b* e *c* estão respectivamente em \$a0, \$a1 e \$a2.
- Será necessário definir a posição final para que o programa pare de processar os dados na posição de memória correta.
- Lembrar que, sempre que movemos a posição do vetor no MIPS, temos que movimentar por quatro blocos de 8 bits.
- Comentários em linguagem de máquina iniciam-se com # e são considerados até o final da linha.

Sendo assim, uma das possíveis soluções seria:

1. addi \$t6, \$a0, 400 #encontrando o final do vetor
 2. LOOP: beq \$a0, \$t6, FIM #caso o endereço final seja atingido,
terminar
 3. lw \$t0, 0(\$a0) #carrega o número da posição a[i]
 4. lw \$t1, 0(\$a1) #carrega o número da posição b[i]
 5. add \$t0, \$t0, \$t1 #coloca a soma em um registrador temporário
 6. sw \$t0, 0(\$a2) #carrega o resultado na posição c[i]
- #lembra que neste caso i era 0 porque pegamos o endereço de base
#de cada um dos vetores, agora precisamos fazer i++, porém isso
#será feito posição por posição
7. addi \$a0, \$a0, 4 #movimenta o vetor a 1 posição

8. addi \$a1, \$a1, 4 #movimenta o vetor b 1 posição
9. addi \$a2, \$a2, 4 #movimenta o vetor c 1 posição
10. j LOOP #continua o loop

11. FIM:

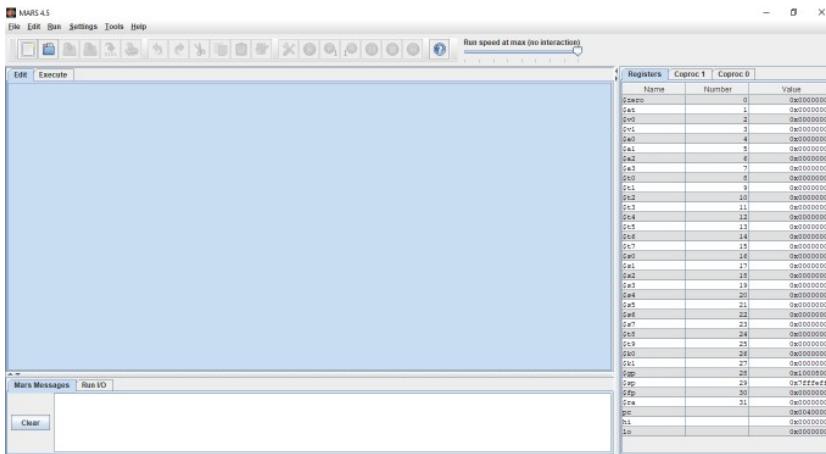
Veja que, ao implementar em linguagem de máquina um programa, é preciso considerar diversos pontos e pensar de forma diferente. Os *labels* loop e fim são utilizados como forma de controlar a execução do loop. Com cuidado analise o programa apresentado como solução para o problema e tente entender melhor o uso de cada instrução.

Simulando instruções

Para encerrar, vamos fazer um exemplo de simulação de um código simples no simulador MARS (*MIPS Assembler and Runtime Simulator*). Para iniciar a simulação, é necessário apenas visitar o site *Missouri State University* e procurar pelo download da versão 4.5 (MISSOURI..., 2014).

Assim que for feito o download, é só executar o programa, que é um .jar, que o simulador abrirá em sua tela, como na Figura 3.13.

Figura 3.13 | Simulador MARS – Tela inicial



Fonte: captura de tela do simulador MARS.

Agora vamos executar o código de um condicional simples e verificar como o simulador funciona. Inicialmente vamos pensar no código da parte que desejamos implementar em linguagem de máquina:

```
if (x == y) {  
    a = 1;  
}else{  
    a = 2;  
}
```

Inicialmente é necessário atribuir registradores às variáveis. Vamos então definir que a variável **x** está no registrador **s1**, a variável **y** está no registrador **s2** e que a variável **a** está no registrador **s3**. Lembrando que poderiam ser utilizados registradores **t** também.

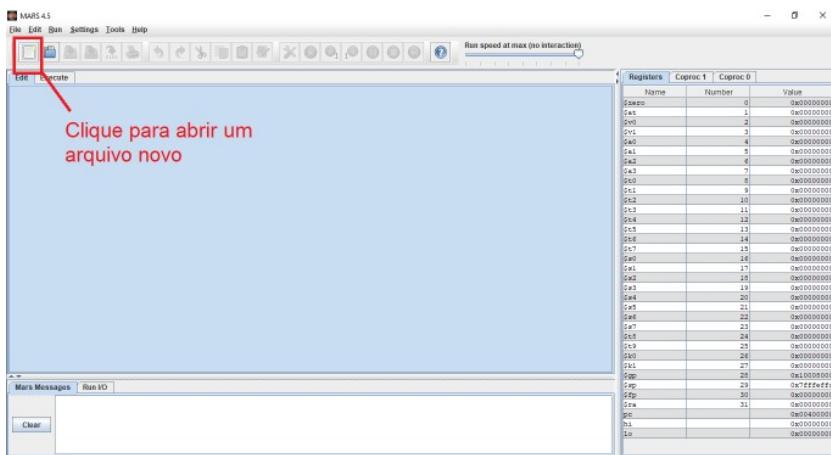
O resultado do código é o seguinte:

```
.text  
beq $s1, $s2, IGUAL #se x for igual a y pula para IGUAL  
bne $s1, $s2, DIFF #se x for diferente de y pula para DIFF  
IGUAL: addi $s3, $zero, 1 # a recebe 0 + 1  
j FIM #pula para FIM  
DIFF: addi $s3, $zero, 1 # a recebe 0 + 2  
FIM:
```

Vamos analisar a solução. A diretiva **.text** é necessária na primeira linha do simulador para digitar o código. Em seguida as duas instruções de teste são feitas, uma para igual e outra para diferente. Cada uma delas, caso verdadeiro, redireciona para seu respectivo Label. No Label **IGUAL**, somamos 0 com 1 e armazenamos no registrador que representa a variável **a**; esse procedimento é feito, pois, como não sabemos o que temos no registrador, garantimos, dessa maneira, que ele irá armazenar 1. O mesmo procedimento é feito para o Label **DIFF**, que soma 0 e 2 e armazena no registrador que representa **a**. Entre os dois Labels é necessário pular para o final do programa, porque, como a execução é sequencial, se não fizermos isso, será executada a operação que está na linha do Label **DIFF** também.

Para simular o programa, é necessário criar um arquivo novo no simulador clicando no ícone indicado na Figura 3.14.

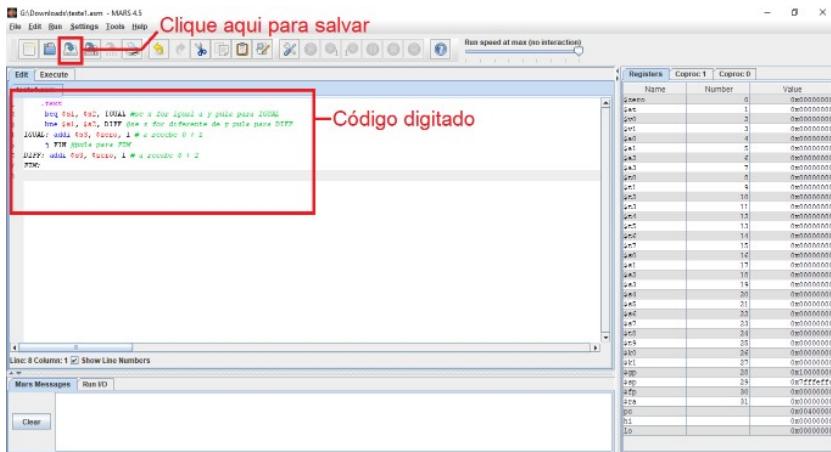
Figura 3.14 | Passos para simulação no MARS



Fonte: captura de tela do simulador MARS.

Após clicar, na interface que aparecer, digite as instruções do código de máquina que elaboramos. Em seguida salve o programa clicando no ícone indicado na Figura 3.15.

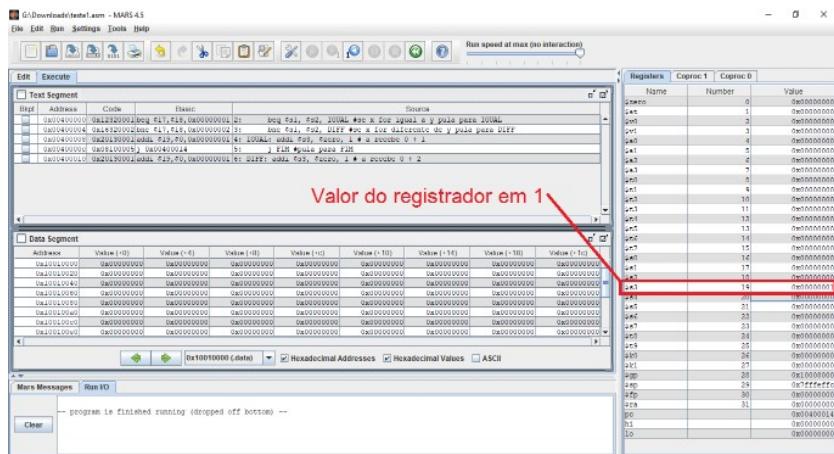
Figura 3.15 | Inserindo o código no MARS



Fonte: captura de tela do simulador MARS.

Por fim, faça a montagem do código e sua execução clicando em Run->Assemble e execute o código clicando em Run-> Go. Depois da execução, espera-se que seja armazenado no registrador que indicamos para representar a variável *a* o valor 1, já que, como não iniciamos valores para x e y, eles são iguais. Após a execução, é possível verificar, no local indicado, que o registrador s3 teve seu valor alterado para o hexadecimal 0x00000001, que representa o valor decimal 1, mostrado na Figura 3.16.

Figura 3.16 | Resultado da simulação no MARS



Fonte: captura de tela do simulador MARS.

Pronto, você acabou de simular a execução de um código de montagem para o processador MIPS!

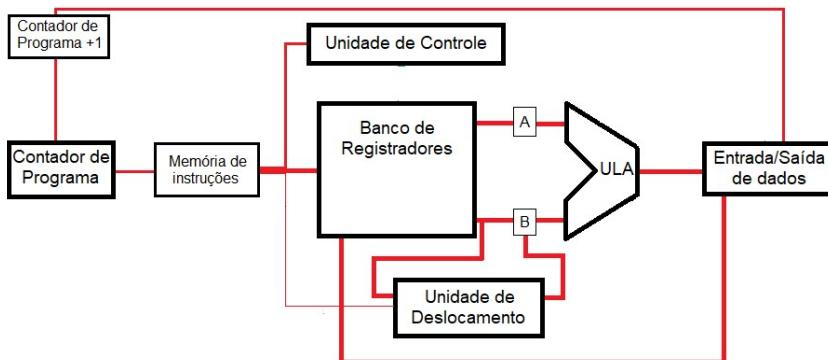
Esta seção apresentou os conceitos de linguagem de montagem (que é a linguagem que utilizamos para programar o processador com mnemônicos para facilitar o entendimento), máquina (que são as instruções em formato binário e são processadas pelo processador) e como organizar instruções para realizar as operações desejadas. Um paralelo com a linguagens de programação foi feito para facilitar o entendimento, e todas as características básicas de uma linguagem foram discutidas com relação à sua implementação em linguagem de máquina. Após todo esse exercício, é possível entender que o conjunto de instruções é parte importante e deve ser considerado totalmente na construção do hardware, pois, caso haja negligência, é possível que o tempo de execução de um conjunto de instruções seja comprometido por uma arquitetura mal planejada.

Sem medo de errar

Após adquirir os conhecimentos necessários, é possível retornar ao problema que apresentamos no início da seção. Agora, você, como parte da equipe de desenvolvimento da X, teve uma nova atribuição: foi designado para a função de elaboração dos mnemônicos da linguagem de montagem do processador que está sendo desenvolvido pela empresa. Nessa etapa você será apresentado ao diagrama do processador e aos tipos de instrução que ele irá suportar. Baseando-se nessas informações e considerando as otimizações já discutidas em seções anteriores, você deverá criar as instruções que serão utilizadas para o desenvolvimento dos programas em linguagem de montagem e, futuramente, dos compiladores de linguagem de alto nível para a arquitetura do processador da X.

Este problema envolve basicamente todo o conteúdo visto até o momento em arquitetura de computadores. Vamos aos dados: o processador possui apenas instruções do tipo R e do tipo de desvio incondicional (j). Dessa forma, não será possível ter instruções imediatas. O diagrama do processador apresentado é este:

Figura 3.12 | Diagrama para o desenvolvimento da linguagem



Fonte: elaborada pelo autor.

Inicialmente esta situação-problema pode parecer complexa, porém desenvolver o formato e os mnemônicos da linguagem não é o problema. Considerando que o hardware está pronto, é possível criar uma tabela com possíveis instruções e com os ajustes a serem feitos na arquitetura, que serão realizados pelos outros idealizadores do projeto. Lembre-se de que buscamos uma arquitetura otimizada.

Quadro 3.4 | Solução para a situação-problema

	Formato da Instrução	Operação Realizada
1	add \$r1, r2	R1 recebe a soma entre o que é armazenado em r1 e r2.
2	mult \$r1,r2	R1 recebe a multiplicação entre o que é armazenado em r1 e r2.
3	or \$r1, \$r2	R1 recebe o “ou” lógico entre os bits que estão armazenados em r1 e r2.
4	and \$r1, \$r2	R1 recebe o “e” lógico entre os bits que estão armazenados em r1 e r2.
5	inv \$r1	r1 recebe o inverso do valor que está armazenado em r1, ou seja, -r1.
6	bnr \$r1,\$r2,\$r3	Se o conteúdo de r1 for igual ao conteúdo de r2, desvia-se o contador de programa para a posição de memória de programa indicada em r3.
7	bqr \$r1,\$r2, \$r3	Se conteúdo de r1 for diferente do conteúdo de r2, desvia-se o contador de programa para a posição de memória de programa indicada em r3.
8	jr \$r1	O contador de programa é levado diretamente ao endereço armazenado em r1.
9	jrl \$r1	O contador de programa é levado diretamente ao endereço armazenado em r1 e salva seu estado atual no registrador \$ra.
10	lw \$rd, \$r3, \$r1	O endereço armazenado em r3 é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato (na memória de dados) de um dado que é salvo em rd.
11	sw \$rd, \$r3, \$r1	O endereço armazenado em r3 é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato de um endereço (na memória de dados) onde vai ser salvo o conteúdo de rd.

Fonte: elaborado pelo autor.

Essas instruções apresentam um número pequeno de operandos e não utilizam nenhuma parte que receba um imediato ou label. O endereço sempre deve estar em um registrador, pois assim é possível acessar a memória utilizando esse endereço do registrador como base. Portanto, esse conjunto de instruções é otimizado para uma arquitetura simples por não utilizar muitas informações em cada instrução e também elimina a necessidade de o hardware tratar imediatos. Outra questão é que permite que programas sejam desenvolvidos nessa linguagem de máquina, utilizando todos os conceitos básicos apresentados nas seções desta unidade.

Resolvendo problemas de *pipeline*

Uma das instruções mais utilizadas em programas de linguagem de montagem é a instrução que carrega dados da memória nos registradores, a instrução `lw`. Esta demora normalmente quatro estágios para que o dado esteja disponível para a gravação, já que todo o cálculo é feito para obtenção do endereço do dado buscado na memória. Esse fato pode gerar alguns problemas ao se desenvolver códigos para processadores com pipeline.

Após desenvolver um processador, os técnicos da X encontraram um código que gerava bolhas no pipeline e você foi selecionado para resolver o problema. Segue o código para que seja identificado o erro e resolvido o problema.

1. `lw $t0, 0($a0)`
2. `lw $t1, 4($a0)`
3. `add $t3, $t0, $t1`
4. `sw $t3, 12($a0)`
5. `lw $t4, 8($a0)`
6. `add $t7, $t0, $t4`
7. `sw $t5, 16($a0)`

Resolução da situação-problema

Nesse código não há nada de errado no sentido da escolha e utilização das instruções; o problema é que se considera uma arquitetura com pipeline. Aqui é necessário lembrar que a instrução `Iw` apenas libera o dado a partir do quarto ciclo de execução e isso faz com que, entre a instrução 2 e 3, forme-se uma bolha, já que os dados não estarão disponíveis, e também que se forme outra bolha entre as instruções 5 e 6 pelo mesmo motivo. A solução neste caso seria reordenar as instruções de modo que, enquanto dados são carregados, outras operações são feitas. O resultado seria:

1. `lw $t0, 0($a0)`
2. `lw $t1, 4($a0)`
3. `lw $t4, 8($a0)`

4. add \$t3, \$t0, \$t1
5. sw \$t3, 12(\$a0)
6. add \$t7, \$t0, \$t4
7. sw \$t5, 16(\$a0)

Neste caso não se espera nada, porque, quando é realizada a primeira operação de soma (instrução 4), os dados já estão disponíveis, assim como na segunda operação de soma da instrução 6.

