# Introdução a Arquitetura de Computadores

### Alisson Vasconcelos de Brito

**Índice**

[Prefácio](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/pr01.html)

[1. Público alvo](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/pr01s01.html)

[2. Método de Elaboração](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/pr01s02.html)

[3. Contribuição](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/pr01s03.html)

[1. Introdução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01.html)

[1.1. O que é a Arquitetura de um Computador?](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s01.html)

[1.2. Por que estudar Arquitetura de Computadores?](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s02.html)

[1.3. Um aluno de Licenciatura em Computação precisa estudar](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s03.html)

[1.4. Arquitetura geral](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s04.html)

[1.4.1. Operações básicas](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s04.html#_operacoes_basicas)

[1.5. Sistemas Analógicos x Sistemas Digitais](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s05.html)

[1.6. O Transistor](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s06.html)

[1.7. A Lei de Moore](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s07.html)

[1.8. A evolução dos computadores](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html)

[1.8.1. O ENIAC](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_o_eniac)

[1.8.2. A Arquitetura de von Neumann](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_a_arquitetura_de_von_neumann)

[1.8.3. A IBM](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_a_ibm)

[1.8.4. As gerações dos computadores](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_as_geracoes_dos_computadores)

[1.8.5. Memórias de semicondutores](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_memorias_de_semicondutores)

[1.8.6. A Intel](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_a_intel)

[1.8.7. A Apple e a Microsoft](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#_a_apple_e_a_microsoft)

[1.9. Recapitulando](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s09.html)

[1.10. Atividades](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s10.html)

[2. Unidade Central de Processamento (CPU)](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02.html)

[2.1. O que é um programa?](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s01.html)

[2.1.1. Software X Hardware](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s01.html#_software_x_hardware)

[2.2. Estrutura de uma CPU](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html)

[2.2.1. Os papéis dos barramentos e da memória](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#_os_papeis_dos_barramentos_e_da_memoria)

[2.2.2. Os registradores](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_registradores)

[2.2.3. Unidade Lógica e Aritmética (ULA)](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_ULA)

[2.2.4. Unidade de Controle (UC)](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_UC)

[2.3. Ciclo de Instrução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s03.html)

[2.3.1. Busca de Dados](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s03.html#_busca_de_dados)

[2.4. Interrupções](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s04.html)

[2.5. Sobre o desempenho](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s05.html)

[2.6. Exemplo de execução de um programa](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s06.html)

[2.7. Aumentando o desempenho com Pipeline](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html)

[2.8. Limitações do Pipeline](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s08.html)

[2.8.1. Medidas de desempenho](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s08.html#_medidas_de_desempenho)

[2.8.2. Exemplos de calcular o desempenho de um processador](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s08.html#_exemplos_de_calcular_o_desempenho_de_um_processador)

[2.9. Recapitulando](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s09.html)

[3. Unidade de Controle](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03.html)

[3.1. Introdução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s01.html)

[3.2. Microoperações](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html)

[3.2.1. Busca de Instrução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html#_busca_de_instrucao)

[3.2.2. Busca indireta](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html#_busca_indireta)

[3.2.3. Execução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html#_execucao)

[3.2.4. Salvar resultado](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html#_salvar_resultado)

[3.2.5. Salto condicional](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s02.html#_salto_condicional)

[3.3. Tipos de Microoperações](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s03.html)

[3.4. Decodificação](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s04.html)

[Glossário](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html)

**Lista de Figuras**

1.1. [Arquitetura básica de um computador](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s04.html#fig_arquitetura)

1.2. [Sinal Analógico versus Sinal Digital](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s05.html#fig_analog_digital)

1.3. [Estrutura de um transistor tipo MOSFET](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s06.html#fig_mosfet)

1.4. [Evolução do número de transistores nos processadores em comparação com a Lei de Moore](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s07.html#fig_moore)

1.5. [Estrutura da Máquina de von Neumann](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#fig_neumann)

2.1. [Estrutura de uma CPU](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_CPU)

2.2. [Estrutura de uma CPU com barramentos](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_CPU_com_barramento)

2.3. [Estrutura de uma CPU com registradores](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_arq_detalhada)

2.4. [Ciclo de Instrução](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s03.html#fig_ciclo_instrucao)

2.5. [Ciclo de Instruções com interrupções](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s04.html#fig_ciclo_com_interrupcao)

2.6. [Execução com várias interrupções](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s05.html#fig_entrada_saida)

2.7. [Processador adaptado para trabalhar com Pipeline de cinco estágios](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html#fig_pipeline)

2.8. [Execução em pipeline de cinco estágios](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html#fig_pipeline2)

3.1. [Execução em Microoperações](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s01.html#fig_execucao)

**Lista de Tabelas**

2.1. [Exemplo de configuração de um processador](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s08.html#tabela_exercicio1)

**Capítulo 1. Introdução**

**Objetivos do capítulo**

Ao final deste capítulo você deverá ser capaz de:

* Definir o que é a Arquitetura de Computadores e sua relevância
* Apresentar a Arquitetura Geral de um computador e suas principais operações
* Diferencias sistemas digitais de analógicos
* Apresentar o funcionamento de um transistor e sua relevância para a industria de dispositivos digitais
* Destacar a Lei de Moore e seu impacto para a evolução da industria de dispositivos eletrônicos digitais
* Identificar os principais fatos da evolução dos computadores

Você sabe o que é Arquitetura de computadores? Você já se perguntou porque precisa estudar Arquitetura de Computadores? Esse capítulo nós vamos aprender que essa é uma das principais disciplinas da Ciência da Computação. Não só isso, foi a Arquitetura de Computadores que permitiu que a humanidade avançasse em todos os aspectos da ciência, saúde e tecnologia. Ao final desse capítulo, espero que você concorde comigo.

|  |
| --- |
| Anterior |

## 1.1. O que é a Arquitetura de um Computador?

O termo arquitetura é principalmente utilizado na construção e decoração de edificações. Ele diz respeito à forma e a estrutura de uma construção. O termo refere-se à arte ou a técnica de projetar e edificar o ambiente habitado pelo ser humano. Na computação o termo foi adaptado para denominar a técnica (talvez até a arte também) de projetar e construir computadores. Nesse livro você não vai aprender a construir seu próprio computador. Para isso eu recomendo outros autores, John L. Hennessy, David A. Patterson e Andrew S. Tanenbaum. Esses autores produzem livros para engenheiros de computadores e acompanhá-los antes de se tornar um pode ser uma tarefa bastante árdua. Aqui você vai conhecer o computador por dentro e saber como ele funciona. Você não será capaz de construir um computador, mas saberá o suficiente para entender como os programas funcionam e até porque o computador para de funcionar as vezes, ou funciona lentamente, e que nessas situações, pressionar teclas do teclado rapidamente, ao mesmo tempo que move o mouse aleatoriamente, não faz o computador voltar a trabalhar novamente.

## 1.2. Por que estudar Arquitetura de Computadores?

É essencial que todos profissionais da Computação tenham pelo menos conhecimentos básicos de Arquitetura de Computadores. Saber como o computador funciona nos permitirá entender sua capacidade (e incapacidade) de resolver problemas, sobre como programá-los da melhor forma possível, como deixar o computador e os dados contidos neles mais seguros, como ganhar desempenho e o que faz ele ficar tão lento às vezes a ponto de querermos destrui-lo. Então, estudar Arquitetura de Computadores é tão importante para um profissional de Computação, como estudar Anatomia é importante para um médico. Antes de iniciar qualquer estudo na Medicina, um médico precisa saber em detalhes o funcionamento do corpo humano. Quais são seus órgãos, como eles trabalham individualmente e como se relacionam para formar um sistema (digestivo, respiratório, motor etc.). Com a Arquitetura de Computadores é semelhante. Vamos aprender quais são os componentes de um computador, como eles funcionam e como eles trabalham em conjunto formando um sistema. Sem dúvidas o ser humano é a máquina mais perfeita já criada, mas vamos ver que o Computador é uma das máquinas mais incríveis que o homem já criou.

## 1.3. Um aluno de Licenciatura em Computação precisa estudar

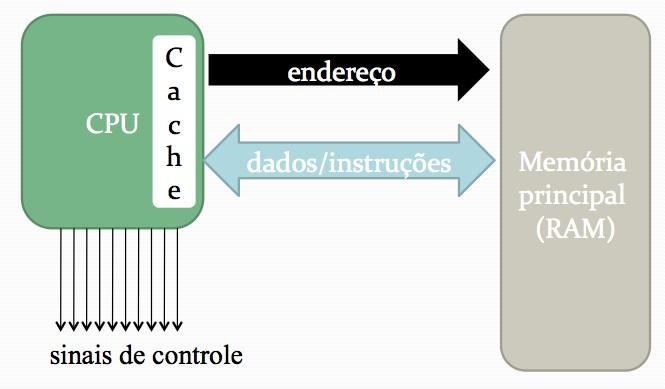
Arquitetura de Computadores?