Este caso é interessante por demonstrar que não é possível solucionar problemas em programas escritos em linguagem de máquina sem conhecer o hardware que está executando essas instruções.

Faça valer a pena

1. A arquitetura do processador MIPS, apresentada por Hennessy e Patterson (2017) e utilizada como base para os estudos desta seção, apresenta três tipos básicos de instrução, com algumas variações possíveis, para implementação dos programas em linguagem de máquina.

Considerando as instruções abaixo, assinale a alternativa que apresenta uma instrução com estrutura de tipo diferente do que era esperado.

- a. sub \$t0, \$t1,\$t2.
- b. add \$a0, \$t1,\$t0.
- c. lw \$r3, 20(\$t0).
- d. addu \$a0, \$t0, \$t2.
- e. addi \$t3, \$t1, \$t2.

2. Como visto ao longo da seção, o conjunto de instruções é parte do projeto da arquitetura do processador. A escolha de como as instruções serão executadas e como o hardware será construído é importante, pois impacta diretamente no desempenho da arquitetura.

Dentre as otimizações apresentadas, análise as afirmativas que apresentam as modificações possíveis de se implementar em linguagem de máquina.

- I. add \$t0, \$t1.
- II. add \$t0.
- III. slti \$t0, 20.

IV. or \$t0, T1.

É correto o que se afirma em:

- a. I e II, apenas.
- b. I, II e III, apenas.
- c. II, III e IV, apenas.
- d. III e IV, apenas.
- e. I, II, III e IV.

3. As bolhas no *pipeline (stall)* são formadas quando uma instrução precisa aguardar o encerramento de outra instrução para ser executada. Isso ocorre, por exemplo, quando uma instrução de soma precisa utilizar um operando que ainda não está carregado nos registradores. Uma das formas de se identificar esse problema é quando o mesmo registrador é utilizado por duas instruções em sequência, como a seguir:

add \$r1, \$r2, \$r3

sub \$r4, \$r1, \$r5

Quando essa situação ocorre, dizemos que há um problema de **dependência de dados** entre as linhas de código envolvidas. Uma das formas de evitar que existam problemas de bolha no pipeline (*stall*) é reordenar as instruções de um programa, afim de dar tempo ao processador para resolver as operações e sanar as dependências de dados. No caso apresentado, instruções que não utilizam o valor do registrador \$r1 poderiam ser colocadas entre as duas instruções apresentadas. Esta técnica é normalmente aplicada pelos compiladores na geração do código intermediário.

Considere o programa a seguir:

1. sub \$r2, \$r1, \$r3
2. and \$r12, \$r2, \$r5
3. or \$r13, \$r6, \$r2
4. add \$r14, \$r2, \$r2
5. sw \$r15, 100(\$r2)

Este programa contém dependências de dados em relação a um determinado registrador.

Assinale a alternativa que apresenta corretamente: os números das linhas onde existe dependência de dados e entre quais linhas seria possível atrasar a execução para que a operação que está causando a dependência de dados fosse resolvida, respectivamente.

- a. 1, 3 e 5 e entre as linhas 2 e 3.
- b. 3, 4 e 5 e entre as linhas 1 e 2.
- c. 2, 3 e 4 e entre as linhas 3 e 4.
- d. 1, 2, 3 e 4 e entre as linhas 3 e 4.
- e. 2, 3, 4 e 5 e entre as linhas 1 e 2.

Referências

AN EXAMPLE Hardwired CPU. 2020. Disponível em: <https://minnie.tuhs.org/CompArch/Tutes/week03.html>. Acesso em: 6 mar. 2020.

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação de computadores:** algoritmos, Pascal, C, C++ e Java. 3. ed. São Paulo: Pearson Educacional, 2012. 584 p.

HENNESSY, J. L.; A PATTERSON, D. **Organização e projeto de computadores:** a interface hardware/software. 5. ed. São Paulo: Elsevier, 2017. 680 p.

HENNESSY, J. L.; A PATTERSON, D. **Arquitetura de computadores:** uma abordagem quantitativa. 6. ed. Santos: Gen | Ltc, 2019. 816 p.

IFRAH, G. **História universal dos algarismos:** a inteligência dos homens contada pelos números e pelo cálculo. Rio de Janeiro: Nova Fronteira, 1997. 2 v.

MISSOURI STATE UNIVERSITY. **MARS (MIPS Assembler and Runtime Simulator).** An IDE for MIPS Assembly Language Programming. 2014. Disponível em: <http://courses.missouristate.edu/KenVollmar/mars/>. Acesso em: 27 fev. 2020.

ROJAS, R. Konrad Zuse's legacy: the architecture of the Z1 and Z3. In: IEEE Annals Of The History Of Computing. California, p. 5-16, jul. 1997.

SHANNON, C. E. A symbolic analysis of relay and switching circuits. **Electrical Engineering**, [s.l.], v. 57, n. 12, p.713-723, dez. 1938. Institute of Electrical and Electronics Engineers (IEEE). Disponível em: <http://dx.doi.org/10.1109/ee.1938.6431064>. Acesso em: 24 fev. 2020.

STALLINGS, W. **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson Universidades, 2009. 640 p. Disponível em: <https://plataforma.bvirtual.com.br/Leitor/Publicacao/1247/pdf>. Acesso em: 4 fev. 2020.

TOCCI, R.; WIDMER, N.; MOSS, G. **Sistemas digitais:** princípios e aplicações. 12. ed. São Paulo: Pearson Universidades, 2019. 1056 p. Disponível em: <https://plataforma.bvirtual.com.br/Leitor/Publicacao/168497/pdf>. Acesso em: 29 jan. 2020.

TOCCI, R.; WIDMER, N.; MOSS, G. **Sistemas digitais:** princípios e aplicações. 12. ed. São Paulo: Pearson, 2019. 1.056 p.

WAZLAWICK, R. S. **História da computação.** São Paulo: GEN: LTC, 2016. 563 p.

Unidade 4

Michel Bernardo Fernandes da Silva

Arquiteturas de alto desempenho

Convite ao estudo

Desde os primeiros computadores, cientistas e engenheiros dos fabricantes de computadores e processadores, centros de pesquisa e universidades estudam métodos e técnicas para melhoria do desempenho e da capacidade de sistemas computacionais. Estas pesquisas se deram tanto no campo de software, com desenvolvimento de sistemas operacionais, quanto no campo de hardware, pela evolução da arquitetura dos processadores.

Mesmo que você seja somente um usuário final do sistema computacional e tenha contato mais direto com o software, é importante conhecer a arquitetura do computador, que gera uma série de implicações para as aplicações.

Os resultados das pesquisas e desenvolvimentos mostram a evolução do hardware: a velocidade de processamento da frequência do clock (velocidade em ciclos por segundo – hertz – para executar as funções) do computador passou de poucos MHz em 1980 para GHz na década de 2010, além da criação de novas tecnologias de processamento como pipelining e processamento paralelo, entre outras.

Os primeiros sistemas de computadores precisavam funcionar por muitos anos, já que demandavam alto investimento. Entretanto, esses sistemas apresentavam problemas de confiabilidade, pois além de falhas no hardware, o software poderia apresentar falhas (*bugs*) na operação, sendo necessária a realização de manutenções neste sistema.

Em meados da década de 50, começaram a ser desenvolvidas linguagens de alto nível para que o programador elaborasse os algoritmos de forma mais enxuta. Estes poderiam utilizar linguagem estruturada ou linguagem orientada a objetos.

Entretanto, existe uma diferença muito relevante entre a linguagem de máquina, entendida pelo computador, e a linguagem de programação. Assim, a diferença semântica que ocorre entre as operações fornecidas em linguagens de alto nível e as linguagens oferecidas na arquitetura do computador era uma questão a ser resolvida.

Inicialmente foram utilizados conjuntos extensos de instruções, dezenas de modos de endereçamento. Para representar uma instrução em linguagem de máquina, ela era traduzida por várias instruções em linguagens de alto nível, com objetivo de reduzir essa diferença semântica. Este tipo de arquitetura de computadores foi denominado CISC – *Complex Instruction Set Computers*, ou computadores com conjunto de instruções complexas.

Entretanto, com o passar do tempo, foram realizados avanços na área de arquitetura de processadores. Verificou-se que, com um conjunto menor de instruções, seria possível melhorar o desempenho do processador com uma arquitetura denominada RISC - *Reduced Instruction Set Computers* (computadores com conjunto reduzido de instruções). Na Seção 1 desta unidade serão apresentadas as arquiteturas CISC e RISC, com exemplos de instruções de ambas.

A Seção 2 trata de arquitetura de processamento paralelo e analisa cada tipo de classificação como SISD (única instrução, único dado) e SIMD (única instrução, múltiplos dados) e GPUs (unidades de processamento gráfico); MISD (múltiplas instruções, único dado) e MIMD (múltiplas instruções múltiplos dados), segundo a classificação proposta por Flynn.

A Seção 3 estabelece o conceito de threads de processos, explicando o funcionamento de processadores monothread – somente 1 thread por vez –, e multithreads – vários threads simultâneos –, exemplificando os processadores multithreads.

Por fim, a Seção 4 explica o funcionamento das arquiteturas manycore e multicore, isto é, dual core, quad core ou octa core, entre outros, exemplificando essa arquitetura com processadores comerciais.

Estes conhecimentos se tornam úteis na elaboração de projetos de novos processadores. Nesses projetos devem ser analisados elementos como conjuntos de instruções, arquitetura paralela, processamento multithread e chip multiprocessado.

Como profissional área da computação, você terá muitos desafios pela frente. Portanto, é chegada a hora de se iniciar esse novo desafio profissional! Bons estudos!

Seção 1

Introdução a arquiteturas de alto desempenho

Diálogo aberto

Para atender aos moradores de uma casa é necessário organizar, com uma arquitetura, espaços para criar ambientes a fim de abrigar os diversos tipos de atividade humanas. De forma similar, os processadores também precisam estar organizados de forma a atender melhor aos seus usuários. Em outras palavras, dependendo da necessidade do usuário, existe uma arquitetura que o atenda de forma efetiva. Uma das características que o usuário, ao adquirir o seu computador, está sempre atento, está relacionada à velocidade do processador. Nesse sentido, a forma como o processador é projetado influencia essa velocidade.

Você foi contratado como um dos analistas do projeto do desenvolvimento da arquitetura de um novo processador de uma importante fabricante internacional de processadores, conhecida pela qualidade e pela inovação. Esse novo processador deve ser projetado para oferecer um desempenho superior ao dos concorrentes.

Existem diversas questões que implicam decisões impactantes para o futuro produto que exigem um compromisso (ou *trade-off*, em inglês) no desempenho desse processador.

Uma das primeiras decisões que a equipe de projeto deve tomar é em relação à quantidade de instruções. Ela deve escolher entre um número maior de instruções com instruções mais complexas ou um número menor de instruções com sintaxe simplificada.

Haverá uma reunião de trabalho entre toda a equipe de trabalho e o gerente, e você ficou responsável de trazer diversos elementos para enriquecer a discussão. Ao final dessa reunião vocês chegarão a um consenso sobre a quantidade de instruções que o processador suportará, e a complexidade de cada instrução.

O projeto já está atrasado em relação ao cronograma, e caso a decisão seja equivocada trará consequências nas próximas etapas de definição da arquitetura. Além disso, o principal concorrente da sua empresa também está desenvolvendo um projeto similar.

Você já participou de reuniões anteriores que não chegaram a um consenso sobre as características desejadas. Foi debatido o tamanho da instrução: se deveria ter a mesma quantidade de elementos na instrução ou

se quantidade desses elementos poderia variar. A conclusão dessa discussão contribuirá para definição da quantidade e complexidade das instruções com que o processador vai trabalhar.

Como material para essa reunião, você deve levantar diversas recomendações para o projeto com base nas arquiteturas CISC e RISC, indicando quais características seriam mais adequadas para o projeto. Para isso, prepare um relatório com as características das arquiteturas CISC, RISC e as desejadas no projeto e verifique qual arquitetura está mais próxima do desejado.

O futuro desse importante projeto está em suas mãos.

Portanto, mãos à obra!

Não pode faltar

Características de arquiteturas – Histórico

O estudo da arquitetura dos computadores pode garantir uma melhor compreensão sobre o funcionamento dos processadores, desde os primeiros sistemas computacionais.

O surgimento de linguagens de alto nível – a primeira a surgir foi o Fortran, criada em meados da década de 1950 – trouxe a evolução dos conceitos de programação como linguagens estruturadas e, posteriormente, linguagens orientadas a objetos. Nessas linguagens de programação, os comandos e as estruturas existentes foram criados para atender ao raciocínio do ser humano e à necessidade de redução de vida dos sistemas de informações e aplicações. Com isso, os comandos e estruturas se afastam cada vez mais da linguagem de máquina (MONTEIRO, 2010). Assim, surge uma separação acentuada entre os comandos de linguagens de alto nível e as instruções de máquina, ou de baixo nível. Dessa forma, um único comando de uma linguagem de alto nível poderia gerar muitas instruções de máquina no código-objeto correspondente, dando bastante trabalho ao programa compilador.

Tal diferença de complexidade entre os comandos de linguagens de alto nível e de instruções de máquina foi estudada nas áreas de pesquisa como *semantic gap*, ou espaço semântico, e forçou alguns fabricantes de processadores a tomarem decisões estratégicas em acordo com desenvolvedores de compiladores, os programas responsáveis por converter linguagens de alto nível para linguagens de baixo nível. Os fabricantes de processadores da época eram Intel, IBM, DEC – que anos depois foi comprada pela Compaq e posteriormente absorvida pela HP – e a AMD.

Alguns fabricantes, como DEC e IBM, produziam os computadores de grande porte, mainframes predominantes na época, e com o tempo foram aumentando a quantidade de instruções de máquina nos seus processadores. No início da década de 1980, essas arquiteturas foram denominadas *Complex Instruction Set Computers* (CISC) ou computadores com conjunto de instruções complexas.

Em alguns processadores, a quantidade ultrapassava 300 instruções e a havia uma grande quantidade de modos de endereçamento. Nessa época, o objetivo era facilitar a construção e o serviço dos compiladores, pois era necessária a construção de compiladores e ligadores (linkeditores) bastante sofisticados e capazes de realizar o mapeamento de comandos de alto nível em muitas instruções primitivas da forma correta. Com o avanço da tecnologia, no final da década de 1970 já existia uma deficiência entre o que as máquinas podiam fazer e o que as linguagens de alto nível exigiam.

No início da década de 1970 e com o intuito de simplificar os compiladores e melhorar o desempenho partindo-se do desenvolvimento de computadores, foi desenvolvida a arquitetura de computadores com conjunto reduzido de instruções ou *Reduced Instruction Set Computers* (RISC) (MONTEIRO, 2010).

No início da década de 1980, os desenvolvedores de hardware iniciaram a reconsiderar diversos princípios de arquitetura nos computadores, principalmente nos processadores. Inicialmente, a primeira questão dessa reavaliação foi a arquitetura do conjunto de instruções, do inglês *instruction set architecture* (ISA). Esse conjunto de instruções é um elemento determinante para a arquitetura do computador, posto que é a parte visível ao programador ou projetista de compiladores que utiliza de seus comandos e parâmetros.

Uma discussão entre os projetistas era porque o computador precisava de um extenso conjunto de instruções complexas. Um pequeno conjunto de instruções era o mais utilizado durante o tempo de processamento. Houve um desenvolvimento de máquinas RISC, principalmente pelo desempenho superior.

Características da Arquitetura CISC

A arquitetura de processadores CISC (*Complex Instruction Set Computers*) ou processadores com conjunto de instruções complexas, apresentava grande quantidade de instruções com múltiplos modos de endereçamento.

Quando foram desenvolvidos a memória era pequena e cara, assim os códigos gerados pelos compiladores tinham necessidade de ser compactos e com operação de processamento bastante eficiente. Essa estrutura possibilitou

o conceito de família de processadores: um conjunto de computadores era oferecido com características de preço e desempenhos diversos, apresentando a mesma arquitetura ao usuário. O primeiro caso foi desenvolvido pela IBM com o System/360, em 1964. Outro exemplo é a arquitetura x86: cada processador era lançado sem alterações básicas de projeto, mas com novas instruções acrescentadas. Nessa linha, existiram os processadores 386 (Figura 4.1), que evoluíram para 486, posteriormente para Pentium e assim por diante. Outra vantagem do emprego de microcódigo reside na rapidez da execução de instruções que estão gravadas em uma memória de controle do tipo ROM, bem mais veloz do que a memória RAM convencional.

Figura 4.1 | Processador 386 que utiliza arquitetura CISC



Fonte: Shutterstock.

Dessa forma, pode-se concluir que os projetistas de arquiteturas CISC consideraram três aspectos básicos (MONTEIRO, 2010):

- Utilização de microcódigo.
- Construção de conjuntos com instruções completas e eficientes.
- Criação de instruções de máquina com complexidade similar aos comandos de linguagens de alto nível.

Embora os fabricantes tenham realizados projetos de arquitetura independentes, pode-se observar no desenvolvimento das arquiteturas CISC alguns pontos em comum. Uma dessas características é que instruções com dois operandos são preponderantes, sendo um campo de origem e destino. Veja a instrução a seguir:

ADD CX, mem: esse comando soma o conteúdo do registrador CX com o valor do endereço da memória **mem** e coloca o resultado da operação no registrado CX (MONTEIRO, 2010).

Outra característica é o uso de diversos modos de endereçamento de memória e de registrador, como registrador para registrador; registrador para memória, memória para registrador e memória para memória.

Como as instruções são complexas, elas apresentam variados tamanhos e quantidade de bytes, e essa variação se dá de acordo com o modo de endereçamento utilizado. Além disso, necessitam de múltiplos ciclos de clock para sua execução.

Adicionalmente, pela limitação de espaço no chip por conta da memória de controle, existem poucos registradores de uso geral, e muitos deles são especializados.

Veja, a seguir, exemplos de instruções complexas da arquitetura CISC:

Quadro 4.1 | Exemplo de funções da arquitetura CISC

Instrução	Significado
CAS	Compara e troca operando.
RTR	Retorna e restaura códigos.
SWAP	Troca palavras dos registradores.

Fonte: elaborado pelo autor.

Refletá

O processador CISC VAX 11/780 tinha 303 instruções diferentes. Pense que existem instruções para manipulação de dados, entradas e saídas, operações lógicas e aritméticas e operação de controle. Será que todas eram utilizadas pelos programadores e desenvolvedores de compiladores? Considere que essas funções são implementadas pelo hardware: será que existe algum espaço físico no chip desse processador?

No entanto, existiam eventuais problemas na arquitetura CISC, que incentivaram pesquisadores e projetistas de sistemas a criar uma alternativa, considerada por eles mais vantajosa. Como as linguagens de alto nível tinham se desenvolvido, cada vez mais os processadores deveriam ter mais instruções para contribuir com a complicaçāo. A cada novo integrante de uma família de processadores eram acrescentadas mais instruções, mas não era possível retirar nenhuma para manter a compatibilidade com as versões anteriores. Além disso, a quantidade de memória disponível aumentou e não era mais necessário um código tão compacto. Apesar disso, essa compactação de código acarretava mais bits para representar a instrução, uma vez que cada comando manipulava uma maior quantidade de bits. Uma das maiores críticas a essa arquitetura era o longo tempo de execução de determinadas

instruções, devido ao uso de microcódigo, implicando a interpretação de cada microoperação, com o decorrente atraso na execução total da instrução.

Assimile

Na arquitetura CISC, o objetivo era a criação de um conjunto numeroso de funções, na maioria das vezes complexas e com vários operandos, o que aumentava o tempo de execução.

Características da arquitetura RISC

Como resultado de trabalhos de diversos pesquisadores em diferentes locais, visando encontrar possibilidades de aperfeiçoamento do desempenho dos processadores com arquitetura CISC, surgiu a arquitetura RISC. A ideia original era entregar um conjunto mínimo de instruções que poderiam realizar todas as operações essenciais: movimentação de dados, operações para Unidade Lógico Aritmética e desvios. Somente instruções explícitas LOAD e STORE tinham permissão para acessar a memória.

Alguns elementos constituem a base da arquitetura RISC: o pequeno conjunto de instruções como tamanho fixo e execução rápida da instrução. A cada ciclo de clock uma instrução é selecionada (buscada) e executada. Essa arquitetura contém uma menor quantidade de modos de endereçamento, maior quantidade de registradores e uso de pipelining. O pipeline (ou paralelismo) é a divisão de uma instrução em muitas partes, para que cada parte seja manipulada por uma parte de hardware dedicada e, com isso, todas as partes possam ser executadas em paralelo (TANEMBAUM, 2013).

Assimile

Quando se fala em ciclo de clock ou ciclo de instrução, trata-se do tempo que o processador gasta para a execução de uma instrução.

Em outras palavras, é o tempo gasto para a busca, decodificação, execução e armazenamento da instrução, ou seja, é o tempo para buscar dois operandos em registradores, para posteriormente realizar a execução das operações da ULA e armazenar o resultado em um registrador.

A característica mais relevante de um sistema RISC era a tendência de apresentar um conjunto de instruções menor que o das máquinas CISC de mesma capacidade. Com um conjunto reduzido de instruções de máquina, são necessários menos transistores no chip, reduzindo o espaço físico e o

custo. Também reduz a complexidade do decodificador de instruções, resultando em diminuição do tempo de decodificação.

Na arquitetura RISC existe uma diminuição da quantidade de instruções disponíveis e o modo de endereçamento em relação à arquitetura CISC. Adicionalmente, a quantidade de bytes utilizados por instrução é fixa na arquitetura RISC com 4 bytes (32 bits), enquanto no CISC é variável, podendo utilizar até dezenas de ciclos para realizar uma instrução. Por sua vez, a arquitetura CISC apresenta mais de uma dezena de modos diferentes.

A quantidade de registradores nas arquiteturas RISC é consideravelmente maior que na arquitetura CISC. As máquinas RISC utilizam os registradores da Unidade Central de Processamento para armazenamento dos parâmetros e variáveis em chamadas de funções e rotinas. Pela menor quantidade de registradores na arquitetura CISC, é necessário a utilização de memória para armazenar esses parâmetros, aumentando o tempo de execução, pois o tempo de acesso à memória de registradores é muito menor que o tempo de acesso à memória.

Exemplificando

O Quadro 4.2 compara diversas características de processadores RISC e CISC. Por exemplo, o processador de arquitetura CISC VAX 11/780 tinha 303 instruções, enquanto o processador da arquitetura RISC MIPS R4000 tinha 94 instruções. A quantidade de registradores em processadores na arquitetura RISC foi de 32 registradores, que pelo menos representa o dobro que os registradores presentes nos processadores CISC analisados.

Quadro 4.2 | Comparativo entre características de alguns processadores RISC e CISC

Características	RISC		CISC	
	MIPS R4000	RS/6000	VAX11/780	INTEL 486
Quantidade de instruções	94	183	303	235
Modos de endereçamento	1	4	22	11
Largura de cada instrução (bytes)	4	4	2-57	1-12
Quantidade de registradores de emprego geral	32	32	16	8

Fonte: Monteiro (2010 p. 383).

Além disso, no RISC há utilização altamente produtivo de pipelining ou paralelismo obtido em face do formato simples e único das instruções de máquina. Pipeline é uma técnica de hardware que permite que a CPU realize a busca de uma ou mais instruções além da próxima a ser executada. O melhor funcionamento do pipelining pode ser atingido quando as instruções não apresentam grandes semelhanças. A semelhança pode ser pelo menos no seu formato e complexidade, e isso pode resultar em um mesmo tempo de execução de suas diversas etapas.

Este fato pode ser comprovado considerando estágios de uma linha de montagem que consistem em tarefas semelhantes em tempo de execução e tamanho (ou largura) de instrução. Uma instrução de processamento é subdividida em etapas, uma vez que cada uma destas etapas é executada por uma porção especializada da CPU, podendo colocar mais de uma instrução em execução simultaneamente.

Adicionalmente, as únicas funções dos processadores RISC que têm acesso à memória são LOAD e STORE e não requerem microcódigos.

Uma implicação da arquitetura RISC é que o programador e o compilador passam a utilizar mais funções para que as instruções de alto nível sejam realizadas, entretanto o tempo de execução em arquitetura RISC pode ser menor que na arquitetura CISC.

Comparativo entre CISC e RISC

É possível comparar as duas arquiteturas, RISC e CISC, analisando diversas características, conforme o Quadro 4.3 a seguir:

Quadro 4.3 | Comparativo entre as arquiteturas CISC e RISC

Característica	Arquitetura CISC	Arquitetura RISC
Quantidade e complexidade das instruções	Muitas instruções complexas.	Menor conjunto de instruções, que são mais simples.
Tamanho das instruções	Tamanho variável.	Tamanho fixo.
Quantidade de ciclos de execução para uma instrução	Múltiplos ciclos, instruções mais complexas.	Apenas um ciclo, instruções mais simples.
Utilização de paralelismo (pipeline)	Fraca utilização.	Ampla utilização.
Quantidade de modos de endereçamento	Muitos modos de endereçamento.	Poucos modos de endereçamento.
Linhas de código versus tempo de execução	Menos linhas de código, mas tempo de execução maior.	Mais linhas de código com tempo de execução menor.

Fonte: elaborado pelo autor.

A categorização dos processadores atuais como sendo RISC ou CISC tem se tornado cada vez mais difícil, já que as linhas que dividem estas arquiteturas não são mais visíveis. Processadores ARM usados em celulares são um bom exemplo de uso da arquitetura RISC. Outro exemplo de uso dessa arquitetura é em consoles, como o Nintendo 64 e o Playstation. Muitos processadores correntes utilizam ambas, com uma abordagem híbrida, ou seja, combinando características CISC e RISC.

Se você observar os datasheets (manuais de processadores) de máquinas RISC atuais, constatará que a quantidade de instruções não é tão pequena, e que existem instruções mais complexas do que existiam em algumas máquinas CISC. O RISC PowerPC, por exemplo, tem um conjunto de instruções maior do que o Pentium CISC (MONTEIRO, 2010). À medida que as tecnologias de microeletrônica avançavam, cada vez mais transistores eram colocados em um mesmo circuito ou pastilha de silício e o custo de produção do transistor, assim como seu tamanho, era reduzido. Dessa forma, uma tecnologia de integração de circuitos em escala muito grande foi desenvolvida e denominada VLSI, ou *very large-scale integration*. A expansão do conjunto de instruções está agora se tornando uma questão menor do debate CISC *versus* RISC. Além disso, hoje não há mais restrições com relação ao tamanho do código, pois a memória tem capacidade elevada.

O Quadro 4.4 mostra algumas das instruções mais importantes no processador 8088, que utiliza a arquitetura CISC. Na coluna operando, um operando de registrador é indicado por um *r* e um endereço efetivo é indicado por um *e*. Se a origem for um registrador, o destino pode ser um endereço efetivo. Portanto, essa combinação de operandos é denotada por *e* \leftarrow *r*. Dados imediatos são indicados pelo sinal (#).

Quadro 4.4 | Exemplos de Instruções principais do processador 8088

Mnemônico	Descrição	Operandos
MOV(B)	Mover palavra, byte	$e \leftarrow r, r \leftarrow e, e \leftarrow \#$
XCHG(B)	Trocar palavra	$r \leftarrow e$
LEA	Carregar endereço efetivo	$r \leftarrow \#e$
PUSH	Passar para pilha	$e, \#$
POP	Retirar da pilha	-
ADD(B)	Somar palavra	$e \leftarrow r, r \leftarrow e, e \leftarrow \#$
ADC(B)	Somar palavra com vai-um	$e \leftarrow r, r \leftarrow e, e \leftarrow \#$
SUB(B)	Subtrair palavra	$e \leftarrow r, r \leftarrow e, e \leftarrow \#$
SBB(B)	Subtrair palavra	$e \leftarrow r, r \leftarrow e, e \leftarrow \#$
INC(B)	Incrementar destino	e
DEC(B)	Decrementar destino	e

Fonte: adaptado de Tanembaum (2010, p. 554).

Como exemplo de instruções da arquitetura RISC, o Quadro 4.5 apresenta a lista de instruções da arquitetura SPARC (arquitetura de processador escalável), criada pela Sun Microsystems. Essa arquitetura é inspirada na máquina RISC I de Berkeley, e o seu conjunto de instruções e a sua organização de registradores são baseados no modelo RISC da Berkeley.

Quadro 4.5 | Exemplos de Instruções do processador SPARC

Tipo de Instrução	Comando / Opcode	Descrição
Instruções Carregar/ Armazenar	LDSB	Carregar byte com sinal
	LDSH	Carregar meia palavra com sinal
	LD	Carregar palavras
	STB	Armazenar byte
	STD	Armazenar palavra
Instruções Aritméticas	ADD	Adicionar
	ADDXC	Adicionar com transporte, modificação de ICC
	SUB	Subtrair
	SUBX	Subtrair com carry
	MULSCC	Passo múltiplo, modificação do ICC

Fonte: adaptado de Stallings (2013, p. 422).

Para efeito de comparação entre os comandos das arquiteturas RISC e CISC serão somados dois valores e armazenados na memória.

Para exemplificar, tomamos como exemplo uma simples instrução de soma de dois operandos em uma arquitetura de processadores CISC (como exemplo, Intel x86). Um dos operandos está armazenado em um registrador A (regA) e o outro operando está armazenado na memória.

Nesse caso, para uma arquitetura CISC é necessário realizar complexos cálculos entre dois valores. Assim, temos que:

```
ADD regA, regB C
```

Nesse exemplo, demandará um tempo adicional para realizar o cálculo do endereço de acesso à memória, ao somar-se o valor C com o conteúdo do registrador B (regB).

Já no caso de uma arquitetura RISC, teremos a mesma operação da seguinte maneira:

```
MOV regB, A  
ADD regA, regB
```

Na arquitetura RISC utilizamos duas instruções, em vez de uma na CISC. Nesse sentido, pode-se verificar o maior número de instruções e a simplicidade das instruções em arquiteturas RISC. Isso permite executar instruções mais rapidamente quando comparada às da arquitetura CISC, uma vez que arquiteturas RISC apresentam menor quantidade de ciclos de instruções (MONTEIRO, 2010).

Com as novas tecnologias houve uma mudança da lógica de controle na arquitetura CISC, combinada com uso de conceitos de otimização de desempenho na arquitetura RISC. Isso desenvolveu processadores que combinam as vantagens de cada arquitetura, formando uma arquitetura de processadores híbridos. Assim, os processadores atuais não são mais nem 100% RISC nem 100% CISC, apresentando características desejadas de ambas arquiteturas. Finalizamos a seção, e agora você tem o conhecimento sobre as diferenças entre as arquiteturas RISC e CISC. Assim, você conseguirá descrevê-las e identificar as características encontradas em cada uma delas.

Sem medo de errar

Como analista do projeto de desenvolvimento de um novo processador de alto desempenho, você deve conhecer a arquitetura do computador que resulta em elementos visíveis a um programador, como a arquitetura do conjunto de instruções. Inicialmente, é interessante conhecer as arquiteturas CISC e RISC, entendendo suas características.

No relatório que você preparará para a reunião, os elementos quantidade e complexidade das instruções são importantes da arquitetura, e você deve levantar diversas características que sejam relevantes, tais como:

- Arquiteturas RISC não apresentam microprogramação e microcódigo; todas as instruções são executadas diretamente pelo hardware.

Na arquitetura CISC era comum, em estruturas computacionais, adicionar uma camada de software entre o hardware e os demais programas, providenciando um novo nível de interpretação, o microcódigo. O microcódigo oferece a possibilidade de incluir ou alterar instruções sem alterar o hardware. Entretanto, ter a maior flexibilidade não é considerada uma boa prática, pois o nível adicional de interpretação acarreta tempo de processamento para interpretação e, com isso, perda de desempenho, que não compensa às novas instruções que possam ser adicionadas.

- As instruções devem ser simples de decodificar, uma característica da arquitetura RISC. Quanto menos complexas forem e menos formatos alternativos tiverem as opções de execução de uma instrução, será melhor. Isto ocorre porque quando a instrução tem poucos formatos, o tempo de identificação da instrução será reduzido. Logo, o tempo de processamento será otimizado.
- Somente as instruções LOAD e STORE devem referenciar a memória.

O acesso à memória principal é uma operação mais demorada que o acesso a registradores. Por esse motivo, é recomendado que apenas as instruções LOAD (leitura) e STORE (gravação) tenham acesso à memória. Isto ocorre na arquitetura RISC.

- Maximize a taxa de execução de instruções.

Além do aumento da velocidade de clock, uma das formas de maximizar o desempenho do computador é por meio do paralelismo, amplamente utilizado na arquitetura RISC.

- Implemente muitos registradores de uso geral.

Quando o processador não tem um registrador disponível para armazenar um valor resultante, ele transfere esse valor para a memória principal. Já sabemos que a transferência de dados entre o processador e a memória principal é um processo mais lento do que a movimentação de dados dentro do processador. Assim, quanto mais registradores possíveis e de uso geral, melhor. A arquitetura RISC apresenta maior quantidade de registradores que a CISC.

Com essas recomendações, será possível ter uma reunião produtiva e com insumos para decidir os próximos passos desse projeto, que ainda está no começo.

Faça valer a pena

1. O Conjunto de Instruções Reduzidas (RISC) apresenta características um pouco diferentes das do Conjunto de Instruções Complexas (CISC). Sobre estas características do RISC, considere as seguintes afirmativas:

- I. O tamanho das instruções no RISC é fixo e apresenta o mesmo formato.
- II. Cada instrução RISC tem tempo de execução de um clock.
- III. Muitas instruções no RISC utilizam o microcódigo.
- IV. O RISC tem alto uso de pipeline.

Considerando o contexto apresentado, é correto o que se afirma em:

- a. I, II e III.
- b. I e III, apenas.
- c. I, II e IV, apenas.
- d. I, III e IV, apenas.
- e. II e IV, apenas.

2. Os processadores modernos e atualmente utilizados em Computadores Pessoais (PCs) realizam a busca de uma ou mais instruções além da próxima a ser executada.