Ao longo de minha experiência como professor de Arquitetura de Computadores para alunos de Licenciatura em Computação, eu sempre ouvi essa pergunta. “Por que precisamos estudar Arquitetura de Computadores?”. Espero que isso já esteja claro para você. Mas se ainda não estiver, aqui vai uma outra razão. Você será um licenciado e vai trabalhar no ensino e aprendizagem da Ciência da Computação. Como ensinar alguém sobre essa ciência se você não souber em detalhes como um computador funciona? Isso seria como um professor de Farmácia que não conhece bem a Química, ou um professor de Matemática que não conhece os números.

## 1.4. Arquitetura geral

Hoje em dia há muitos tipos de computadores e diversas arquiteturas. Elas são frutos de muitos estudos, pesquisas e avanços tecnológicos. Mas todos computadores compartilham uma arquitetura comum. Essa arquitetura é o que separa um computador de uma calculadora de bolso, de um aparelho de televisão ou um relógio de pulso. Essa arquitetura é apresentada na [Figura 1.1, “Arquitetura básica de um computador”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s04.html#fig_arquitetura).

**Figura 1.1. Arquitetura básica de um computador**



Todo computador possui uma Unidade Central de Processamento, ou, do inglês, **Central Processing Unit** (CPU) e uma Memória Principal. Todos os dados a serem processados pela CPU, para operações lógicas e aritméticas, precisam estar na memória. Da memória os dados são transferidos para a CPU através de fios paralelos de comunicação, chamados de Barramento de Dados. Entretando, a CPU não toma decisões por si própria. Ela não sabe que dados deve trazer da memória, muito menos que operação executar com eles. Para isso, ela precisa que instruções, também armazenadas na memória, sejam trazidas para a CPU através do Barramento de Endereço. Cada instrução informa para a CPU que operação ela deve executar, com quais dados e o que ela deve fazer com o resultado da operação.

Para poder se localizar, a memória é organizada em endereços. Todos os dados e as instruções são localizadas através desses endereços. Cada instrução indica para a CPU que dados devem ser transferidos e processados através dos endereços desses dados. Esse endereço é transferido para a memória pela CPU através do Barramento de Endereço. A memória localiza o tal dado e o transfere para a CPU via Barramento de Dados. As instruções são desenvolvidas pelo programador, através de linguagens de programação. As ferramentas de compilação transformam os programas escritos em linguagens de alto nível, como C, Java e Phython, em instruções de máquina, que são finalmente copiadas para a memória no momento em que precisam ser executadas. Cada instrução é armazenada em um endereço diferente da memória. Na execução normal, a CPU passa para a memória, via Barramento de Endereço, o endereço da primeira instrução do programa, a memória transfere a instrução pelo Barramento de Instrução, a CPU a executa e, em seguida, solicita a instrução do endereço seguinte. Assim, os programas são executados sempre de forma sequencial, a não ser que uma instrução especial solicite que ela salte para uma instrução que não seja a consecutiva. Isso é o caso quando há instruções condicionais (como o “se” ou “if”), instruções de repetição (como “while” e o “for”), ou chamadas a sub-programas, ou mesmo, por ordem do Sistema Operacional, para que o programa pare de executar para que um outro tome seu lugar.

As memórias são, quase sempre, muito mais lentas do que as CPUs. Isso exigiu, ao longo dos anos, que as CPUs possuíssem também uma porção interna de memória muito rápida, chamada Memória Cache. A tecnologia que permite essas memórias serem mais rápidas, tornam-as também muito caras. Por isso que sua capacidade geralmente é muito limitada. Para acelerar ainda mais, elas são instaladas dentro das CPUs. Todos os dados e instruções transferidos da Memória Principal para a CPU são salvos também na Cache. Como a Cache não é capaz de guardar todos os dados da Memória Principal, apenas os dados mais recentes transferidos para a CPU permanecem na Cache. Técnicas muito avançadas são aplicadas para que se consiga, no máximo possível, manter os dados mais importantes daquele instante na Memória Cache.

A CPU também é responsável por enviar sinais de controle aos outros dispositivos do computador, como periféricos, dispositivos de entrada e saída, e memórias externas. Esse sinais são enviados quando uma instrução dá ordem para tal. Por exemplo, quando uma instrução pede que uma mensagem sem impressa na tela, a CPU, ao receber e executar essa instrução, envia para o controle do monitor que imprima na tela a mensagem contida o endereço que também foi passada pela instrução.

É esse comportamento que diferencia um computador de outros dispositivos eletrônicos mais simples. A essência da CPU não é muito diferente de uma calculadora de bolso. Ela executa operações lógicas e aritméticas. Entretanto, no projeto do computador, o papel do homem foi substituído pela programação. Todas instruções das tarefas que a CPU precisa executar são armazenadas na memória e, a partir de então, a CPU pode trabalhar sem qualquer interferência externa. Com a programação, a CPU pode também executar tarefas diversas, desde simulações, jogos, tocar músicas e vídeos etc. Simplificando, o computador é uma máquina programável e de propósito geral.

### 1.4.1. Operações básicas

Todos computadores executam três operações básicas:

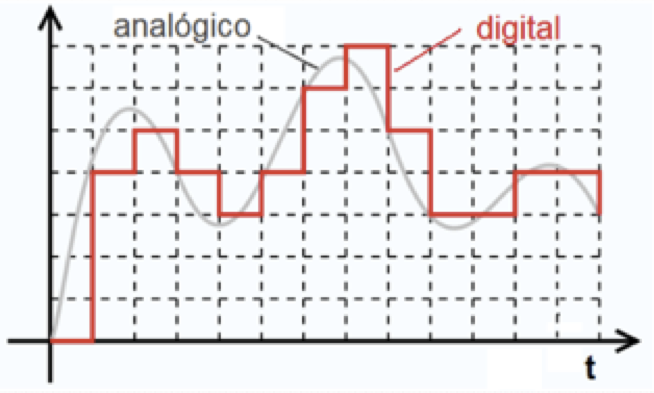
* Movimentação de dados
* Processamentos de dados
* Armazenamento de dados

A movimentação de dados é a transferência de um dado de um ponto para outro do computador. Pode ser de um endereço de memória para outro, de um dispositivo de entrada para a memória, ou da memória para um dispositivo de saída. O processamento de dados ocorre quando a CPU recebe um determinado dado e executa uma operação que o modifica de alguma forma. Já as operações de armazenamento ocorre quando a CPU precisa registrar um dado em algum local específico, como salvar um dado no disco rígido, ou num pendrive, ou mesmo na Memória Principal.

## 1.5. Sistemas Analógicos x Sistemas Digitais

Para sabermos a importância de um computador e sua forma de funcionamento, precisamos conhecer suas potencialidades e suas limitações. O computador é um dispositivo eletrônico digital. Isso significa que ele armazena, processa e gera dados na forma digital. Por outro lado, o computador não é capaz de processar dados analógicos. Eles antes precisam ser convertidos para digital para poderem ser utilizados por computadores. Mas o que venha a ser um dado analógico? Qualquer informação presente na natureza, como uma imagem, um som ou um cheiro, pode ser analisada em no mínimo duas componentes. Uma seria a sua intensidade e outra o tempo. A [Figura 1.2, “Sinal Analógico versus Sinal Digital”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s05.html#fig_analog_digital) a seguir apresenta essa representação, onde o sinal em forma de onda cinza seria a representação de um sinal analógico.

**Figura 1.2. Sinal Analógico versus Sinal Digital**



Um som, por exemplo, é formado por vibrações no ar de diferentes intensidades (amplitudes) ao longo do tempo. Cada amplitude vai soar para nossos ouvidos como um tom diferente e alguns são até imperceptíveis aos nossos ouvidos. Por outro lado, como o computador é um dispositivo baseado em números, para que ele armazene um som em sua memória e possa fazer qualquer processamento sobre ele (gravar, transmitir, mixar), ele deve antes representá-lo na forma de números. Ai que está a dificuldade. As intensidades possíveis de um som são tantas que se aproximariam do infinito. Para tornar essa grandeza mais clara, imagine que pudéssemos emitir a intensidade do som emitido por um pássaro. Se em terminado momento dissermos que essa intensidade tem valor 5. Logo em seguida um outro som é emitido, medidos e constatamos que sua intensidade é 4. Até aí tudo bem! Mas o pássaro poderá em seguida emitir diversos sons que estariam entre 4 e 5, como 4,23, ou 4,88938, ou até uma dízima periódica, como 4,6666… Um ser humano, mesmo que não consiga medir a intensidade do canto do pássaro, consegue ouvi-lo, apreciá-lo e até repeti-lo com uma certa proximidade com alguns assobios. Mas o computador não trabalha assim! Antes de tudo, um computador teria que discretizar esses valores medidos, ou seja, passá-los do domínio dos números reais para o domínio dos inteiros. Assim, o que era 4 permanece 4, o que era 5, continua como 5, mas o que foi medido como 4,23 é convertido para 4, e o que era 4,88938 e 4,666 são convertidos para 5. Dessa forma, o computador passa a tratar com números reais e finitos. Um canto de um pássaro (ou até de uma orquestra sinfônica) pode ser armazenado e processador pelo computador. Na [Figura 1.2, “Sinal Analógico versus Sinal Digital”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s05.html#fig_analog_digital) apresentada, a onda quadrada representa um sinal digital.

Mas perceba que o som emitido pelo pássaro teve que ser modificado. Ele antes era complexo, rico e cheio de detalhes. Agora se tornou algo mais simples e reduzido. Houve uma perda de informação ao passarmos o dado do analógico para o digital. Processo semelhante ocorre quando outras informações da natureza são passadas para o computador, como uma imagem através de uma foto, ou uma cena através de um vídeo. Parte da informação deve ser ignorada para que possa ser armazenada em computadores. Você deve estar se perguntando então, quer dizer que imagens e sons analógicos possuem mais qualidade do que digitais? A resposta rigorosa para essa pergunta é, sim! Mas uma resposta mais consciente seria, as vezes! Isso porque a perda causada pela digitalização pode ser reduzida até níveis altíssimos que modo que nem o ouvido, nem a visão humana serão capazes de perceber.

Como exemplo de dados analógicos podemos citar tudo o que vem da natureza, som, imagem, tato, cheiro, enquanto que digitais são todos aqueles armazenados por dispositivos eletrônicos digitais, como computadores, celulares e TVs (exceto as antigas analógicas). Se uma foto digital, por exemplo, possui menos qualidade do que uma analógica, por que todos procuram apenas máquinas fotográficas digitais, transformando as analógicas quase em peças de museu? A resposta está na praticidade. Os computadores só entendem informações digitais. Uma máquina fotográfica, mesmo com qualidade inferior, vai nos permitir passar as fotos para o computador, compartilhar com os amigos, aplicar edições e melhorias, ampliar e copiar quantas vezes quisermos. Tarefas que antes eram impossíveis com máquinas analógicas. O mesmo pode ser refletido para músicas, documentos e livros. O mundo hoje é digital, e não há como fugirmos disso!

## 1.6. O Transistor

O transistor é um componente eletrônico criado na década de 1950. Ele é o responsável pela revolução da eletrônica na década de 1960. Através dele foi possível desenvolver sistemas digitais extremamente pequenos. Todas funcionalidades de um computador são internamente executadas pela composição de milhões de transistores. Desde operações lógicas e aritméticas, até o armazenamento de dados em memórias (a exceção do disco rígido, CD, DVD e fitas magnéticas), tudo é feito pelos transistores.

Os primeiros eram fabricados na escala de micrômetros 10-6 metros). Daí surgiram os termos microeletrônica e micro-tecnologia. Depois disso deu-se início a uma corrida tecnológica para se desenvolver transistores cada vez mais rápidos, menores e mais baratos. Essa revolução dura até hoje, mas foi mais forte nas décadas de 1980 e 1990. Foi emocionante acompanhar a disputa entre as empresas norte-americadas Intel e AMD para dominar o mercado de computadores pessoais. A cada 6 meses um novo processador era lançado por um delas, tomando da concorrente a posição de processador mais rápido do mercado. Poucos eram aqueles consumidores que conseguiam se manter a atualizados com tantos lançamentos.

O princípio básico é utilizar a eletrônica (corrente elétrica, resistência e tensão) para representar dados e depois poder executar operações com eles. A forma mais fácil de fazer isso foi primeiramente limitar os dados a apenas dois tipos. Zero e um. O sistema de numeração binário é muito mais fácil de representar com dispositivos eletrônicos do que o decimal, por exemplo. O transistor possui dois estados. Ou ele está carregado, ou está descarregado, assim como uma pilha. Isso facilmente pode ser mapeado para o bit 1 (carregado) e o bit (0). O revolucionário, diferente de uma pilha, foi possibilitar que esse estado pudesse ser mudado eletronicamente a qualquer momento e de forma muito rápida.

Assim, com 8 transistores em paralelo, eu posso representar, por exemplo um número de 8 bits. Posso mudar seus valores mudando suas cargas, e posso ler seus valores chegando se cada um possui, ou não carga. Esse é o princípio básico de construção de uma memória.

De forma semelhante, é possível integrar transistores para que os mesmos executem operações lógicas e aritméticas. As portas lógicas estudadas por você em Introdução à Computação são todas fabricadas utilizando transistores.

Quanto menores são os transistores, mais dados podem ser armazenados por área. Ao mesmo tempo, transistores menores guardam menos carga. Isso torna mais rápido o processo de carregamento e descarregamento, que, por consequência, torna o processamento e armazenamento de dados muito mais rápidos também.

Com a evolução da nanoeletrônica, os transistores são tão pequenos que possibilitou a construção de memórias de 1GB (um giga byte) do tamanho da unha da mão de um adulto. Para ser ter uma ideia, 1 Giga é a abreviação de 109, ou seja, um bilhão. Um byte são 8 bits. Então, uma memória de 1GB possui, pelo menos, 8 bilhões de transistores. Os processadores também se tornaram bastante velozes com a miniaturização dos transistores. Os processadores atuais trabalham na frequência de GHz (Giga Hertz), ou seja, na casa de bilhões de ciclos por segundo (diferente de operações por segundo). Isso é muito rápido!

**Figura 1.3. Estrutura de um transistor tipo MOSFET**

|  |
| --- |
| images/fig1.png |

Na [Figura 1.3, “Estrutura de um transistor tipo MOSFET”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s06.html#fig_mosfet) anterior é apresentada a estrutura de um transistor MOSFET. Esse transistor é o mais utilizado para se construir sistemas eletrônicos digitais, como os computadores. O nome vem da abreviação de “Metal-Oxide Semiconductor Field-Effect Transistor”. Vamos ver o que significa cada palavra dessas, e isso nos ajudará a conhecer um pouco mais o MOSFET e sua relevância. O termo MOS (“Metal-Oxide Semiconductor”) vem dos materiais utilizados para compor um MOSFET, que são principalmente, óxido metálico e semicondutor.

Semicondutores são materiais que possuem propriedades que nem os permitem classificar como condutor, nem como isolante. Em algumas condições ele age como um isolante, e em outras, como um condutor. O semicondutor mais utilizado em transistores é o silício (símbolo Si na Tabela Periódica). Em condições ambientes, o silício age como um isolante, mas se misturado a outros materiais, ele pode se tornar um condutor até a intensidade desejada.

|  |  |
| --- | --- |
| [Nota] |  |
| O Silício se tornou tão importante que modificou toda uma região da Califórnia nos Estados Unidos na década de 1950, tornando-a uma das mais promissoras do mundo até hoje. Essa região abrigou e abriga as mais importantes empresas do ramo de projeto de computadores, como Intel, AMD, Dell, IBM e Apple, e depois de softwares que iriam executar nesses computadores, como Microsoft, Oracle e Google. Essa região é chamada de Vale do Silício. |

No transistor da [Figura 1.3, “Estrutura de um transistor tipo MOSFET”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s06.html#fig_mosfet) a cor verde representa um cristal de silício que foi dopado com cargas negativas. Já a cor vermelha, representa a parte que foi dopada com cargas positivas. Na situação normal uma corrente elétrica aplicada no Dreno (Drain) consegue produz percorrer o estreito canal negativo e seguir até a Fonte (Source). Nessa condição dizemos que o transistor está ativo. Porém, for aplicada uma tensão negativa na Porta (Gate), as cargas positivas da região P (em vermelho) serão atraídas para mais próximo da Porta, e isso irá fechar o canal por onde passava a corrente elétrica. Nesse caso, dizemos que o transistor está inativo.

Por que isso tudo nos interessa? Quando o transistor está ativo, ele pode ser visto com o valor 1, e quando inativo, ele pode ser visto com o valor 0. Assim, temos a menor memória possível de ser construída. Quando quisermos que ela guarde o valor 1, basta desligar a tensão da Porta e aplicar uma corrente no Dreno. Já quando quisermos que ele armazene o valor 0, precisamos aplicar uma corrente na Porta e fechar o canal. Então, a memória de 8 bilhões de bits, pode ser elaborada com 8 bilhões de transistores como esses.