O nome do recurso tecnológico embutido nesses processadores que possibilita essa ação é:

- a. Pipeline.
- b. SPARC.

- c. CISC.
- d. RISC.
- e. Thread.

3. Os processadores utilizados na década de 1970 eram baseados na arquitetura CISC, Conjunto de Instruções Complexas.

Com base no contexto apresentado, avalie as seguintes asserções e a relação proposta entre elas.

- I. A diferença de compreensão entre os comandos de linguagens de alto nível e de instruções de máquina ficou conhecida nas áreas de pesquisa como semantic gap, ou espaço semântico.

PORQUE

- II. Ao se criar processadores com uma considerável quantidade de instruções, com diversos quantidades de modos de endereçamento, o objetivo era a redução do gap semântico.

A respeito dessas asserções, assinale a alternativa correta.

- a. As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- c. A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- d. A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

Seção 2

Arquitetura de sistema de processamento paralelo

Diálogo aberto

Na arquitetura de computadores estabelecida por von Newmann, os sistemas executam seu processamento de forma sequencial ou serial, também conhecido como processamento escalar. Com isso, os processadores apresentavam uma limitação de desempenho, realizando somente uma instrução por vez. Assim, foram desenvolvidas novas tecnologias que executassem seus ciclos de instrução de forma não sequencial. Com essas implementações, em um instante o processador pode realizar diversos ciclos, mesmo que em etapas diferentes, e outras, ainda, que permitem executar mais de uma instrução de forma absolutamente simultânea. Uma alternativa é a utilização do pipeline.

Você está participando do desenvolvimento da arquitetura de um novo processador de uma importante fabricante internacional de processadores, conhecida pela qualidade e pela inovação. Esse novo processador deve ser projetado para oferecer um desempenho superior ao dos concorrentes, mas com preço similar.

Sua participação no projeto tem sido destacada e você contribuiu com uma série de recomendações sobre o conjunto de instruções para o novo processador, recebendo um feedback positivo do gerente do projeto.

Nesta etapa do projeto, será determinado o tipo de arquitetura paralela, já que a arquitetura paralela propiciará um desempenho superior em relação à arquitetura sequencial. O gerente do projeto encontrou a seguinte definição dada pelos autores Almasi e Gottlieb (1989): um computador paralelo é uma coleção de elementos de processamento que cooperam e se comunicam para resolver grandes problemas mais rápido. A decisão a ser tomada pela equipe do projeto é sobre o tipo de arquitetura paralela que será utilizada no novo processador, visando à melhoria de desempenho.

A tarefa de realizar as pesquisas sobre os tipos de arquitetura foi designada para você, sendo requisitado um breve relatório analisando as classificações de Flynn, que descreva brevemente cada uma das categorias e forneça um exemplo de um problema de alto nível que pode ser solucionado com aquela classificação. Adicionalmente, você deve inserir no relatório qual a

classificação recomendada para o projeto. Com base nessas informações, será decidido o tipo de arquitetura paralela a ser implementado.

Para o desenvolvimento desta atividade serão abordadas as arquiteturas para processamento paralelo com a implementação de pipeline, abordando suas classificações, características e utilizações. Será adotada a classificação de Flynn que separa as arquiteturas em: SISD (única instrução, único dado) e SIMD (única instrução, múltiplos dados) e GPUs (unidades de processamento gráfico); MISD (múltiplas instruções, único dado) e MIMD (múltiplas instruções, múltiplos dados).

O futuro desse importante projeto está em suas mãos. Bons estudos!

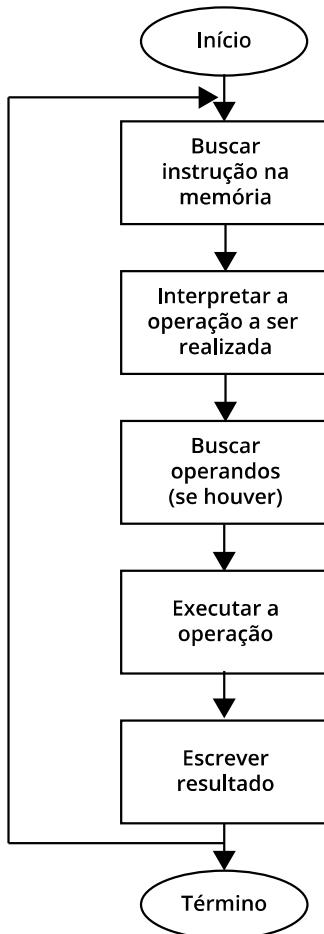
Não pode faltar

Introdução a arquiteturas paralelas

Os primeiros microprocessadores eram baseados na arquitetura de von Newmann e realizavam um controle sequencial do ciclo de instrução, havendo apenas uma instrução em execução por ciclo (MONTEIRO, 2013). Entretanto, visando à melhoria de desempenho, foram desenvolvidas arquiteturas que possibilitavam a execução paralela de atividades.

A Figura 4.2 descreve o ciclo sequencial para a realização de uma instrução, em que a primeira etapa é buscar uma instrução na memória. Logo após ocorre a etapa de interpretação, na qual é identificado o comando o ser realizado. Na sequência, acontece a busca dos dados onde estiverem armazenados, para trazê-los até o processador, realizando a execução efetiva da operação com o(s) dado(s) e o resultado, que é salvo no local definido na instrução. Por fim, o processo reinicia, buscando uma nova instrução.

Figura 4.2 | Fluxograma de ciclo de instrução



Fonte: Monteiro (2013, p. 157).

A arquitetura sequencial de operações foi utilizada nos primeiros processadores de grande porte da década de 1950 e início da década de 1960, e nos primeiros microprocessadores até o início da década de 1980, como o Intel 8080/8085 (MONTEIRO, 2013).

A metodologia sequencial era simplificada e de fácil implementação, porém apresentava desempenho insatisfatoriamente lento e aproveitava inefficientemente os recursos disponíveis do processador. Este modelo pode ser comparado com processos industriais do início do século XX, quando as

montadoras de automóveis construíam um veículo de cada vez, iniciando a montagem do carro seguinte somente após terminar o carro anterior.

Exemplificando

No caso da uma linha de montagem de um veículo, a cada estágio é incorporada uma tarefa de modo que existirão centenas de veículos em preparação, cada um deles em estágios diferentes nessa linha de produção.

Um pipeline funciona de maneira análoga a uma linha de montagem em uma fábrica, permitindo que diferentes estágios de execução de diferentes instruções ocorram ao mesmo tempo por ele (STALLINGS, 2013). O conceito é válido para qualquer produto feito em série e larga escala, como os eletroeletrônicos.

Em nossa vida cotidiana, existem diversas formas de exemplificar o uso de paralelismo, como é o caso das rodovias, que têm mais de uma pista, permitindo a passagem de mais de um veículo por vez (simultaneamente) e nos mercados, que dispõem de mais de um caixa para atender mais de uma pessoa ao mesmo tempo, por exemplo.

Com adoção do paralelismo a intenção é acelerar o começo e o término das execuções das instruções de máquina, criando internamente mecanismos nos processadores para permitir a execução de mais de uma instrução de forma concorrente. Foram desenvolvidos muitos modos de implementar o paralelismo por meio de um tipo novo de tecnologia ou uma maneira diferente de reduzir o tempo de execução das instruções e, consequentemente, melhorar o desempenho do sistema.

Segundo Stallings (2013), a metodologia pipelining é utilizada em processadores de grande porte desde a década de 1960 e nos microprocessadores Intel a partir do 80486. Com essa metodologia, é possível implementar arquitetura do tipo RISC, isto é, conjunto de instruções reduzido.

Para obter mais velocidade, o pipeline deve ter mais estágios. Considere os seguintes desmembramentos do processamento da instrução (STALLINGS, 2013, p. 365):

- **Buscar instrução (FI):** do inglês *Fetch Instruction*, ler a próxima instrução armazenada em um buffer.
- **Decodificar instrução (DI):** determinar o opcode e os especificadores dos operandos.

- **Calcular operandos (CO):** determinar o endereço efetivo de cada operando de origem, que pode envolver diversas formas de endereçamento.
- **Obter operandos (FO):** do inglês *Fetch Operands*, receber cada operando da memória. Já operandos armazenados em registradores não precisam ser lidos da memória.
- **Executar instrução (EL):** realizar a operação indicada e armazenar o resultado na posição do operando de destino, se especificado.
- **Escrever operando (WO):** do inglês *Write Operands*, gravar o resultado na memória.

A Figura 4.3 ilustra como é possível realizar nove instruções diferentes em até 14 ciclos de operação com a utilização do paralelismo nos seis estágios descritos. Neste caso, a cada instante de tempo há somente uma instrução para cada um dos estágios (FI, DI, CO, FO, WO), porém instruções em estágio diferentes podem ser executadas.

Figura 4.3 | Diagrama de tempo para operação do pipeline da instrução

	Tempo →													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instrução 1	FI	DI	CO	FO	EI	WO								
Instrução 2		FI	DI	CO	FO	EI	WO							
Instrução 3			FI	DI	CO	FO	EI	WO						
Instrução 4				FI	DI	CO	FO	EI	WO					
Instrução 5					FI	DI	CO	FO	EI	WO				
Instrução 6						FI	DI	CO	FO	EI	WO			
Instrução 7							FI	DI	CO	FO	EI	WO		
Instrução 8								FI	DI	CO	FO	EI	WO	
Instrução 9									FI	DI	CO	FO	EI	WO

Fonte: Stallings (2013, p. 366).

Assim, a instrução 1 é iniciada no instante 1 com a busca de instrução (FI) e finalizada depois de escrever o operando, no instante 6. A instrução

2 é iniciado no instante 2, sendo executada em paralelo com a instrução 1, e finalizada no instante 7. E, assim sucessivamente, até que no instante 14, a instrução 9 é finalizada.

É importante notar que nem todas as aplicações podem se beneficiar do paralelismo. Por exemplo, paralelismo multiprocessador adiciona custos, como a sincronização de processos e outros aspectos de administração de processos. Se uma aplicação não é sensível à solução paralela, geralmente não compensa, em termos de custo, portá-la para uma arquitetura paralela multiprocessada.

Classificação de Flynn

A tecnologia pipeline teve melhorias com o passar do tempo e, assim, surgiram as modalidades superpipeline e superescalar (MONTEIRO, 2013). A possibilidade de acelerar a execução das instruções, compactando várias delas em uma única operação de leitura e execução paralela, conduziu ao tipo palavra longa de instrução ou *Very Long Instruction Word* (VLIW). Este método utiliza bastante o compilador para produzir seus resultados. O uso intenso e completo de paralelismo aponta para o emprego de multiprocessamento, um sistema constituído de diversos processadores.

Foram criadas diversas formas de organizar as arquiteturas, não existe uma classificação perfeita. Mas uma das classificações mais conhecida e utilizada de processamento não-escalar foi idealizada e divulgada em 1972 por Michael Flynn (FLYNN, 1972) e se refere ao modo como instruções e dados são organizados em um determinado tipo de processamento. Todos os tipos definidos na taxonomia de Flynn mencionam o elemento fluxo de dados (DS), do inglês *data stream*, que significa um conjunto seguido de dados, e fluxo de instruções (IS), do inglês *instruction stream*, que representa um conjunto seguido de instruções a serem executadas. Os quatro tipos de processamento são SISD (única instrução, único dado), SIMD (única instrução, múltiplos dados), MISD (múltiplas instruções, único dado) e MIMD (múltiplas instruções, múltiplos dados), que serão detalhadas a seguir.

SISD – *Single Instruction Single Data* ou Única Instrução, Único Dado

Somente um único conjunto de instruções e de dados são considerados. As máquinas enquadradas nesta classificação apresentam arquitetura definida por von Neumann, isto é, somente há um processador que executa uma instrução completa de cada vez, sequencialmente (em série), cada uma delas manipulando um dado específico ou os dados daquela operação. De

acordo com Stallings (2013), são processadores lentos para os padrões de desempenho atualmente requeridos, deixando, por isso, de ser fabricados. Os processadores 8080/8085 da Intel, 6800 da Motorola e Z-80 da Zilog usavam essa arquitetura. Nela, a unidade de controle (UC) envia um fluxo de instruções (IS) para o processador, que recebe e envia dados para unidade de memória (MU), conforme mostra a Figura 4.4.

Figura 4.4 | Arquitetura SISD



Fonte: Stallings (2013, p. 517).

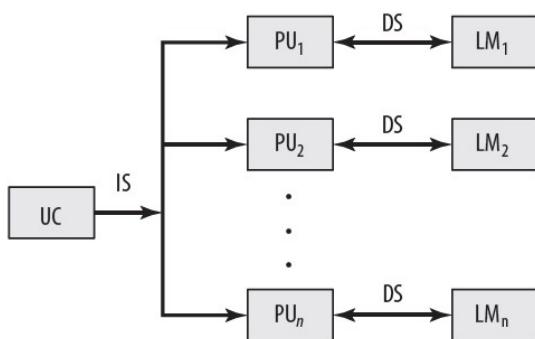
SIMD – *Single Instruction Multiple Data ou uma Única Instrução, Múltiplos Dados*

Neste tipo de arquitetura, o processador opera de modo que uma única instrução acessa e manipula um conjunto de dados simultaneamente (STALLINGS, 2013). Há um único ponto de controle e os processadores executam a mesma instrução simultaneamente com valores de dados diferentes. Um exemplo dessa categoria de arquitetura é o do processador vetorial ou processador matricial. Diversas aplicações científicas processando dados representados em enormes matrizes que precisam ser operadas matematicamente. Os resultados precisam dessas operações, que devem estar disponíveis em pouco tempo após o início do processo. Estas necessidades somente podem ser plenamente atendidas por supercomputadores, máquinas de alto custo e com elevada capacidade computacional, sendo atualmente projetadas com larga quantidade de processadores, integrados em processamentos dedicados, sendo aplicados normalmente em centros de pesquisa governamentais, como pesquisas meteorológicas, ou grandes empresas.

Como exemplo, pode-se imaginar o processamento do seguinte comando 10000 vezes: A (I) * B (I). Em um processador do tipo SISD este comando seria executado 10000 vezes, um após o outro, enquanto que em um processador do tipo SIMD ele poderia ser executado em uma única vez, considerando que a máquina dispusesse de 10000 unidades de cálculo. Outro exemplo desse tipo de arquitetura é o caso das instruções MMX, SSE ou 3D Now.

Nessa arquitetura, a unidade de controle (UC) envia um fluxo de instruções (IS) para múltiplos processadores (PU_n), que recebem e enviam dados para memórias locais (LM_n), conforme mostra a Figura 4.5. Existe um fluxo de dados bidirecional (DS) entre as memórias locais e os múltiplos processadores.

Figura 4.5 | Arquitetura SIMD



Fonte: Stalling (2010, p. 517).

MISD – *Multiple Instruction Single Data* ou Múltiplas Instruções, Dados Únicos

Nesse tipo de arquitetura pode-se utilizar múltiplas instruções para manipular apenas um conjunto único de dados, como um vetor (MONTEIRO, 2010). Esta estrutura não é implementada comercialmente. Hipoteticamente, poderíamos criar por esta teoria uma máquina com vários processadores tentando quebrar um código de criptografia.

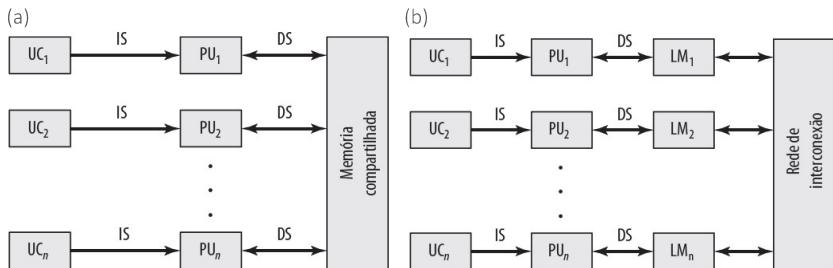
MIMD – *Multiple Instructions Multiple Data* ou Múltiplas Instruções, Múltiplos Dados

É a categoria que requer mais recursos tecnológicos, pois existem múltiplas instruções e múltiplos dados, sendo necessários barramentos para transmissão desses dados e elementos para controle. É a mais avançada tecnicamente, permitindo, quando implementada, elevado desempenho do sistema de computação. É um conjunto de processadores que executam sequências de instruções diferentes simultaneamente em diferentes conjuntos de dados. Sistemas de Multiprocessador Simétrico (SMPs), clusters e sistemas de acesso não uniforme à memória (NUMA) enquadram-se nesta categoria. Um cluster é um tipo de sistema de processamento paralelo que consiste em uma coleção de computadores independentes interconectados por uma rede, trabalhando cooperativamente como um único e integrado recurso computacional (STALLINGS, 2013).

A Figura 4.6 mostra esse tipo de arquitetura com duas organizações de memória diferentes: memória compartilhada para todos os processadores e memória distribuída. Na figura à esquerda, a memória é compartilhada

entre os processadores. Já na figura à direita, cada processador possui uma memória dedicada.