Agora conhecemos o primeiro aspecto que faz dos transistores essenciais para o entendimento do computador. Eles são usados para a construção de memórias. Memórias feitas a base de transistores são chamadas também de Memórias de Estado Sólido. Mas há outras, não tão eficientes e miniaturizadas, como memórias ópticas e magnéticas. O importante percebermos é que quanto menores pudermos construir esses transistores, melhor. O processo de abertura e fechamento do canal não é instantâneo. Ele leva um curtíssimo tempo, mas quando somados os tempos de todos os bilhões de transistores, ele passa a se tornar relevante. Quanto menor ele for, mais estreito é o canal e, portanto, mais rápido ele liga e desliga, da mesma forma, menor será a distância entre o Dreno e a Fonte, levando também menos tempo para os elétrons deixarem o Dreno em direção à fonte. Isso tudo fará a memória mais rápida. Transistores pequenos também possibilitam que mais dados sejam armazenados por área. É por isso que hoje enormes capacidades de armazenamento são disponíveis em dispositivos tão reduzidos, como são os exemplos de pen-drives e cartões de memória.

Os transistores também são usados para executar operações lógicas e aritméticas. A carga retirada de um transistor pode servir para alimentar um outro e que, se combinados de forma correta, podem executar as operações lógicas básicas, E, OU, NÃO e as aritméticas, adição, subtração, divisão e multiplicação. Com isso, os transistores não apenas podem ser utilizados para armazenar dados, mas como executar operações lógicas e aritméticas sobre esses dados. Isso é fantástico e vem revolucionado todo o mundo. Não só na Ciência da Computação, mas como também em todas áreas do conhecimento. O que seria da humanidade hoje sem o computador? Sem o telefone celular? Sem os satélites?

## 1.7. A Lei de Moore

Durante os anos de 1950 e 1965, a industrias do Vale do Silício disputavam pelo domínio do recém-surgido mercado da computação e eletrônica. Naquela época ainda não havia surgido o termo TIC (Tecnologia da Informação e Comunicação), mas ele seria mais apropriado para definir o nicho de clientes e serviços que eles disputavam. Eles dominavam a produção de circuitos eletrônicos digitais, dominados pela Intel e AMD, a produção de computadores e equipamentos de comunicação, como a Dell, Apple, IBM, HP e CISCO, além da indústria e software e serviços, como a Apple, Microsoft e, mais tarde, a Google. A disputa era grande e nem sempre leal.

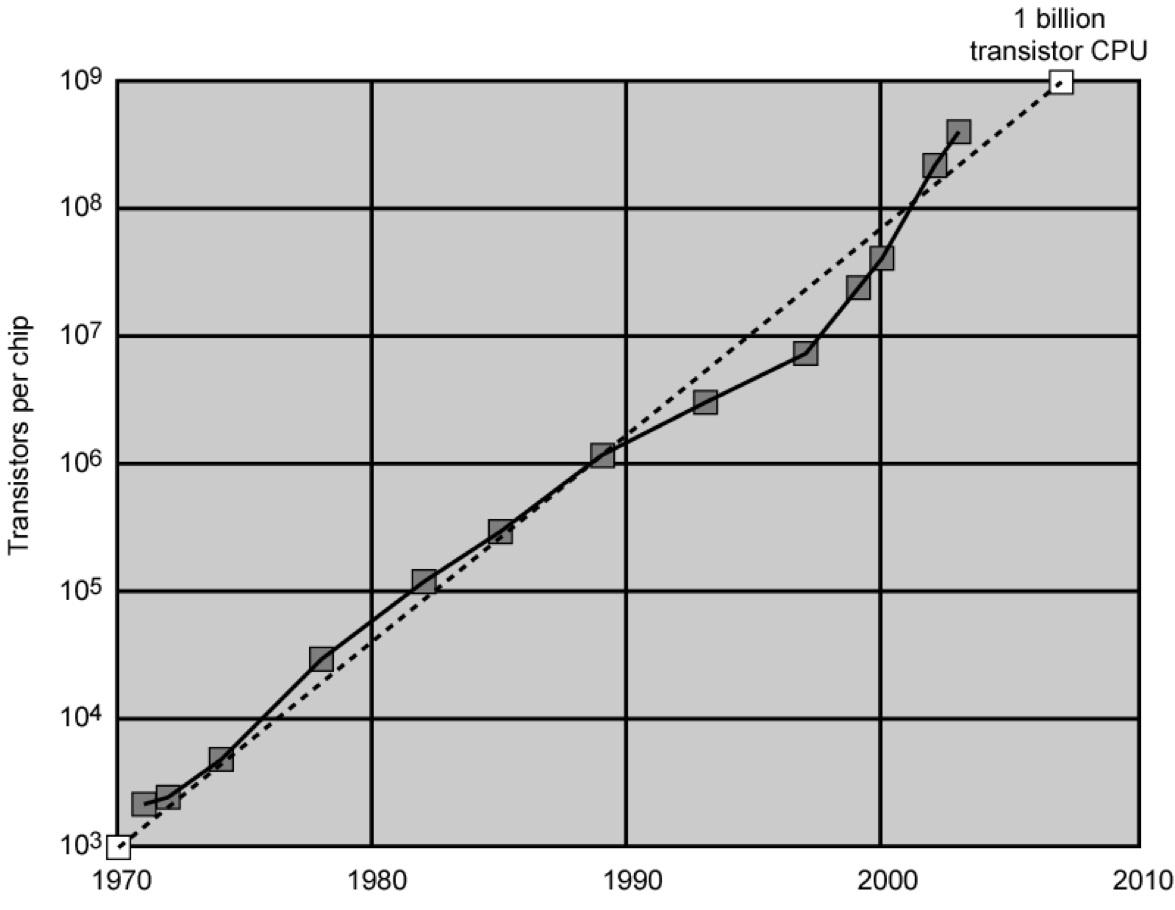
|  |  |
| --- | --- |
| [Nota] |  |
| Assista ao filme “Piratas do Vale do Silício” (1999) e tenha uma ideia de como essa guerra estava longe de ser limpa. |

Entretanto, não se sabia naquela época onde essa disputa ia parar, nem quem seriam os vencedores, nem mesmo, se haveria sequer vencedores. Até um dos sócios e presidente da Intel, Gordon Moore, lançou um trabalho minucioso onde ele destacava a experiência que ele adquiriu ao longe de alguns anos trabalhando na indústria de fabricação de processadores e circuitos para computadores. Ele percebeu que, sempre a indústria avançava em sua tecnologia e conseguia reduzir o tamanho de cada transistor de um circuito integrado, os computadores tornavam-se também muito mais velozes do que antes. Porém, essa redução no tamanho dos transistores requer uma total atualização nos equipamentos da indústria, tornando os equipamentos anteriores obsoletos. Assim, só seria viável a evolução para transistores menores se o lucro da empresa fosse o suficiente para pagar todas essas despesas. Por outro lado, ele também percebeu que os computadores e equipamentos mais obsoletos ainda possuíam mercado aberto em países menos desenvolvidos economicamente. Ele concluiu então que a indústria seria sim capaz de continuar evoluindo na redução do tamanho dos transistores porque os novos computadores, sendo tornando mais velozes, seriam tão mais eficientes e atrativos, que todos os clientes, principalmente as empresas, fariam de tudo para trocar seus computadores antigos por novos, afim de se tornarem cada vez mais produtivos.

Além dessa análise de mercado, ele analisou como o processo industrial era concebido e como os novos computadores se beneficiariam da redução do tamanho dos transistores. Ao final, ele afirmou que “A cada ano a quantidade de transistores por chip irá dobrar de tamanho, sem alteração em seu preço”. Essa frase pode ser interpretada também pelas consequências da quantidade de transistores por chip. Ou seja, a cada ano, com o dobro dos transistores, os chips se tornarão duas vezes mais rápidos. Um exemplo mais comum de chip são os processadores dos computadores. Então, por consequência, os computadores irão dobrar sua velocidade de processamento a cada ano, e ainda vão permanecer com o mesmo preço.

Naquela época essa era uma afirmação muito forte e ambiciosa. Muitos receberam esse estudo com cautela. Mas não demorou muito para todos perceberem que as previsões de Moore estavam se realizando. Foi tanto, e o trabalho dele foi depois chamado de “Lei de Moore” e ela ainda é válida até os dias de hoje. Na [Figura 1.4, “Evolução do número de transistores nos processadores em comparação com a Lei de Moore”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s07.html#fig_moore) a seguir é possível perceber como a quantidade de transistores por processadores cresceu dos anos 1970 até por volta de 2003 (linha contínua). É possível ver que ela não se afastou muito das previsões de Moore (linha tracejada).

**Figura 1.4. Evolução do número de transistores nos processadores em comparação com a Lei de Moore**



A Lei de Moore se tornou tão importante que ela não é usada apenas como uma meta a ser buscada e batida a cada ano, mas também como um meio para se verificar se a indústria está evoluindo na velocidade esperada. Apesar de Moore está muito correto em suas previsões, todos sabem, inclusive ele próprio, que esse crescimento não vai durar para sempre. Os transistores hoje estão na escala de 25 nanometros. Essa é a mesma escala de alguns vírus e bactérias. Reduzir mais do que isso está se tornando cada vez mais difícil. Pesquisadores e cientistas buscam outras formas de fazer com que os computadores continuem evoluindo em sua velocidade e reduzindo seu tamanho. Alguns pensam na substituição de transistores de Silício por outros materiais, como Grafeno. Outros até são mais radicais e defendem que a forma de computação deve mudar, talvez através de Computadores Quânticos ou de Bio-Computadores.

Quanto menores forem os transistores, mais rapidamente eles podem ser carregados e descarregados. Isso possibilita que o sistema trabalhe cada vez mais veloz. Mas há ainda outra limitação para a redução do tamanho dos transistores é a dissipação de calor. Quanto menores os transistores, mais deles são adicionados num mesmo circuito. O funcionamento dos transistores, como dito anteriormente, é feito através da passagem de corrente elétrica (elétrons em movimento). Como toda máquina elétrica, nem toda corrente é aproveitada. Muito dela é desperdiçada através da dissipação de calor. Então, uma vez que há milhões desses transistores trabalhando juntos, a dissipação de calor é ainda maior.

É muito importante para toda a humanidade que os computadores continuem evoluindo. A redução do tamanho dos computadores, aliada ao aumento de desempenho e sem o crescimento dos preços, permitiu que todas as ciência evoluíssem ao mesmo tempo, com a mesma velocidade. A metereologia, a medicina, as engenharias e até as Ciências Humanas avançaram sempre em conjunto com o avanço da computação. Para se ter um exemplo, foi a evolução dos transistores que permitiu que computadores se comunicassem numa velocidade tão grande que permitiu a formação da rede mundial de computadores, a Internet. Qualquer pessoa hoje consegue em poucos milissegundos fazer uma pesquisa por informações que estão do outro lado do planeta. Algo que antes só era possível viajando até bibliotecas distantes e cheirando bastante mofo e poeira. Hoje, ter em casa bilhões de bytes (Giga bytes) armazenados num minúsculo cartão de memória, é algo corriqueiro. A informação está hoje disponível numa escala tão grande e numa velocidade tão intensa que parece que mais nada é impossível para a humanidade. Após a Revolução Industrial do século XVIII que substitui os trabalhadores braçais por máquinas, o século XX, puxado pela evolução dos transistores, passou pelo o que muitos consideram a Revolução da Informação e o século XXI, já é considerado a “Era do Conhecimento”.

## 1.8. A evolução dos computadores

### 1.8.1. O ENIAC

O primeiro computador criado foi o ENIAC (‘Electronic Numerical Integrator And Computer’), desenvolvido por Eckert e Mauchly na Universidade da Pennsylvania, Estados Unidos. O projeto iniciou em 1943 financiado pelo governo americano. O período era da Segunda Guerra Mundial e o objetivo era poder calcular de forma mais ágil as melhores trajetórias para transporte de armas e mantimentos em meio aos exércitos inimigos. Esse é o tipo de cálculo que pequenos aparelhos celulares fazem hoje para encontrar rotas nas cidades através de GPS (‘Global Positioning System’) e análise de mapa. O projeto só foi concluído em 1946, tarde demais para ser utilizado para a Segunda Guerra, mas foi bastante utilizado até 1955.

O ENIAC ocupava uma área de 4500 metros quadrados, pesava 30 toneladas e consumia cerca de 140KW. Ele era capaz calcular 5000 somas por segundo. A programação era feita manualmente através da manipulação de chaves, ou seja, não havia linguagem de programação, nem compiladores ou interpretadores de comandos. O Sistema Operacional só surgiu bem depois e tomou o emprego de muitos funcionários chamados na época de operadores de computadores. Profissão hoje extinta! O ENIAC ainda não utilizada transistores, mas válvulas que, dependendo de seu nível de carga, representavam um número. Cada válvula precisava estar devidamente aquecida para funcionar corretamente, então o processo de ligar o ENIAC era trabalhoso e levava bastante tempo. Ele trabalhava com o sistema de numeração decimal, o que parecia óbvio naquela época, mas que depois dos transistores, se tornaram complexo demais e foi adotado o sistema binário.

Após a Segunda Guerra iniciou-se o período chamado de Guerra Fria, quando a espionagem, sabotagem e muito especulação reinava entre os países liderados pela União Sovitética e Estados Unidos. Prato cheio para os computadores. Possuir um computador que fosse capaz de decifrar mensagens codificadas dos inimigos era o sonho de consumo de todo general daquela época.

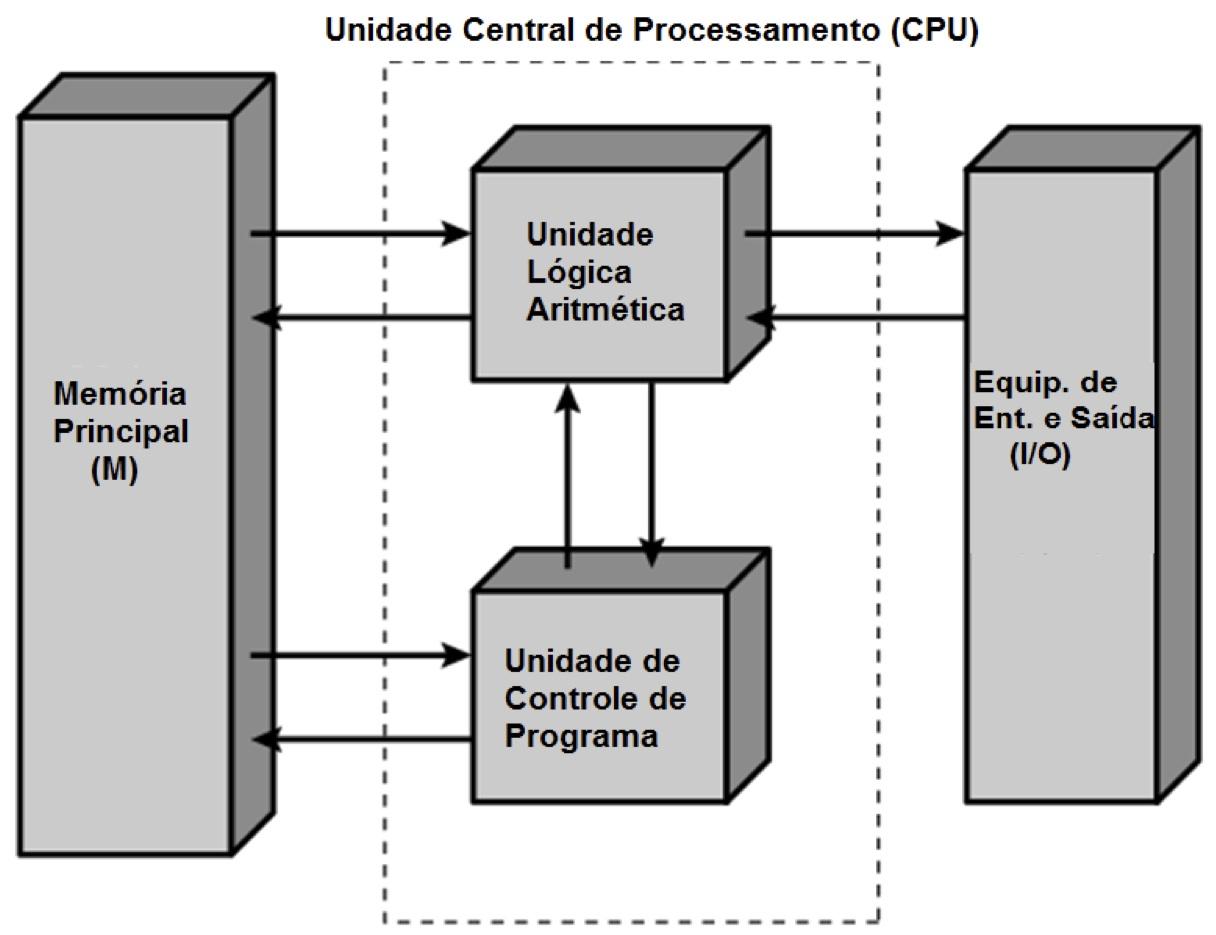
### 1.8.2. A Arquitetura de von Neumann

Muitas empresas e governos corriam para construir seu próprio computador que fosse mais avançado do que os anteriores. Muitos projetos surgiram depois do ENIAC. Mas todos eles eram barrados por algumas dificuldades e limitações. Como por exemplo, o fato de não serem programados e trabalharem com números decimais. O problema de trabalhar com decimais é que cada algarismo armazenado possui 10 estados possíveis, representando os números de 0 a 9. Dentro de um sistema eletrônico, isso é complicado por que a carga de cada dispositivo, seja transistor, seja válvula, deveria ser medida para verificar se que número ela estava representando. Os erros eram muito frequentes. Bastava que uma válvula estivesse fora da temperatura ideal para que os resultados das operações começassem a sair errado. Von Neumann recomendou em sua arquitetura que os dados e instruções fossem agora armazenados em binário, facilitando a análise dos mesmos e reduzindo a quantidade de erros.