Figura 4.6 | Arquitetura MIMD com memória compartilhada, à esquerda (a); e com memória distribuída, à direita (b)



Fonte: Stallings (2013, p. 517).

Arquitetura MIMD com sistemas de memória compartilhada são aqueles nos quais todos os processadores têm acesso a uma memória global e se comunicam por variáveis compartilhadas, da mesma forma que processos em uniprocessadores.

No caso da memória distribuída, vários processadores não compartilham memória, cada processador deve ter uma porção de memória. Consequentemente, todos os processadores devem se comunicar por meio de passagem de mensagens, o que pode ser ineficiente e penoso para o desempenho. O uso da memória é um fator determinante de classificação de hardware. Entretanto, memória compartilhada e passagem de mensagens são modelos de programação, e não modelos de hardware. Portanto, pertencem mais propriamente ao domínio de software de sistema.

Assimile

A classificação de Flynn considera duas dimensões: se a instrução é única ou são múltiplas instruções e se os dados são únicos ou são múltiplos. O Quadro 4.6 sintetiza essa classificação.

Quadro 4.6 | Classificação de Flynn

Classificação Flynn	Dado único	Dados múltiplos
Instrução única	SISD	SIMD
Múltiplas instruções	MISD	MIMD

Fonte: elaborado pelo autor.

Assim como as arquiteturas vetoriais, as unidades de processamento gráfico – ou *Graphics Processing Units* (GPUs) – exploram o paralelismo em nível de dados, aplicando uma única instrução a uma coleção de dados em paralelo (HENNESY; PETTERSON, 2014). A unidade de processamento gráfico fica localizado na placa de vídeo, e sua função específica é realizar a renderização de gráficos em tempo real. A GPU administra as variadas situações a que cada pixel, ou seja, cada ponto na tela, está vinculado em termos de luminosidade e cor em instantes de tempo muito curtos. Esse tipo de placa é bastante utilizado para games, permitindo imagens mais realistas e de maior qualidade.

Com a popularização das aplicações gráficas, as GPUs também são encontradas em dispositivos móveis, além dos servidores tradicionais ou desktops (HENNESY; PETTERSON, 2014).

A programação da GPU comprehende desafios que vão além de obter bons desempenhos da GPU, como coordenar o escalonamento da computação no processador do sistema e a GPU, assim como a transferência de dados entre a memória do sistema e a memória da GPU.

Para o desenvolvimento de GPU, a empresa de tecnologia NVIDIA elaborou uma linguagem semelhante à C e um ambiente de programação para melhoria da produtividade dos programadores de GPU, atacando os desafios da computação heterogênea e do paralelismo multifacetado.

Figura 4.7 | Exemplo de GPU: NVIDIA GeForce GTX 1080



Fonte: Burnes (2016, [s.p.]).

Refita

Apesar de muito utilizada, a classificação de Flynn apresenta algumas inconsistências. Praticamente não há aplicações para máquinas MISD. Adicionalmente, Flynn assumiu que o paralelismo era homogêneo, mas a coleção de processadores pode ser heterogênea. Já na categoria MIMD, não distingue como processadores são conectados ou como eles enxergam a memória. Se você elaborar um sistema de classificação, quais as variáveis de utilização para classificar o processamento paralelo? Quais seriam as classes mais relevantes?

A arquitetura do processador exerce uma influência direta no desempenho do sistema computacional. Uma arquitetura paralela propicia desempenho superior ao possibilitar um maior fluxo de instruções, gerando uma maior vazão das instruções. A evolução da capacidade de processamento esteve muito ligada a novas arquiteturas paralelas.

Nesta seção, foram apresentadas as soluções para processamento paralelo em diversas configurações. Dada uma aplicação, é possível construir diferentes arquiteturas de processamento paralelo, sendo que o desempenho do sistema se altera significativamente com essa mudança.

Sem medo de errar

Nesta etapa do projeto de novo processador de alto desempenho, foram apresentadas as arquiteturas para processamento paralelo com a implementação de pipeline, abordando suas classificações, características e utilizações. Foram detalhadas as classificações de Flynn.

Essa classificação divide os processadores da seguinte maneira: com instrução simples ou múltiplas instruções e com dados únicos ou dados múltiplos. Os quatro tipos são: SISD (única instrução, único dado), SIMD (única instrução, múltiplos dados), MISD (múltiplas instruções, único dado) e MIMD (múltiplas instruções, múltiplos dados). No SISD há apenas um processador, e o exemplo são máquinas uniprocessadas, sendo recomendada para processamento de texto. SIMD executa uma instrução específica para vários grupos de dados, sendo recomendada para operações matriciais. MISD não é uma arquitetura utilizada comercialmente. Já a arquitetura MIMD consiste em multiprocessadores, e a maior parte de sistemas paralelos utilizam essa arquitetura.

Nessa etapa do projeto será decidido o tipo de arquitetura em relação ao processamento, e você pesquisou sobre as arquiteturas paralelas segundo a classificação de Flynn.

Com base nos requisitos do projeto, para desenvolver um processador de alto desempenho a arquitetura mais adequada é a MIMD, mais avançada tecnologicamente e que apresenta elevado desempenho do sistema de computação. É um conjunto de processadores que executam sequências de instruções diferentes simultaneamente em diferentes conjuntos de dados. Máquinas MIMD, que adotam vários pontos de controle, apresentam sequências de instruções e dados independentes.

Assim, nesse projeto, a arquitetura mais recomendada para ser utilizada é a MIMD.

Faça valer a pena

1. Um computador paralelo é uma coleção de elementos de processamento que cooperam e comunicam-se para resolver grandes problemas mais rápido (ALMASI; GOTTLIEB, 1989). A classificação de Flynn é baseada na quantidade de dados e na quantidade de instruções simultâneas.

Qual a classificação de Flynn que representa a arquitetura de von Newmann, com um processador e uma instrução realizada por vez?

- a. MIMD.
- b. MISD.
- c. SISD.
- d. SIMD.
- e. Pipeline.

2. Em relação à arquitetura MIMD (ou múltiplas instruções, múltiplos dados), considere as seguintes afirmações.

- I. Clusters são grupos de computadores configurados para trabalhar em conjunto com aplicações específicas e são exemplos de MIMD.
- II. Máquinas MIMD atendem à necessidade de computação vetorial.
- III. Em máquina MIMD, um conjunto de processadores que executam sequências de instruções diferentes simultaneamente em diferentes conjuntos de dados.

Assinale a alternativa correta.

- a. Todas as afirmações estão corretas.
- b. Somente as afirmações I e II estão corretas.
- c. Somente as afirmações I e III estão corretas.
- d. Somente as afirmações II e III estão corretas.
- e. Somente a afirmação III está correta.

3. Uma das arquiteturas paralelas existentes em processadores é a categoria MIMD, múltiplas instruções, múltiplos dados. Sobre essa categoria, considere as seguintes afirmações:

- I. Um grupo homogêneo de processadores está inserido nessa classificação.
- II. Existe uma arquitetura MIMD com memória compartilhada entre todos os processadores.
- III. Se vários processadores não compartilham memória, cada processador deve possuir uma porção de memória.

Assinale a alternativa correta.

- a. Todas as afirmações estão corretas.
- b. Somente as afirmações I e II estão corretas.
- c. Somente as afirmações I e III estão corretas.
- d. Somente as afirmações II e III estão corretas.
- e. Somente a afirmação III está correta.

Seção 3

Arquiteturas multithreaded

Diálogo aberto

Caro estudante, seja bem-vindo!

Iniciamos agora uma nova etapa na jornada que capacitará você a compreender a arquitetura do processador multithread. Dentro do projeto de desenvolvimento de um novo processador, iniciou-se a fase de definição do fluxo de um processo e de seus componentes, sendo as threads um desses componentes. Como analista de projetos do projeto de desenvolvimento de um novo processador de alto desempenho, você deve conhecer a arquitetura do computador, que resulta em elementos visíveis a um programador.

Você já participou de reuniões anteriores que definiram por consenso que o projeto terá troca de threads, que são mais rápidas do que as trocas de processos. Foi definido que a arquitetura multithread deve buscar as seguintes características: capacidade de envio de instruções superescalares e utilização de múltiplos contextos de threads. Entretanto, não houve a definição de arquitetura multithread a ser utilizada.

Agora você deve pesquisar as arquiteturas multithread e preparar um relatório com as suas principais características, que vai contribuir para a escolha da arquitetura mais apropriada para o projeto.

Bons estudos!

Não pode faltar

Processos e Threads

Desde os primeiros computadores, os cientistas se empenham para fazer as máquinas resolverem problemas de forma melhor e mais rápida. A aplicação de técnicas de microeletrônica resultou em circuitos integrados (CIs) de alta complexidade e encapsulados em um único chip, isto é, o CI está dentro de um invólucro protetor. Os clocks ou relógios aumentaram a frequência para a faixa de Gigahertz (GHz). Entretanto, existem barreiras físicas que delimitam o desempenho de uniprocessadores.

A medida de desempenho mais relevante para um processador é a taxa na qual ele executa as instruções, que pode ser expressa como:

$$\text{Taxa MIPS} = f \cdot \text{IPC}$$

Onde:

- Taxa MIPS representa a quantidade de milhões de instruções executadas em um segundo.
- f é a frequência de clock do processador, em MHz.
- IPC é o número médio de instruções executadas por ciclo.

Analizando a fórmula de Taxa MPIS, para aumentar a MIPS é possível aumentar a frequência do clock ou o número médio de instruções por ciclo. Para aumentar o IPC, os projetistas implementaram um pipeline de instruções e pipelines múltiplos paralelos de instruções em uma arquitetura superscalar. Com projetos de pipeline e pipelines múltiplos, o principal problema é maximizar a utilização de cada estágio do pipeline. Para melhorar o rendimento, os projetistas criaram mecanismos cada vez mais complexos, como executar algumas instruções em uma ordem diferente da forma que ocorrem no fluxo de instruções e começar a execução de instruções que podem nunca ser necessárias. Entretanto, essa abordagem pode estar alcançando o limite, por causa da complexidade e dos problemas de consumo de energia.

Segundo Monteiro (2010), uma dessas estratégias para melhoria do desempenho é conhecida como *superpipelining* e ocorre quando um pipeline tem estágios que requerem menos da metade de um ciclo de relógio para executar. Pode ser adicionado um relógio interno, o qual, ao realizar a execução com o dobro da velocidade do relógio externo, pode completar duas tarefas por ciclo do relógio externo. Embora superpipelining seja aplicável a arquiteturas RISC e CISC, ele é mais frequentemente incorporado a processadores da arquitetura RISC.

Como não é desejável o aumento da complexidade e do consumo de energia, outra abordagem que permite um grau mais alto de paralelismo em nível de instruções é denominada de multithreading. Segundo Hennessy e Pettersson (2014, p. 193), “multithreading é uma técnica primária usada para expor mais paralelismo para o hardware”. Basicamente, o fluxo de instruções é dividido em vários fluxos menores, conhecidos como threads, de forma que cada thread possa ser executada em paralelo. A variedade de projetos específicos de multithreading realizada nos sistemas comerciais e nos experimentais é grande.

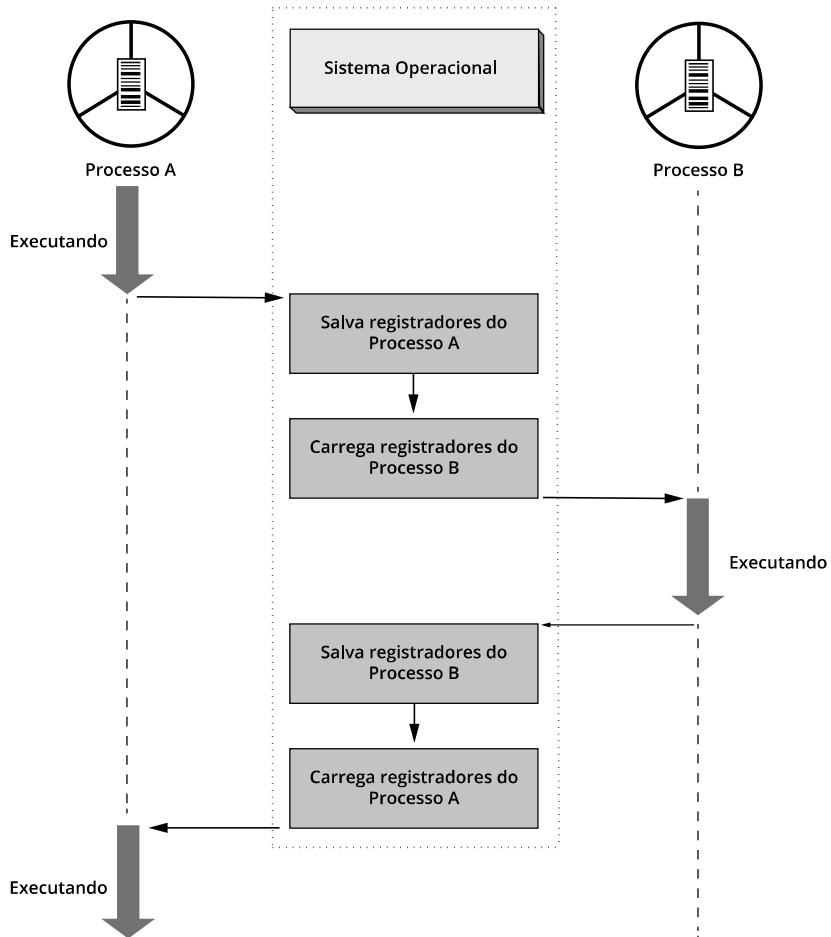
Segundo Machado e Maia (2013), pode-se definir **processo** como uma instância de um programa executando em um computador. Um processo apresenta duas características principais: a posse de um recurso do hardware e o escalonamento/execução, cujas características estão detalhadas a seguir.

Existe a **posse do recurso**, pois um processo inclui um espaço de endereço virtual para guardar a imagem do processo. A imagem do processo é a coleção de programas, dados, pilhas e atributos que definem o processo. Periodicamente pode ser dada a um processador a posse ou controle de recursos, tais como memória principal, canais de E/S, dispositivos de E/S e arquivos.

O **escalonamento/execução** ocorre porque a execução de um processo segue um caminho de execução ou rastro por um ou mais programas. Esta execução pode ser intercalada com a de outros processos. Assim, um processo apresenta um estado de execução – por exemplo, Executando, Pronto, Bloqueado – e uma prioridade de despacho, sendo a entidade escalonada e despachada pelo sistema operacional.

Os sistemas computacionais atuais têm capacidade de realização de uma grande variedade de tarefas ao mesmo tempo. De acordo com Machado e Maia (2013), uma operação muito frequente é a **troca de processos**, na qual um processador muda de um processo para outro, salvando todos os dados de controle do processador, registradores e outras informações do primeiro e substituindo-as com informações de processo. O fluxo da troca de contextos está ilustrado na Figura 4.8 e apresenta as etapas da troca de processo.

Figura 4.8 | Fluxo da troca de contexto



Fonte: elaborado pelo autor.

Outro conceito relevante é a **thread**, que segundo Machado e Maia (2013) consiste em uma unidade de trabalho dentro de um processo que pode ser despachada. Ela contempla um contexto de processador, incluindo o contador de programa (PC), o ponteiro de pilha (SP) e sua própria área de dados para uma pilha, que possibilitam desvio de subrotinas. Uma thread é executada de forma sequencial e pode ser interrompida para que o processador possa se dedicar a outra thread.

Os sistemas operacionais tradicionais, como versões anteriores do Unix, não suportavam threads. A maioria de sistemas operacionais modernos, como Linux, outras versões de Unix e Windows, suporta threads.

Outra operação do processador é a **troca de thread**, que consiste no ato de trocar o controle do processador de uma thread para outra dentro do mesmo processo (MACHADO; MAIA, 2013). Este tipo de troca é menos custoso do que uma troca de processo, já que não é necessário salvar os registradores dos processos e carregar registradores dos processos.

Desta forma, uma thread está focada somente no escalonamento e execução, enquanto um processo se preocupa com escalonamento/execução e posse de recursos. Stallings (2010) aponta que existem situações nas quais várias threads dentro de um processo compartilham os mesmos recursos.

Assimile

Uma troca de thread – unidade de trabalho dentro de um processo – consome bem menos tempo do que uma troca de processo. O processo engloba o escalonamento/execução e posse de recursos, enquanto a thread se preocupa apenas com escalonamento/execução.

Multithreading implícito e explícito

Existe uma distinção entre threads em nível de usuário e threads em nível de kernel que deve evidenciada. As threads em nível de usuário são visíveis para o programação da aplicação, enquanto em nível kernel não são visíveis para aplicação, somente o são ao sistema operacional (STALLINGS, 2010).

Tanto threads em nível de usuário como threads em nível de kernel são referidas como **threads explícitas** e são definidas em software. Os processadores comerciais e a maioria de processadores experimentais utilizam multithreading explícito. Tais sistemas executam instruções de diferentes threads explícitas, diferentes de forma concorrente, ou com intercalação de instruções de diferentes threads em pipelines compartilhados ou com execução paralela em pipelines paralelos.