Em 1952, o professor John von Neumann, da Univeridade de Princeton, Estados Unidos, apresentou um projeto inusitado para a arquitetura de um computador. Ele sugeriu que o computador fosse organizado em componentes, cada um executando apenas uma única tarefa e de forma muito mais organizada. Ele propôs que o computador fosse composto por (ver [Figura 1.5, “Estrutura da Máquina de von Neumann”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch01s08.html#fig_neumann)):

* Memória Principal: responsável por armazenar os programas a serem executados, assim como os dados a serem processados
* Unidade Lógica e Aritmética (ULA): para realização das operações lógicas e aritméticas
* Unidade de Controle: que, baseado nas instruções lidas da memória, enviaria sinais de controle para a ULA para que a mesma executasse as operações devidas
* Unidade Central de Processamento (CPU): que agruparia a ULA e a Unidade de Controle
* Unidade de Entrada e Saída: responsável pela comunicação com os periféricos do computador (teclado, monitor, memória externa etc.)

**Figura 1.5. Estrutura da Máquina de von Neumann**



A Arquitetura de von Neumann deu tão certo que todos os fabricantes começaram a segui-la. Os computadores utilizados até hoje em dia seguem os preceitos básicos propostos por ele. Muitos avanços surgiram, melhorias foram feitas, mas até hoje os computadores são formados por Unidades de Controle, CPU, ULA, memória e Unidades de Entrada e Saída. John von Neumann deixou um legado para toda a humanidade.

### 1.8.3. A IBM

A International Business Machines, ou apenas IBM, foi fundada em 1911 com o nome de Computing Tabulating Recording (CTR) e iniciou produzindo e comercializando calculadoras para empresas e empresários. Só em 1924 é que ela muda de nome para International Business Machines ou apenas IBM. Ela é uma das poucas empresas que sobreviveram a todos os avanços da computação e continua sendo uma potência mundial. Apenas em 1953 a IBM entra no mercado de computadores com o IBM 701, tendo sempre as grande organizações como alvos. Só muitos anos depois é que os computadores pessoais foram surgir. O IBM 701 trabalhava com cartões perfurados, ou seja, toda programação dele era feita através de uma perfuradora que marca buracos para representar o bit 1, e deixava ilesa uma área para representar o 0. O mesmo acontecia depois que os programas era lidos e processados. Uma folha de papel era perfurada pelo computador para representar o resultados das operações executadas. Não preciso nem dizer o que isso era trabalhoso!

Em 1955 a IBM lança o IBM 702 que agora não fazia apenas cálculos científicos, mas também aplicações comerciais, visando deixar de ser um equipamento apenas para cientistas, mas também para empresários. Depois desses vários outros computadores foram lançados nas séries 700. Essas máquinas ainda utilizavam válvulas para armazenar os dados. Só em 1957 é que surge a Segunda Geração de computadores, com a utilização de transistores. Isso tornou os computadores mais leves, baratos, velozes e mais energicamente eficientes. Os primeiros computadores dessa geração foram o IBM 7000 e o PDP-1, da DEC, empresa que não existem mais.

A IBM lança em 1964 o IBM série 360, substituindo os antigos computadores da série 7000. O IBM 360 inicia a primeira família de planejada de computadores. Isso significava que todos computadores seguintes da série 360 seriam compatíveis com os anteriores. Todos programas desenvolvidos ou adquiridos pelas empresas poderiam ser usados mesmo que a empresa substituísse os computadores pela geração mais nova. Isso tornou a IBM uma das empresas mais poderosas do mundo na época, com filiais e representantes em todos o continentes do planeta.

### 1.8.4. As gerações dos computadores

As gerações de computadores surgiram com a miniaturização dos transistores e sua integração em chips em escalas cada vez maiores. Podemos então ver as gerações dos computadores como:

* 1946 a 1957: computadores baseados em tubos de vácuo
* 1958 a 1964: surgimento dos transistores
* 1965: indústrias atingiram a integração de até 100 transistores num único chip
* 1971: chamada de Integração em Média Escala, com até 3000 transistores por chip
* 1971 a 1977: Integração em Larga Escala, com até 100.000 transistores por chip
* 1978 a 1991: Integração em Escala Muito Grande (VLSI), com até 100 milhões de transistores por chip
* 1991 até a atualidade: Integração Ultra-VLSI, com mais de 100 milhões de transistores por chip

### 1.8.5. Memórias de semicondutores

Em 1970, uma empresa chamada Fairchild desenvolveu pela primeira vez uma memória utilizando a mesma tecnologia utilizada para fabricar os processadores, os transistores. Isso possibilitou que memórias muito menores, mais rápidas e mais baratas fossem desenvolvidas. E melhor, elas poderiam ser inseridas muito próximas, e até dentro dos processadores, acompanhando sua miniaturização. E foi o que aconteceu. A medida que a tecnologia foi avançando e produzindo transistores cada vez menores, as memórias também foram encolhendo.

Os processadores tornaram-se cada vez menores e mais velozes, mas infelizmente o avanço não ocorreu também com a velocidade das memórias, mas apenas com o seu tamanho. Isso até hoje é um problema. Armazenamentos rápidos são muito complexos de fabricar e, por consequência, caros. Isso vem limitando o avanço da velocidade dos computadores, mas sempre os cientistas vêm encontrando alternativas para manter Gordon Moore e todos nós muito orgulhosos.

### 1.8.6. A Intel

A Intel Corporation, ou simplesmente Intel, surgiu nos Estados Unidos em 1968, como uma empresa focada no projeto e fabricação de circuitos integrados. Ela foi fundada por Gordon Moore (o mesmo da Lei de Moore) e Robert Noyce. Ela era inicialmente uma concorrente da IBM, mas logo se tornaram parceiras. A Intel fabricava os processadores e memória, e a IBM fazia a composição deles com outros componentes para montar os computadores.

Em 1971 a Intel lança seu primeiro processador, o 4004, que trabalhava com operações e dados de 4 bits. Foi uma revolução, pois todos componentes da CPU estavam num único chip. No ano seguinte eles lançam o 8008, já de 8 bits. Em 1974 é lançado o 8080, primeiro processador de propósito geral. Ou seja, com ela tanto era possível executar aplicações científicas, financeiras, gráficas e jogos. O mesmo princípio dos processadores atuais. Ele foi substituído pelo 8086 de 16 bit. O próximo foi o 80286 que já era capaz de trabalhar com uma memória de 16MBytes. O 80386 trabalhava com 32 bits e tinha suporte a multi-tarefas, ou seja, era finalmente possível executar mais de uma aplicação simultaneamente. Depois veio o 80486 com muito mais memória e bem mais rápido, além de um co-processador específico para aplicações matemáticas. A partir do 80286 as pessoas omitiam o 80 ao falar do processador, chamando-o apenas de 286, 386 e 486.

Em seguida veio a geração Pentium, focando cada vez mais na execução de tarefas paralelas, adicionando várias unidades de processamento e armazenamento de dados dentro processador. Agora os processadores não teriam apenas uma ULA ou uma memória dentro do processador, mas várias delas. Hoje estamos na geração dos processadores multi-núcleos, ou multi-cores, que nada mais são do que vários processadores replicados dentro de um mesmo chip e coordenadores por uma unidade única.

### 1.8.7. A Apple e a Microsoft

Em 1974 Steve Jobs e Steve Wosniak trabalhavam noites a fio para tentar, pela primeira vez, criar um computador que fosse voltado não a empresas, mas a pessoas também. Seria a ideia de um computador pessoal. Eles compraram todos componentes necessários para montar um computador, fizeram várias improvisações e inovações, acoplaram uma TV e um teclado. Wosniak, um gênio da eletrônica e programação, desenvolveu o software para controlar o computador e ainda alguns aplicativos, como uma planilha de cálculos e alguns jogos. Assim que o protótipo ficou prontos, Steve Jobs, eximiu negociador e vendedor, colocou o computador na mala de seu carro e foi visitar várias empresas para conseguir algum apoio financeiro para poder fabricar o produto em escalas maiores. Foi até na IBM, mas ouviu deles que o mercado de computadores pessoais não era promissor e o negócio deles era a produção de grandes computadores para empresas.

Assim que conseguiram o primeiro cliente, em 1976, Jobs e Wosniak fundaram a Apple e lançaram o Apple I. Um produto mais maduro e melhor acabado. Jobs sempre gostava de produtos de design diferenciado, que fossem não apenas eficientes, mas bonitos e, principalmente, fáceis de usar. Suas apresentações anuais de lançamento de novos produtos eram sempre aguardados com grande expectativa e especulações.

A IBM inicialmente também desenvolvia o Sistema Operacional e os programas que iriam ser executados por suas máquinas. Logo ela percebeu que poderia fazer parcerias com outras empresas e agregar ainda mais valor aos seus produtos. Foi aí que surgiu a Microsoft, liderada pelo seu fundador, Bill Gates, com o seu sistema operacionais MS-DOS. Não demorou muito para que todos computadores lançados pela IBM trouxessem também o MS-DOS integrados e eles. Depois surgiram as evoluções do MS-DOS, o Windows e suas várias gerações. A Microsoft se beneficiou bastante dessa parceria, já que todos a grande maioria dos computadores do mundo executavam seu sistema, as pessoas teriam que aprender e se familiarizar com seu sistema operacional. As empresas de desenvolvimento de aplicativos e jogos tinham que fazê-los compatíveis com o MS-DOS e Windows e foi aí que a Microsoft se tornou uma das líderes do mercado e, por muitos anos, a mais rica empresa do mundo.

Steve Jobs sempre acusou o Bill Gates de ter copiado dele o código principal para o funcionamento do primeiro sistema operacional Windows. Gates nunca negou. Eles sempre trocavam acusações e isso gerou muito assunto para a impressa e fanáticos por tecnologia. A verdade é que a Microsoft cresceu bastante e a Apple passou por vários apertos. Só no ano 2000, quando Jobs retornou à Apple depois de ter sido expulso da própria empresa que ele fundou, foi que as coisas melhoraram para a Apple. Eles lançaram produtos em outras linhas que não fossem computadores pessoais, como o iPod para ouvir música e o telefone celular iPhone. A Apple passou então a dominar o mercado de música online com sua loja de músicas, iTunes e o iPhone é o Smartphone mais vendido do mundo.

Steve Jobs seguia a filosofia não de fazer clientes, mas de criar fãs. E deu certo. Hoje há vários “Apple Maniamos” que compram seus produtos antes mesmo deles serem apresentados ao público. Nos dias em que esse livro está sendo escrito, a Apple ultrapassou a IBM e a Microsoft em valor, e é a empresa mais valiosa do mundo.

## 1.9. Recapitulando

Ao final desse capítulo vimos o que é a arquitetura de um computador e porque é tão importante estudá-la. Vimos que o transistor é o dispositivo básico para todo o funcionamento de um computador. Estudar seu funcionamento e sua evolução, é estudar a própria Ciência da Computação e a eletrônica digital. Depois de seu surgimento, os computadores foram possíveis e avançaram a medida que eles encolhiam de tamanho e aumentavam de velocidade, consumindo menos energia. Com a evolução dos computadores, cada vez mais rápidos, menores e mais baratos, toda a humanidade avançou na mesma velocidade. No próximo capítulo vamos estudar mais a fundo como os processadores funcionam. Como os programas são executados e o que é feito nos dias de hoje para que eles sejam cada vez mais eficientes.

**1.10. Atividades**

* Quais as quatro funções básicas que todos os computadores executam? Dê um exemplo de operação de cada uma delas.
* Quais os elementos básicos de um computador e quais as funcionalidades de cada um deles?
* Quais as diferenças entre um sinal analógico e um digital? Apresente os pontos fortes e fracos de cada um deles. Na sua opinião, qual dos dois sinais apresentam maior qualidade?
* Caracterize o que é uma Máquina de von Neumann
* O que são transistores? Quais as vantagens na concepção de computadores com o surgimento dos transistores?
* Por que quantos menores os transistores, mais velozes os computadores? Há desvantagens nessa miniaturização das máquinas? Quais?
* O que diz a Lei de Moore? Em sua opinião, há um limite para esse crescimento? Onde vamos chegar?
* Que outras técnicas podem ser utilizadas para aumento do desempenho dos processadores que não pela redução do tamanho dos transistores? Explique cada uma delas.

**Capítulo 2. Unidade Central de Processamento (CPU)**

**Objetivos do capítulo**

Ao final deste capítulo você deverá ser capaz de:

* Identificar os principais componentes de uma CPU
* Conhecer o funcionamento de uma CPU
* Saber como um programa é executado em ciclos
* Definir e explicar o que é uma interrupção e suas consequências
* Argumentar sobre aspectos que influenciam no desempenho de uma CPU

Nesse capítulo vamos estudar a parte mais importante de um computador, que é a Unidade Central de Processamento, ou UCP, ou, do inglês, CPU. A CPU é responsável não apenas por executar os programas contidos na memória, mas também de controlar todos os dispositivos de entrada e saída. Seu avanço ao longo dos anos tem permitido que programas fossem executados cada vez mais rapidamente. Hoje temos processadores de vários núcleos capazes de executar várias atividades ao mesmo tempo. São esses processadores e funcionalidades que iremos estudar nesse capítulo.

## 2.1. O que é um programa?

Nesse momento, você não apenas deve saber o que é um programa, como já deve ter até já escrito seus próprios programas e entendido um pouco como o computador funciona. Os programas são sequências finitas de passos que foram definidas por um programador para alcançar um objetivo específico. Cada passo desse programa é chamado de instrução. Não necessariamente, uma instrução escrita em uma linguagem de alto nível, como C, Java, Python, por exemplo, é diretamente transformada em uma instrução de máquina e armazenada em memória para execução da CPU. Na verdade, geralmente, uma instrução de uma linguagem de alto nível embute vários comandos e ações a serem executadas pela CPU. Essa é a principal razão da criação dessas linguagens de alto nível. O programador deve ter o menor trabalho possível ao escrever um programa. Ele deve se preocupar com o problema que está tentando solucionar, ao invés de memorizar dezenas de comandos de uma linguagem de máquina extensa e repleta de detalhes.

Após compilado, o programa de linguagem de alto nível é transformado em um programa apenas com instruções de máquina. Cada instrução de máquina contém apenas uma única operação a ser realizada pela CPU. Para ser executado, esse programa deve ser transferido para a Memória Principal. No princípio, um Operador de Máquina, copiava todas as instruções para a memória de maneira quase que manual. Hoje em dia essa operação é realizada pelo Sistema Operacional (Windows, Linux etc.). Assim que um usuário clica com o mouse, ou pressiona a tecla Enter do teclado solicitando que um determinado programa execute, o Sistema Operacional copia o programa para a memória e solicita que a CPU o execute.

Não podemos esquecer que a memória do computador apenas armazena números binários. Então, podemos dizer que um programa em linguagem de máquina é formado por instruções em binário. A cada instrução trazida da memória, a CPU lê seu código binário de operação para saber do que se trata, e inicia o processo de execução. Dependendo da operação, que pode ser de movimentação de dados, uma operação lógica, ou aritmética, ou uma operação de armazenamento de dados, a CPU envia ordens para que os outros dispositivos do computador atuem de forma a completar a operação. Essas ordens são enviadas através de pulsos elétricos passados por fios dentro do computador. Esses fios são chamados de **Barramento de Controle**.

### 2.1.1. Software X Hardware

O computador é composto por dois elementos, o software e o hardware. Tanto o hardware quando o software foram escritos por um programador, ou engenheiro, para se resolver um determinado problema. O início é sempre o mesmo. O profissional se depara com um problema e projeta uma solução algorítmica para ele. A diferença está na concepção. O hardware é concebido em chip, utilizando transistores interconectados. Uma vez elaborado, o hardware não pode mais ser modificado. Ele é uma solução rígida (do inglês, Hard) para o problema. Já o software é elaborado para ser armazenado numa memória e ser executado com um processador de propósito geral. Ele é uma solução flexível (do inglês, Soft) para o problema, já que o programador pode, a cada momento, modificar seu programa afim de torná-lo cada vez melhor.

Soluções em software são sempre mais lentas do que soluções equivalentes em hardware. Isso porque para executar um programa, cada instrução deve antes ser armazenada em memória, transferidas para a CPU (lembre-se que memórias são muito mais lentas do que CPUs) e, só então, ser executada pela CPU. Já as soluções em hardware não utilizam instruções, elas executam as operações diretamente.

Por outro lado, as soluções em software ganham em flexibilidade, já que os programas podem ser facilmente modificados. Já as soluções em hardware, não. Uma vez concebido, um hardware não pode mais ser modificado, ele deve ser descartado para dar lugar a uma versão mais nova. Isso torna projetos em hardware muito mais caros.