Por sua vez, multithreading implícito refere-se à execução concorrente de múltiplas threads extraídas de um único programa sequencial. Essas threads implícitas podem ser definidas estaticamente pelo compilador ou dinamicamente pelo hardware.

Abordagens para multithreading explícito

Um processador multithread deve conter ao menos um contador de programa exclusivo para cada thread de execução a ser executada simultaneamente, sendo que há diferenças nos projetos de processadores em relação à quantidade e ao tipo de hardware adicional usado para suportar execução de threads concorrentes (STALLINGS, 2010). Normalmente, a busca de instruções ocorre na base de threads. O processador trata cada thread de forma separada e pode usar uma série de técnicas para otimizar a execução de uma thread, tais como técnicas superescalares, previsão de desvio e renomeação de registradores. Assim, obtém-se paralelismo em nível de threads, fato que possibilita prover melhor desempenho quando combinado com paralelismo em nível de instruções, já que a taxa de instruções executadas em determinado tempo será ampliada.

O multithreading permite o compartilhamento de unidades funcionais em um único processador por vários threads em um padrão superposto. Com multithreading, não há a duplicação de todo o processador, como ocorre um multithreading.

É possível classificar multithreading em quatro abordagens principais:

Uma primeira abordagem é **multithreading intercalado** ou **multithreading de granularidade fina**. De acordo com Stallings (2013), o processador trabalha com dois ou mais contextos de thread ao mesmo tempo, realizando a troca de uma thread para outra a cada ciclo de clock. Se uma thread é bloqueada por causa das dependências de dados ou latências de memória, uma thread no estado pronta para execução é executada. Com essa abordagem, as falhas de *throughput* ficam escondidas e é aí que se originam os *stalls*, isto é, ciclos de despacho completamente vazios. Entretanto ocorre uma perda de desempenho, mensurada pela latência de um único thread.

Outra abordagem é **multithreading bloqueado**, também conhecido como **multithreading de granularidade grossa**. Nele, as instruções de uma thread são executadas sucessivamente até a ocorrência de um evento que possa causar atraso, como uma falha de cache. Esse evento induz uma troca para outra thread. Essa abordagem é eficiente em um processador que interromperia o pipeline em um evento de atraso, como uma falha de cache. Essa troca diminui a necessidade da comutação de threads ser essencialmente livre e é muito menos provável que atrase o processador, pois as instruções de outros threads só serão enviadas quando um thread encontrar um *stall* dispendioso.

Já na abordagem **multithreading simultâneo**, do inglês *Simultaneous Multithreading* (SMT), as instruções são enviadas simultaneamente a partir

de múltiplas threads para unidades de execução de um processador superscalar (STALLINGS, 2013). Isso combina a capacidade de envio de instruções superescalares com o uso de múltiplos contextos de threads. Multithreading simultâneo pode ser entendido como uma variação do multithreading de granularidade grossa que surge naturalmente quando é utilizado em um processador de múltiplo despacho, escalonado dinamicamente.

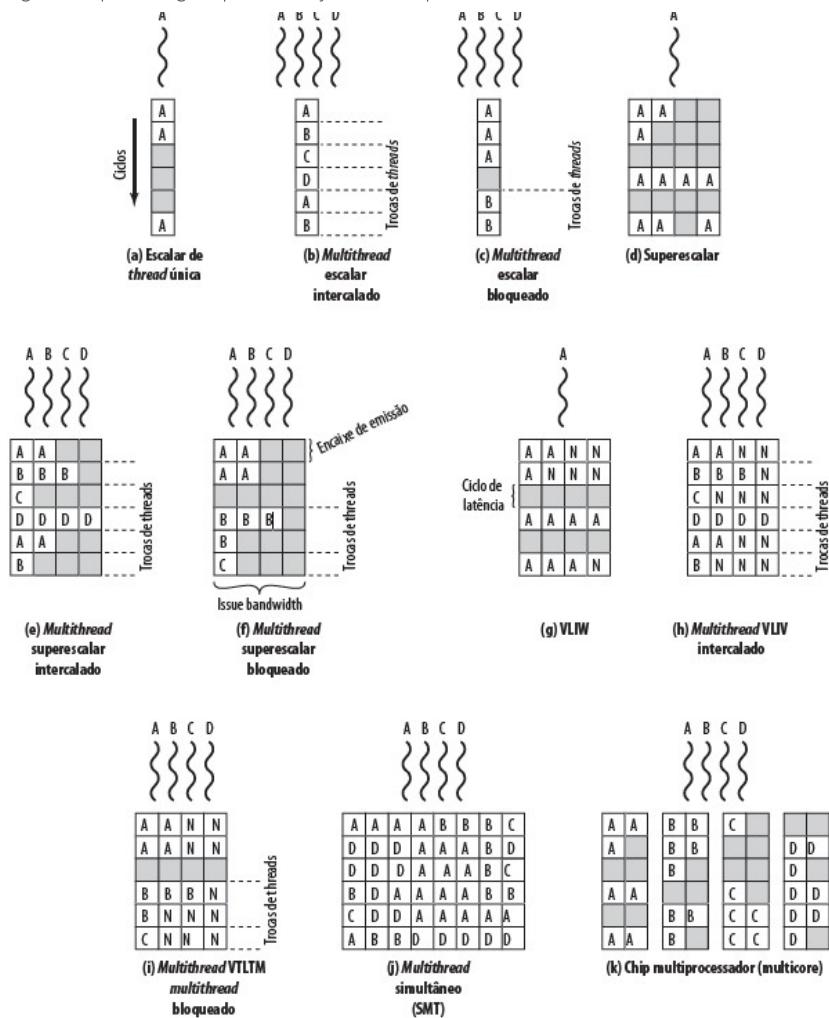
Por fim, na abordagem **chip multiprocessadores**, todo o processador é replicado em um único chip e cada processador lida com threads separadas. O benefício dessa abordagem é que a área de lógica disponível em um chip é utilizada de forma eficiente, sem necessitar de complexidade no projeto do pipeline. Essa abordagem também é conhecida como processadores multicore.

Nas abordagens multithreading de granularidade fina e multithreading de granularidade grossa, instruções de diferentes threads não são executadas simultaneamente, isto é, no mesmo ciclo de relógio. Nesses casos, o processador realiza rapidamente a troca de uma thread para outra, usando um conjunto de registradores diferente e outra informação de contexto. Como consequência, há melhor utilização dos recursos de execução do processador, evitando uma penalidade grande por causa das falhas de cache e outros eventos de atraso. A abordagem SMT envolve a verdadeira execução simultânea de instruções de diferentes threads, usando recursos de execução replicados. Assim como a abordagem SMT, chip multiprocessadores possibilitam também execução simultânea de instruções de diferentes threads.

Visando identificar as diferenças entre as arquiteturas de pipeline é apresentada Figura 4.9, baseada no trabalho de Ungerer, Rubic e Silc (2002), que ilustra algumas arquiteturas possíveis de pipeline com e sem multithreading, sendo comparada a sequência de operações de threads em cada abordagem.

Os slots de envio representam as posições das quais as instruções podem ser enviadas em um dado ciclo de clock. Na Figura 4.9, cada linha horizontal representa um slot ou mais slots de envio em potencial para um único ciclo de relógio. Assim, a largura de cada linha corresponde à quantidade máxima de instruções que podem ser emitidas em um único ciclo de clock. O eixo vertical representa a sequência de tempo de ciclos de relógio. Na ocorrência de um slot vazio, identificado como sombreado, um slot de execução não foi usado em um pipeline. Um no-op (*no operation*) é indicado pela letra *N* no slot.

Figura 4.9 | Abordagens para execução de múltiplos threads



Fonte: Stallings (2010, p. 530).

Reflita

Para o projeto de um sistema multithreading, deve-se verificar qual abordagem que deve ser adotada. A arquitetura poderia variar dependendo do projeto?

Os itens (a), (b) e (c) da Figura 4.9 apresentam abordagens diferentes com um processador escalar, isto é, um processador com execução de somente uma única thread por vez. Nesses casos, fica evidente pelas figuras que a existência de slots vazios, em cinza, acarreta perda de rendimento. **Thread escalar único**, representado na Figura 4.9 (a), é pipeline simples encontrado em máquinas RISC e CISC tradicionais, sem multithreading e com um processador. A abordagem **multithread escalar intercalado** é a estrutura multithreading mais simples de ser implementada. Já que troca de uma thread para outra em cada ciclo de clock, os estágios do pipeline podem ser mantidos com elevada ou total ocupação. Por sua vez, na abordagem **multithread escalar bloqueado**, uma única thread está em execução até que a ocorrência de um evento de atraso que pararia o pipeline, quando ocorreria a troca de thread.

Aparentemente, multithread intercalado oferece melhor utilização do processador do que multithread de bloqueio. Entretanto, essa melhoria ocorre com redução de desempenho em relação a threads únicas, posto que há concorrência de várias threads pelos recursos de cache, o que eleva a probabilidade de uma falha de cache para uma determinada thread.

A partir de abordagens multithread, existe o envio de várias instruções por ciclo, resultado em mais oportunidades para execução paralela das instruções. Nos itens (d) a (i) da Figura 4.9, é possível o envio de até quatro threads em um único ciclo de relógio, já que existem quatro colunas. Na abordagem **superescalar** básica (d), não há nenhuma multithread. Durante alguns ciclos de relógio, nem todos os slots de envio são usados. Durante esses ciclos, um número menor máximo de instruções é usado, resultando na chamada perda horizontal. Durante outros ciclos de instrução, nenhum slot de envio é aproveitado, existindo ciclos em que nenhuma instrução pode ser enviada, dando origem à perda vertical.

Na abordagem **multithread superescalar intercalado**, durante cada ciclo são emitidas tantas instruções quantas forem possíveis a partir de uma única thread. Com essa técnica, atrasos potenciais por causa das trocas de threads são eliminados. No entanto, o número de instruções enviado em qualquer ciclo ainda fica limitado pelas dependências que existem dentro de qualquer thread.

Por sua vez, no **multithread superescalar bloqueado**, as instruções de apenas uma thread podem ser emitidas durante qualquer ciclo, e a multithread bloqueada é usada.

Na arquitetura **slot de envio – VLIW** (do inglês *Very Long Instruction Word*), um número constante de instruções forma uma única palavra. Em

geral, a VLIW é construída pelo compilador, que insere as operações que podem ser paralelizadas na mesma palavra. Caso não for possível preencher a palavra completamente com instruções a serem emitidas em paralelo, são utilizadas a instruções no-op (*no-operation*).

A abordagem **VLIW multithread intercalado** deveria fornecer eficácia semelhante àquela provida por multithreading intercalada em uma arquitetura superescalar. Enquanto a **multithread VLIW bloqueada** deveria fornecer eficácia semelhante àquela provida por multithread bloqueada em uma arquitetura superescalar.

Duas últimas abordagens ilustradas na Figura 4.9 possibilitam execução paralela e simultânea de várias threads. O **multithreading simultâneo** (SMT), mostrado no item (j), é um sistema capaz de emitir oito instruções simultaneamente. Se uma thread apresenta alto grau de paralelismo em nível de instruções, ela poderá preencher totalmente os ciclos horizontais, em alguns ciclos. Em outros ciclos, podem ser enviadas as instruções de duas ou mais threads. Se um número de threads estiver ativo normalmente, é possível enviar o número máximo de instruções em cada ciclo, fornecendo um nível alto de eficiência.

Já **chip multiprocessador ou multicore**, apresentado no item (k) da Figura 4.9, ilustra um chip que dispõe de quatro processadores, cada um tendo um processador superescalar de envio dupla. Para cada processador é atribuída uma thread partir da qual ele pode enviar até duas instruções por ciclo.

Comparando os itens (j) e (k) da mesma figura, observa-se que um chip multicore com a mesma capacidade de envio de instruções como um SMT não pode alcançar o mesmo grau de paralelismo em nível de instruções. Isso ocorre porque o chip multicore não é capaz de esconder os atrasos enviando instruções de outros threads. Por outro lado, o chip multicore deve ter um desempenho melhor que um processador superescalar com a mesma capacidade de envio de instruções, porque as perdas horizontais serão maiores para o processador superescalar. Além disso, é possível usar multithread dentro de cada processador em um chip multicore, e isso é feito em algumas máquinas atuais.

Exemplificando

Modelos mais recentes de Pentium 4 usam uma técnica de multithread a qual a literatura da Intel refere-se como *hyperthreading* (MARR *et al.* 2002). Basicamente, a abordagem do Pentium 4 é a utilização do **multithreading simultâneo (SMT)** com suporte para dois threads. Assim, um único processador multithread torna-se logicamente dois processadores.

Já o chip IBM Power5 é usado em produtos PowerPC de alto nível e combina o chip multiprocessador com SMT. Esse chip é composto por dois núcleos isolados, sendo que cada um é um processador multithread com capacidade de executar dois threads concorrentemente usando SMT. Algumas simulações de desempenho com mais processadores mostraram que multithread adicional, além do suporte para dois threads, pode diminuir o desempenho por causa do trabalho com a cache, pois os dados de uma thread deslocam os dados necessários para outra thread.

Nesta seção foram apresentados os diferentes tipos arquiteturas multithreaded, que incluem multithreading intercalado, multithreadings bloqueados, SMT e chip multiprocessado, verificando como estruturas multithreading podem ser utilizadas para melhoria do desempenho do processador.

Sem medo de errar

Analizando a situação-problema proposta, é necessário a determinação de qual arquitetura multithread será adequada, buscando capacidade de envio de instruções superescalares e utilização de múltiplos contextos de threads.

Com foi definido envio de instruções superescalares, todas as arquiteturas escalares estão excluídas automaticamente.

Como na abordagem **superescalar** básica não há nenhum multithread, essa também deve ser descartada.

Considerando a arquitetura **multithread superescalar intercalado**, durante cada ciclo são emitidas tantas instruções quantas forem possíveis a partir de um único thread. Com essa técnica, atrasos potenciais por causa das trocas de threads são eliminados. No entanto, o número de instruções enviado em qualquer ciclo ainda é limitado pelas dependências que existem dentro de qualquer thread, assim essa arquitetura também será descartada.

Por sua vez, no **multithread superescalar bloqueado**, as instruções de apenas uma thread podem ser emitidas durante qualquer ciclo. Dessa forma, também não é uma alternativa adequada para o projeto.

Na arquitetura **slot de envio (VLIW)** são colocadas várias instruções em uma única palavra. Caso não for possível preencher a palavra completamente com instruções a serem emitidas em paralelo, no-ops são usados, o que leva a uma redução de velocidade.

Por sua vez, **multithreading simultâneo** (SMT) possibilita execução paralela e simultânea de várias threads. Esse é um sistema capaz de emitir diversas instruções ao mesmo tempo. Se um thread apresenta um alto grau

de paralelismo em nível de instruções, ela pode, em alguns ciclos, ser capaz de preencher todos os slots horizontais. Em outros ciclos, as instruções de duas ou mais threads podem ser enviados. Se threads suficientes estão ativos, normalmente seria possível enviar o número máximo de instruções em cada ciclo, fornecendo um nível alto de eficiência.

Já no caso de **chip multiprocessadores ou multicore**, cada núcleo tem um processador superescalar de envio de duas instruções simultâneas. Para cada processador é atribuído um thread, a partir do qual ele pode enviar até duas instruções por ciclo.

Adicionalmente, é possível combinar as potencialidades das arquiteturas **multithreading simultâneo (SMT)** e **chip multiprocessadores** ao utilizar multithread dentro de cada processador em um chip multicore, para potencializar o ganho de processamento, fato que é implementado em diversos processadores atuais.

Faça valer a pena

1. A utilização de arquiteturas multithread possibilita o aumento da taxa MIPS (milhões de instruções por segundo) que processador pode executar. Um processador analisado trabalha com cinco contextos de thread ao mesmo tempo, e a cada ciclo de clock realiza a troca de thread.

Qual o tipo de arquitetura multithread que esse processador apresenta?

- a. Multithread implícito.
- b. Multithread explícito de granularidade fina.
- c. Multithread explícito de granularidade grossa.
- d. Multithread simultâneo.
- e. Chip multiprocessado.

2. Em relação à arquitetura multithread, considere as afirmações a seguir:

- I. A arquitetura superescalar apresenta melhor desempenho de paralelismo que a arquitetura multithread.
- II. Na arquitetura multithread escalar bloqueado, uma única thread é executada até que ocorra um evento de atraso que pararia o pipeline, quando há troca de thread.
- III. Na arquitetura slot de envio (VLIW), várias instruções são colocadas em uma única palavra.

Assinale a alternativa correta.

- a. Somente as afirmações I e II estão corretas.
- b. Somente as afirmações I e III estão corretas.
- c. Somente as afirmações II e III estão corretas.
- d. Somente a afirmação II está correta.
- e. Todas as afirmações estão corretas.

3. Os projetos de arquitetura multithread diferem em quantidade e tipo de hardware adicional usado para suportar execução de threads concorrentes. Considere a seguinte afirmação sobre o multithread simultâneo.

- I. No multithreading simultâneo (SMT) as instruções são enviadas simultaneamente a partir de múltiplas threads para unidades de execução de um processador escalar.

PORQUE

- II. São agregados à capacidade de envio de instruções escalares com o uso de múltiplos contextos de threads.

Assinale a afirmação correta.

- a. Ambas as afirmações estão corretas, mas a afirmação II não justifica a afirmação I.
- b. Ambas as afirmações estão corretas e afirmação II justifica a afirmação I.
- c. Somente a afirmação I está correta.
- d. Somente a afirmação II está correta.
- e. Nenhuma afirmação está correta.

Seção 4

Arquiteturas multicore

Diálogo aberto

Prezado aluno, nesta seção você vai estudar os processadores multicore e conhecer alguns dos processadores da atualidade, além de interpretar as funções dos core (núcleos) e threads dentro desta arquitetura.

Aumentar a velocidade de processamento em um único núcleo já foi tendência no passado. Hoje, com a necessidade de otimização de espaço físico, são desenvolvidos chips de processamento multicore (múltiplos núcleos), nos quais é possível otimizar os transistores, gerando economia de energia e a redução de calor.

Pois bem, chegou o momento de mostrar seu conhecimento como analista de projetos para o desenvolvimento de um novo processador de alto desempenho. Já está acertado com a equipe que serão utilizados processadores multicore, e isso tornará mais viável a ideia de paralelismo entre os núcleos do processador, ou seja: trabalhar com uma arquitetura que possibilita a execução simultânea, em que cada núcleo admite uma tarefa.

Neste caso você deverá apresentar para a equipe um relatório do funcionamento dos núcleos do processador em um ambiente Windows. Você poderá utilizar a ferramenta de gerenciamento de tarefas para facilitar suas explicações. Faça um print (cópia da tela) da execução das tarefas dos respectivos núcleos e, finalizando o seu relatório, contextualize o funcionamento dos threads e dos caches envolvidos em cada core (núcleo) do processador. Fique à vontade para caracterizar um processador em específico.

Aproveite o conteúdo apresentado e bons estudos!

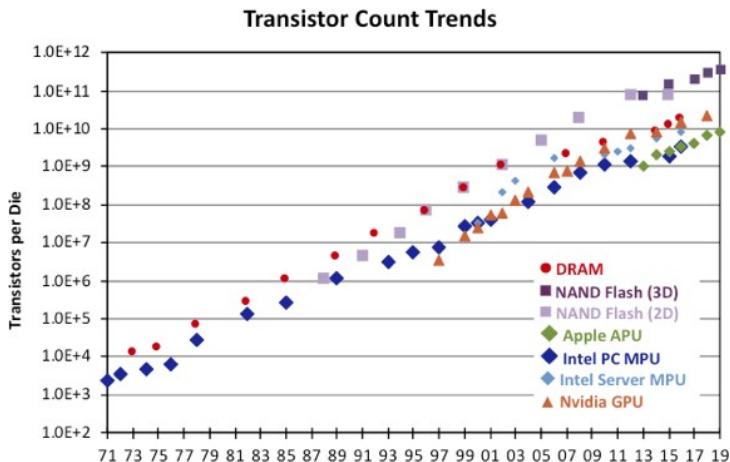
Não pode faltar

Ao longo dos anos temos a evolução no desempenho dos processadores, principalmente no que está relacionado a sua capacidade de processamento. Apesar dos processadores atuais apresentarem alto desempenho em termos de velocidade de processamento e quantidade de tarefas executadas, essa evolução já era prevista na década de 1960, quando Gordon Moore, cofundador da Intel, disse que a tecnologia utilizada na fabricação de processadores, ou seja, os transistores, dobraria a cada dezoito meses. De fato isso ocorreu: a miniaturização de transistores possibilitou duplicar a quantidade

de transistores, permitindo um significativo aumento da capacidade de processamento de chips, além dos dispositivos de armazenamento.

O gráfico ilustra o progresso da lei de Moore, conforme Figura 4.10, que mostra o acompanhamento dessa lei até 2019 para os dispositivos como as memórias RAM, flash, microprocessadores e processadores gráficos, nas últimas cinco décadas.

Figura 4.10 | Quantidade de transistores por pastilhas em processadores comerciais



Sources: Intel, SIA, Wikichip, IC Insights

Fonte: IC INSIGHTS INC. (2020, [s.p.]).



Um gráfico com a evolução das arquiteturas de processadores por números de transistores ao longo dos anos pode ser visualizado acessando o QR Code ou o link a seguir: https://cm-cls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-1/arquitetura_organizacao_computadores/u4/s4/arq_or_com_u4s4.pdf

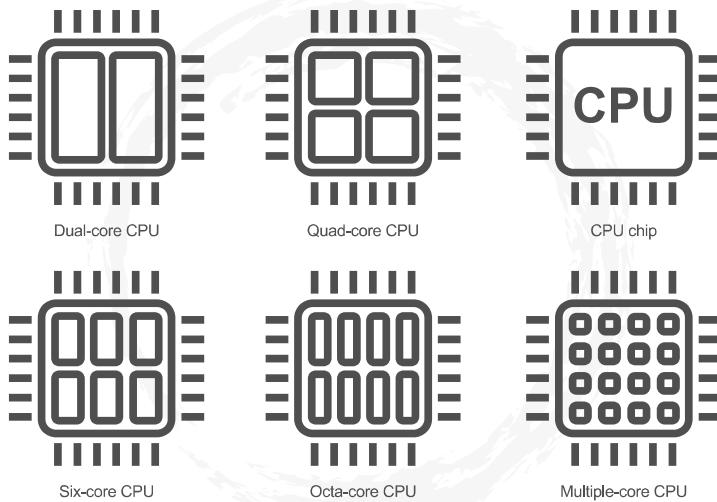
O gráfico ilustra dados de 1971 a 2018, e como o número de transistores cresceu para as arquiteturas de processadores. Aproveite e utilize o zoom para visualizar a imagem.

Outro ponto importante é a atenção dos projetistas de hardware em melhorar continuamente as arquiteturas de processadores, por exemplo, com a inclusão de pipelining a nível de instrução, arquiteturas de processadores superescalar, multithreading e os conhecidos processadores multinúcleo ou multicore.

Um processador multicore pode ser definido como dois ou mais núcleos de processamento em uma única pastilha de silício (chip) (STALLINGS,

2013, p. 559). As tarefas a serem executadas são distribuídas entre esses núcleos do processador. Em outras palavras, cada núcleo pode executar múltiplos processos simultaneamente. A Figura 4.11 ilustra a estrutura de processadores multicore. São processadores com dois, quatro, um chip, seis, oito e dezesseis núcleos.

Figura 4.11 | Estrutura de processadores com múltiplos núcleos de processamento



Fonte: Shutterstock.

Cada núcleo de processamento é organizado como uma unidade de central de processamento independente, composta por unidade lógica aritmética e unidade de controle e registradores. Além dessas, há as memórias cache, que podem ser compartilhadas entre os núcleos e podem ser dedicadas. Por exemplo: temos as caches Level 1 (L1), Level 2 (L2) e Level 3 (L3) para dados e de instruções. As memórias cache são memórias de alta velocidade (estão no topo da hierarquia de memória) e baixa capacidade e podem ser acessadas rapidamente, havendo uma significativa redução nos tempos para o acesso aos dados, uma vez que a cache armazena os dados que são utilizados com maior frequência pelo processador. Nesse sentido, quando a UCP necessita de uma informação a ser processada, essa informação primeiramente é buscada na memória cache e, caso a informação não seja encontrada, é buscada na memória RAM. Outro ponto a destacar é que quanto menor o nível de cache, maior é a sua velocidade e menor é a sua capacidade de armazenamento. Por exemplo: a cache L1 apresenta maior velocidade que a L2, e a cache L2 é mais

veloz que a L3. Em contrapartida, em termos de armazenamento, a cache L1 dispõe de menor capacidade de armazenamento que as caches L2 e L3.

Ao visitar sites de fabricantes de processadores, temos um termo frequentemente utilizado, o *Smart Cache* (cache inteligente). Esse é o nome dado para uma tecnologia desenvolvida pela Intel, em que as caches são compartilhadas entre os núcleos de processamento do processador multicore, sendo as caches L2 e L3 compartilhadas entre os núcleos.

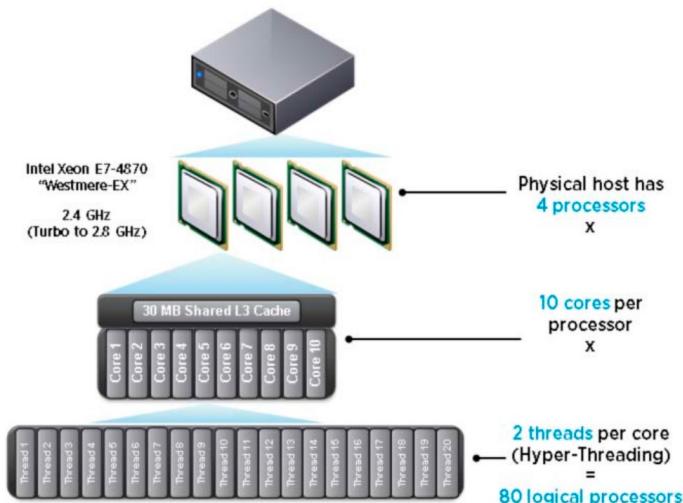
Além das melhorias mencionadas, os engenheiros implementaram alterações na organização de processadores permitindo o aumento do nível de paralelismo em nível de instruções, de forma a executar mais de uma instrução a cada ciclo de clock. Posteriormente, com o surgimento da arquitetura superescalar, foram implementados vários pipelines com recursos de execução replicados para possibilitar a execução de instruções em pipelines paralelos.

Como forma de melhorar o desempenho da arquitetura superescalar, foi desenvolvida a arquitetura multiprocessada (*Simultaneous Multiprocessing – SMP*), que consiste em arquitetura com mais de um processador. Esse tipo de arquitetura tem recursos de memória, disco compartilhados e roda em um mesmo sistema operacional. Nesse sentido, dois processos podem ser executados ao mesmo tempo em dois processadores.

Ainda como evolução, surgiu a arquitetura multithread simultânea (SMT, do inglês *Simultaneous Multithreading*), também chamada de hyper-threading. Nessa técnica é explorado o pipelining em nível de threads em um processador. Assim, um processador físico pode simular dois processadores lógicos. Cada um dos processadores lógicos tem o seu próprio controlador de interrupção, bem como um banco de registradores. Essa arquitetura conta também com alguns recursos compartilhados, tais como, cache, ULA e barramentos.

Na Figura 4.12 é apresentada uma arquitetura de quatro processadores físicos, sendo dez núcleos por processador que serão compartilhados com a memória cache L3 (30 MB compartilhado entre os dez core). Cada core tem 2 threads, ou seja, 80 processadores lógicos (quatro núcleos físicos multiplicados por dez processadores lógicos; o resultado também multiplicado por duas threads por core).

Figura 4.12 | Arquitetura de um processador de quatro núcleos físicos (Intel Xeon-4870)

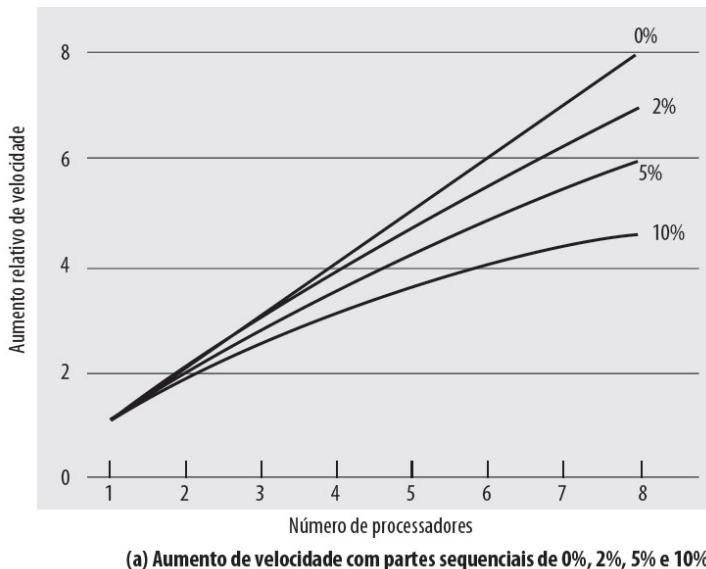


Fonte: VMware, Inc. (2014, p. 5).

Desempenho de software

Os benefícios na utilização de uma arquitetura multicore dependem do quanto efetiva será a exploração dos recursos paralelos para a aplicação. A Lei de Amdahl (1967) mostrou a atratividade de uma organização multicore se apenas uma pequena fração do código for inherentemente serial. A Figura 4.13 ilustra o aumento de velocidade em função do número de núcleos e a fração de código que é serializada, que deve ser realizada no mesmo núcleo.

Figura 4.13 | Aumento de velocidade por número de processador com variação de partes sequenciais



(a) Aumento de velocidade com partes sequenciais de 0%, 2%, 5% e 10%

Fonte: Stallings (2013, p. 564).

Diversos tipos de aplicações se beneficiam com esse ganho de velocidade, tais como: aplicações com múltiplos processos, aplicações Java e aplicações multithread nativas.

Aplicações com múltiplos processos são caracterizadas pela existência de muitos processos de uma única thread. São exemplos desse tipo de aplicação bancos de dados da Oracle e sistemas SAP.

Exemplificando

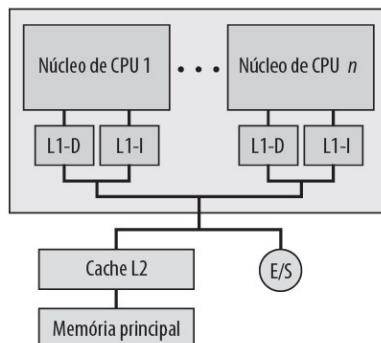
Em aplicações Java, a *Java Virtual Machine* é um processo multithread que agenda e gerencia a memória para aplicações Java. As aplicações que utilizam servidor de aplicações da plataforma *Java 2 Enterprise Edition* (J2EE) se beneficiam diretamente da tecnologia multicore. Já as aplicações multithread nativas se caracterizam por ter um número reduzido de processos ideais para multithread.

Organização multicore

Existem diferentes formas de organização multicore, dependendo do número de núcleos de processadores no chip, número de níveis de memória cache e quantidade de memória cache compartilhada.

Uma organização, ilustrada na Figura 4.14, contém um único cache no chip e esse cache é dedicado por núcleo, sendo utilizado em chip embutidos. De forma geral, a memória cache L1 é dividida em dados e instruções. Um exemplo deste tipo de organização é ARM11 MPCore (ARM, 2009), um produto com múltiplos processadores baseado na família do processador ARM11.

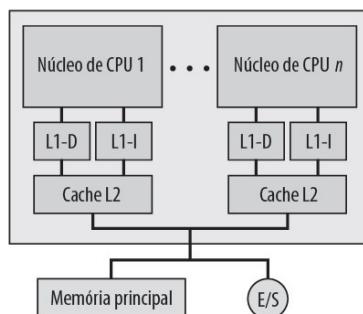
Figura 4.14 | Organização multicore com cache L1 dedicado



Fonte: Stallings (2013, p. 567).

Em um outro tipo de organização não há compartilhamento do cache no chip, mas há espaço para cache L2 no chip. O processador AMD Opteron utiliza essa arquitetura multicore.

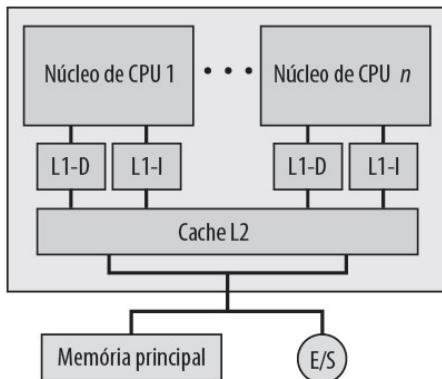
Figura 4.15 | Organização multicore com cache L2 dedicado



Fonte: Stallings (2013, p. 567).

Outra possibilidade está ligada ao compartilhamento de memória cache L2. O compartilhamento do cache L2 demonstra vantagens no desempenho em relação à memória cache exclusiva, pois a comunicação entre os processadores é mais simples e realiza-se por meio das posições compartilhadas e acesso mais rápido para posições da memória principal referenciadas por outras threads. O processador Intel Core Duo é um exemplo desse tipo de organização, ilustrada na Figura 4.16.

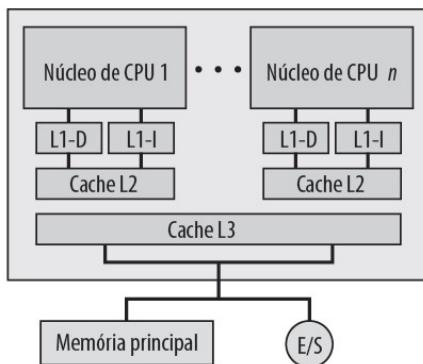
Figura 4.16 | Organização multicore com cache L2 compartilhado



Fonte: Stallings (2013, p. 567).

Com o crescimento da memória disponível e do número de núcleos, a utilização de uma memória cache L3 compartilhada, que combina o compartilhamento de cache L2 e cache L2 dedicados por núcleo, tende a resultar em um desempenho melhor que uma cache L2 compartilhada. Os processadores Intel i7 utilizam esse tipo de organização.

Figura 4.17 | Organização multicore com cache L3 compartilhado



Fonte: Stallings (2013, p. 567).

Outro tipo de decisão em um projeto de sistemas com múltiplos núcleos é se os núcleos implementarão multithread simultâneo (SMT) ou serão superescalares.

Podemos destacar que o Intel Core Duo utiliza núcleos superescalares, enquanto o Core i7 utiliza núcleos SMT, que aumentam o número de threads suportados pelos sistemas multicore.

A seguir são apresentados alguns processadores multicore.

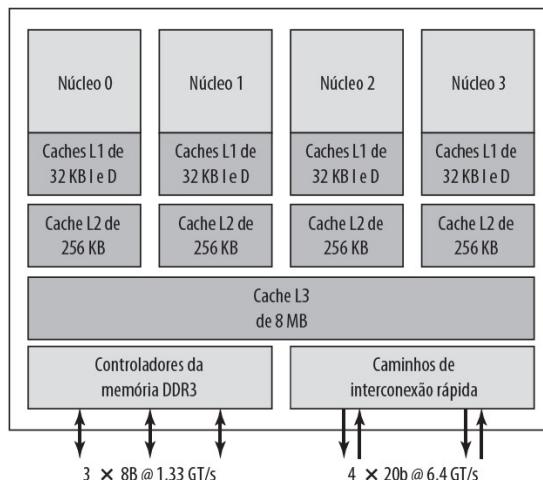
Intel Core Duo

Este processador, lançado em 2006, implementava dois processadores da família x86 com uma cache L2 compartilhada. A memória cache L1 é dedicada para cada núcleo e tem 32 KB (kilobytes) de cache de instruções e 32 KB de cache de dados. Nesta arquitetura, a dissipação de calor do chip é gerenciada visando maximizar o desempenho do processador, considerando os limites de temperatura do chip. A memória cache L2 é de 2 MB, sendo compartilhada entre os núcleos, permitindo a alocação dinâmica do espaço do cache.

Intel Core i7

O processador Core I7 da Intel, lançado ao mercado em 2008, utiliza quatro processadores x86 SMT, com caches L2 de 256 KB dedicadas para cada um dos quatro núcleos, e uma cache L3 compartilhada de 8 MB. A Figura 4.18 traz o diagrama de bloco do processador i7.

Figura 4.18 | Diagrama de blocos do processador Intel Core i7



Fonte: Stallings (2013, p. 567).

Esse processador pode se comunicar externamente com outros chips por meio dos controladores de memória DDR3 e por caminhos de interconexão rápida.

As tabelas 4.1 e 4.2 comparam características de alguns processadores Intel e AMD.

Tabela 4.1 | Comparativo da processadores Intel Core da 9^a Geração

Modelo	Frequência	Núcleos	Threads	Cache	Frequência Turbo max
i9 -10980XE	4.80 GHz	18	36	24.75 MB	4.60 GHz
i9 -7920 X	2.90 GHz	12	24	16.5 MB	4.30 GHz
i9-9960X	3.10 GHz	16	32	22	4.40
i9 -79980XE	2.60 GHz	18	32	24.75 MB	4.20 GHz
Xeon Platinum 8168	2.70 GHz (*3.70 GHz)	28	56	33 MB	3.70 GHz
Xeon Platinum 9282	2.60 GHz (*3.80 GHz)	56	112	77 MB	3.80 GHz
AMD Ryzen™ Threadripper 3990 X	2.9 GHz	64	128	256 MB (máx. L3)	4.3 GHz

Fonte: adaptado de Intel ([s.d.]).

Tabela 4.2 | Comparativo da processadores Intel Core da 9^a Geração (continuação)

Características	Intel Core i9	Intel Core i7	Intel Core i5
Número de núcleo	8	8	6
Número de threads	16	8	6
Máxima frequência do processador (GHz)	Até 5,0	Até 4,9	Até 4,6
Tamanho da memória cache (MB)	16	2	9
Número de canais de memória	2	2	2
Tecnologia Intel Hyper-Threading	Sim	Sim	Não

Fonte: adaptado de Intel ([s.d.]).

A Intel lançou, em 2017, a família Intel® Core™ X-series Processors, com objetivo de criar uma plataforma extrema para jogos e criação de conteúdo (INTEL, 2017). Nessa família existem processadores com diversas quantidades de núcleos e threads diferentes: 4 núcleos e 8 threads, 6 núcleos e 12

threads, 8 núcleos e 16 threads, 10 núcleos e 20 threads, 12 núcleos e 24 threads e até 18 núcleos e 36 threads.

Esperamos que você tenha compreendido os tipos de arquitetura multicore e os benefícios de sua utilização. Para evolução da capacidade de processamento continuar, novas arquiteturas podem ser criadas nos próximos processadores.

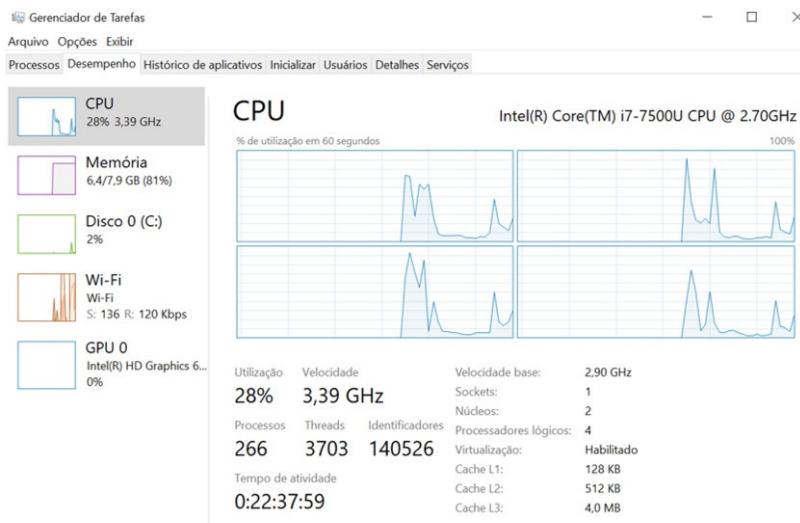
Bons estudos!

Sem medo de errar

Lembre-se de que você ficou encarregado de elaborar um relatório para a equipe de desenvolvimento do novo processador, pelo qual deverá mostrar o funcionamento dos núcleos do processador em um ambiente Windows. Nesse caso específico foi realizada uma demonstração pelo gerenciador de tarefas do Windows, mas você poderá usar outras ferramentas para suas colocações. Finalize o relatório com uma contextualização sobre o funcionamento dos threads e dos caches envolvidos em cada core (núcleo) do processador, ficando à vontade para caracterizar um processador em específico.

Veja a seguir, na Figura 4.19, o funcionamento dos núcleos do processador em um i7 de 7^a geração da Intel.

Figura 4.19 | Funcionamento de desempenho do processador CPU Intel® Core™ i7-7500U CPU @ 2.70GHz



Fonte: captura de tela do gerenciador de tarefas do Windows elaborada pelo autor.

CPU Intel® Core™ i7-7500U CPU @ 2.70GHz

Velocidade base: 2,90 GHz

Sockets: 1

Núcleos: 2

Processadores lógicos: 4

Virtualização: Habilitado

Cache L1: 128 KB

Cache L2: 512 KB

Cache L3: 4,0 MB

Utilização 23%

Velocidade 3,36 GHz

Tempo de atividade 0:23:04:53

Processos 268

Threads 3731

Identificadores 143226

Podemos considerar que a execução das threads no processador pode ter o compartilhamento das informações. Cabe relatar que esse armazenamento na cache pode ser feito distintamente em outros core do processador.

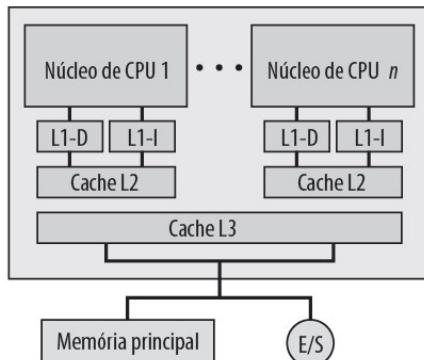
Lembre-se de que as memórias caches apresentam três níveis: a L1 é a de menor nível e menor espaço, porém muito mais rápida que a L2 e L3. A L1 é dividida em duas partes, uma para instruções e outro para os dados. A L2 é destinada para gravação de dados e informações, e a L3 é a maior entre os três níveis, e pode dar apoio a qualquer core para execuções de tarefas.

De acordo com Stallings (2013), existe um ganho quando há um compartilhamento do cache L2 e em paralelo há cache L2 dedicada, como ocorre no caso de Intel Core i7, em que cada núcleo tem a sua cache L2 dedicada, e os núcleos compartilham uma memória cache L3.

Uma das vantagens é a que interferência construtiva pode reduzir as taxas de falhas do sistema, ou seja, se uma thread em um core acessa uma posição de memória, esse acesso traz para linha de conteúdo a posição de referenciada para a cache compartilhada. Caso outra thread de outro core tente acessar depois, as posições de memória já estarão disponíveis na cache compartilhada no chip, gerando maior rapidez de acesso.

Utilizando algoritmos adequados de substituição de linha, existe uma alocação dinâmica da quantidade de cache compartilhada para cada core, de modo que threads com maiores conjuntos de trabalho apresentam mais cache. Além disso, a comunicação entre processadores é facilitada quando implementada por meio das posições de memória compartilhadas. Assim, a estrutura multicore do processador pode ser representada na Figura 4.20.

Figura 4.20 | Organização multicore com cache L3 compartilhado



Fonte: Stallings (2013, p. 567).

Para finalizar, a 10^a geração de processadores Intel® Core™ dispõe de gráficos Intel® Iris® Plus, que proporcionam uso de inteligência artificial (IA) no computador em larga escala para acelerar o desempenho. Esses processadores possibilitam um novo nível de integração para aprimorar as experiências com PCs atuais e as necessárias para o futuro.

Como pudemos verificar, a organização e arquitetura do processador evoluíram muito com o tempo. Como essa arquitetura pode mudar ainda nos próximos anos?

Ótimos estudos!

Faça valer a pena

1. Após décadas de evolução da velocidade de processamento de processadores com somente um núcleo, foram desenvolvidos processadores multicore.

Assinale a alternativa que representa uma razão para migração para arquitetura multicore.

- a. A implementação de pipeline não implicava o aumento da complexidade de um processador.
- b. Ocorre melhoria de desempenho quando as aplicações podem ser executadas em paralelo.
- c. Processadores com maior velocidade tinham o mesmo consumo de potência que processadores mais lentos.
- d. Com o desenvolvimento de tecnologias quânticas, foi facilitada a construção de processadores multicore.
- e. Para aumentar a velocidade do processador, foi necessária a incorporação de múltiplos estágio de memória cache.

2. Considere um computador com um sistema multicore com quatro núcleos e compartilhamento de memória cache L3. Cada núcleo utiliza Multithreading Simultâneo (SMT) e suporta até quatro threads simultâneas.

Analizando do ponto de vista da aplicação, esse sistema multicore aparentemente dispõe de quantos núcleos?

- a. 1 núcleo.
- b. 4 núcleos.
- c. 8 núcleos.
- d. 16 núcleos.
- e. 32 núcleos.

3. Um computador multicore combina dois ou mais processadores, também chamados de núcleos ou core, em uma única pastilha de silício. Considere as seguintes afirmações sobre processadores multicore:

- I. Aplicações com múltiplos processos de thread única terão um ganho de desempenho relevante com essa estrutura multicore.
- II. As memórias cache dos processadores devem ser necessariamente compartilhadas.

III. Existem diferentes formas de organização multicore, dependendo do número de núcleos processadores no chip, número de níveis de memória cache e quantidade de memória cache compartilhada.

É correto o que se afirma em:

- a. I e II, somente.
- b. I e III, somente.
- c. II e III somente.
- d. I, somente.
- e. III, somente.

Referências

ALMASI, G. S.; GOTTLIEB, A. **Highly parallel computing**. Redwood City (EUA): Benjamin-Cummings Publishing, 1989.

AMDAHL, G. M. **Validity of the single-processor approach to achieving large-scale computing capability**. Proceedings of the April 18-20, 1967, Spring joint computer conference. Abr. 1967, p. 483–485 of the AFIPS Conference, 1967.

ARM. **ARM11 MPCore™ Processor Technical Reference Manual Revision**: r2p0. Infocenter, 2008. Disponível em: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0360f/index.html>. Acesso em: 27 mar. 2020.

BURNES, A. **Introducing the GeForce GTX 1080**: Gaming perfected. NVIDIA Corporation, 6 maio 2016. Disponível em: <https://www.nvidia.com/en-us/geforce/news/geforce-gtx-1080/>. Acesso em: 5 mar. 2020.

FLYNN, M. **Some computer organizations and their effectiveness**. IEEE Transactions on Computers, set. 1972.

GIRON, A. A.; OYAMADA, M. S. Análise do desempenho do H.264 em arquiteturas multicore. In: EPAC – ENCONTRO PARANAENSE DE COMPUTAÇÃO, 4., 2011, Cascavel-PR. **Anais** [...].2011. Cascavel: Unioeste, 2011. p. 21-30. Disponível em: http://www.inf.unioeste.br/epac/epac2011/anais/artigos_epac/A03.pdf. Acesso em: 21 fev. 2020.

HENNESSY, J. L.; PETTERSON, D. A. **Arquitetura de computadores**: uma abordagem quantitativa. 5. ed. Rio de Janeiro: Elsevier, 2014.

IC INSIGHTS, INC. Transistor Count Trends Continue to Track with Moore's Law. **Research Bulletin**, 5 mar. 2020. Disponível em: <https://www.icinsights.com/data/articles/documents/1242.pdf>. Acesso em: 27 mar. 2020.

INTEL CORPORATION. **9th Generation Intel® Core™ Desktop Processors**. [S.l.: s.n.], [s.d.]. Disponível em: <https://www.intel.com.br/content/dam/www/public/us/en/documents/product-briefs/9th-gen-core-desktop-brief.pdf>. Acesso em: 22 fev. 2020.

INTEL CORPORATION. **Nova família do processador Intel® Core™ X-series** – apresentando o processador Intel® Core™ i9 Extreme Edition. [S.l.: s.n.], 1 jun. 2017. Disponível em: <https://newsroom.intel.com/news-releases/nova-familia-processador-intel-core-x-series-apresentando-o-processador-intel-core-i9-extreme-edition>. Acesso em: 19 mar. 2020.

MEDEIROS, A.; MEDEIROS, A. L.; KREUTZ, M. E. Exploração de paralelismo em nível de instruções e de tarefas em uma arquitetura de processamento não convencional. **Revista Brasileira de Computação Aplicada**, v. 7. n. 3, 2015, p. 120-133.

MONTEIRO, M. A. **Introdução à Organização de Computadores**. 5. ed. Rio de Janeiro: LTC, 2010.

MONTEIRO, M. A. **Introdução à Organização de Computadores**. 5. ed. Rio de Janeiro: LTC, 2010.

MOTYCKA, L. B. *et al.* **Um Comparativo de Consumo Elétrico entre Arquiteturas x86 e ARM em Servidores de Bancos de Dados**. Universidade Regional do Noroeste do Estado do Rio Grande do Sul (UNIJUI), out. 2013. Disponível em: https://www.researchgate.net/publication/259339458_Um_Comparativo_de_Consumo_Eletrico_entre_Arquiteturas_x86_e_ARM_em_Servidores_de_Bancos_de_Dados. Acesso em: 21 fev. 2019.

MOURA, B. M. P.; VIEIRA JUNIOR, R. R. Arquitetura de sistemas para clusters e grades computacionais: uma solução independente de fabricante baseada em clusters beowulf. **Revista CCEI – URCAMP**, v. 19, n. 34, 2015. Disponível em: <https://pdfs.semanticscholar.org/29ff/b42f1a0107b20d92acd5f521ba6d68a08915.pdf>. Acesso em: 30 mar. 2020.

NULL, L.; LOBUR, J. **Princípios Básicos de Arquitetura e Organização de Computadores**. 2. ed. Porto Alegre: Bookman, 2011.

OLUKOTUN, K.; HAMMOND, L. The future of microprocessors. **ACM Queue**, set. 2005, p. 26-34.

SILVA, L. F.; ANTUNES, M. J. V. **Comparação entre as arquiteturas de processadores RISC e CISC**. Faculdade de Engenharia da Universidade do Porto. Porto, 2010. Disponível em: <http://www.inf.unioeste.br/~guilherme/oac/Risc-Cisc.pdf>. Acesso em: 21 jan. 2020.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

TANEMBAUM, A. S. **Organização estruturada de computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013.

VMWARE, INC. **Performance and scalability of Microsoft SQL server on VMware vSphere 5.5**. Performance study. Technical white paper. Palo Alto/CA, 2014. Disponível em: <https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-sql-server-vsphere55-performance-white-paper.pdf>. Acesso em: 30 mar. 2020.

ISBN 978-65-86461-15-2



9 786586 461152 >