Para entender melhor, podemos citar alguns exemplos de implementações em hardware comumente utilizadas. Todas são escolhidas devido ao seu caráter de pouca necessidade de modificação, mas muito demanda por alto desempenho. Por exemplo, chips de criptografia para celulares (geralmente smartphones), processadores aritméticos para acelerar o cálculos, aceleradores gráficos para gerar gráficos mais rápidos, alguns chips para fazer edições rápidas em fotos, geralmente acoplados às câmeras digitais. As implementações são feitas em software quando a demanda por desempenho não é tanta, ao mesmo tempo em que as atualizações são frequentes, como os Sistemas Operacionais, os jogos e aplicativos em geral.

Apesar de não ser tão rápida quanto gostaríamos, a CPU é uma solução muito boa por permitir a execução de, praticamente, qualquer tipo de programa, se tornando uma máquina de propósito geral.

## 2.2. Estrutura de uma CPU

Toda CPU é formada por duas unidades, como podem ser vistas na [Figura 2.1, “Estrutura de uma CPU”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_CPU):

* Unidade de Controle (UC)
* Unidade de Ciclo de Dados (UCD)

**Figura 2.1. Estrutura de uma CPU**

|  |
| --- |
| images/cap2/fig_CPU.jpg |

A Unidade de Controle é responsável por receber instruções pelo Barramento de Instruções. As instruções vêm da memória de acordo com o endereço enviado pela UC para a memória através do Barramento de Endereço das Instruções (à esquerda da UC na [Figura 2.1, “Estrutura de uma CPU”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_CPU)). Já Unidade de Ciclo de Dados, como o próprio nome deixa entender, é responsável por tratar os dados propriamente ditos. A Unidade de Controle não executa as instruções. Ela as lê, decodifica e passa os comandos para a UCD determinando como as instruções devem ser executadas e com quais dados. Baseada nesses comandos, a UCD pode ir buscar os dados necessários na memória, executar as devidas operações e enviar o resultado de volta para a memória para ser armazenado. Tudo controlado de acordo com os comandos internos enviados pela Unidade de Controle, que por sua vez se baseia na instrução decodificada. Os dados lidos, ou enviados para a memória, são transmitidos através do Barramento de Dados. Os endereços são enviados para a memória através do Barramento de Endereço.

Tudo isso é controlado por um sinal síncrono de relógio (clock, do inglês). A cada batida do relógio a unidade sabe que deve executar um passo, passar os dados para quem deve, e se preparar para o próximo passo. Quanto mais rápido é o relógio mais operações por segundo o processador consegue executar e mais rápido pode se tornar. A velocidade do relógio é medida em frequência, utilizando a unidade Herz (abreviatura é Hz). Um Herz significa um passo por segundo. Os processadores atuais trabalham na faixa dos poucos GHz (leia-se Giga Herz), entre 1 GHz e 5 GHz. Um Giga Herz significa um bilhão de passos por segundo. É por isso que os computadores são tão incríveis. Eles não executam operações extraordinárias. Pelo contrário. Executam operações extremamente simples, como somas, subtrações e multiplicações, mas fazem isso numa velocidade incrível.

### 2.2.1. Os papéis dos barramentos e da memória

Saindo um pouco de dentro da CPU, podemos enxergar os barramentos e a Memória Principal, como é apresentado na [Figura 2.2, “Estrutura de uma CPU com barramentos”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_CPU_com_barramento). Para facilitar a visualização, os Barramentos de Dados e de Endereço são apresentados replicados, tanto do lado esquerdo, quanto do direito da figura.

**Figura 2.2. Estrutura de uma CPU com barramentos**

|  |
| --- |
| images/cap2/fig_CPU_com_barramento.jpg |

A comunicação da Unidade de Controle e da Unidade de Ciclo de Dados é feita sempre com a Memória Principal através dos barramentos. Os endereços são transmitidos sempre via Barramento de Endereços para a memória, sempre de forma unidirecional da CPU para a memória. Quando as instruções são transmitidas da memória para a Unidade de Controle, elas utilizam o Barramento de Dados. Isso porque as instruções são tratadas pela memória como um conteúdo como um outro qualquer. Ela não faz distinção entre dados e instruções. O mesmo Barramento de Dados é utilizado pela Unidade de Ciclo de Dados para receber os operandos das operações a serem realizadas e para enviar os resultados de volta para a memória.

Fica claro então a importância da Memória Principal. Todo e qualquer programa só poderá ser executado a partir dela. Quando você, por exemplo, deseja executar um programa de um pendrive conectado pela USB do computador, ele antes precisa ser copiado para a Memória Principal. Só então ele será executado. A memória precisa ser grande o bastante para armazenar a maior quantidade possível de programas, e também precisa ser rápida o suficiente para buscar os dados e enviá-los o mais rapidamente possível à CPU, e também salvá-los no menor tempo possível. A velocidade das memórias é determinada essencialmente pela tecnologia de transistores utilizada. Essa tecnologia é relacionada ao preço. Quanto mais rápidas, mais caras elas são.

### 2.2.2. Os registradores

Os registradores são memórias elaboradas com o mínimo de transistores possível, utilizando o que há de mais moderno em tecnologia de armazenamento. Elas são as memórias mais rápidas que podem ser construídas e por isso são também as mais caras. Por essa razão, elas aparecem numa quantidade muito pequena em um computador, na casa de alguns Kilo Bytes. Eles podem ser divididos em dois grupos. Os registradores de propósito geral, e os de propósito específico. Como o próprio nome diz, os primeiros podem ser utilizados pelos programas para quaisquer objetivos, já os segundos são específicos para algumas tarefas. Por exemplo, há um registrador na CPU para controlar se o processador deve continuar em execução, ou entrar em modo de espera por nova ordem. Se esse registrador receber um valor diferente de zero, o processador entrará em modo de espera, até que receba a ordem de modificar esse valor. Na [Figura 2.3, “Estrutura de uma CPU com registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_arq_detalhada) os registradores de propósito específico apresentados são:

* Program Counter (PC): Contador de Programas
* Instruction Register (IR): Registrador de Instrução
* Memory Address Register (MAR): Registrador de Endereço
* Memory Buffer Register (MBR): Registrador de Dados

**Figura 2.3. Estrutura de uma CPU com registradores**

|  |
| --- |
| images/cap2/fig_arq_detalhada.jpg |

O PC contém o endereço de memória que será utilizado para buscar a próxima instrução a ser executada pela CPU. Antes de executar qualquer instrução, a CPU envia o conteúdo de PC para a memória através do Barramento de Endereço, a memória envia o conteúdo da memória nesse endereço através do Barramento de Dados. Esse conteúdo é então armazenado no IR. Já o IR, que recebeu a instrução que veio da memória, tem o objetivo de guardar a instrução e passá-la para a Unidade de Controle, que é quem vai lê-la e tomar as decisões necessárias para para que ela seja executada pela Unidade de Ciclo de Dados. Por se tratarem do processo de busca de instruções, o PC e o IR ficam instalados na Unidade de Controle. O PC possui conexão direta com o Barramento de Endereços, e o IR com o Barramento de Instruções.

Com relação ao MAR e ao MBR, eles possuem funções análogas ao PC e IR, respectivamente, mas referentes a dados e não a instruções. Quando uma operação precisa ser realizada com algum dado que está na memória (e não em um registrador), o endereço desse dado é passado para o MAR. A CPU então passa o conteúdo de MAR para a memória através do Barramento de Endereço, que retornará o conteúdo da memória nesse endereço através do Barramento de Dados. O conteúdo trazido pela memória será armazenado em MBR. Só então o dado poderá ser utilizado para o processamento inicialmente planejado. O MBR e MAR possuem, respectivamente, conexões diretas com os Barramentos de Dados e de Endereços. Ambos são situados na Unidade de Ciclo de Dados, por serem utilizados nas fases de processamento das instruções.

O tamanho e quantidade dos registradores de uma CPU é uma das principais decisões de projeto. Se forem grandes demais, ou em quantidade maior do que a necessária, podem resultar em desperdício e aumento desnecessário no preço do processador. Já se forem pequenos, ou em pouca quantidade, com certeza vão tornar o computador muito mais lento do que o desejado. Encontrar o tamanho e quantidade ideais é trabalhoso e geralmente é feito através de simuladores e de muito testes e anos de experiência.

Os registradores de propósito geral são utilizados para guardar as variáveis dos programas. Como eles estão presentes em quantidades muito pequenas, são poucas as variáveis que ficam armazenadas em registradores. As demais ficam na Memória Principal. Quando uma operação precisa ser realizada e seus dados estão nos Registradores de Propósito Geral, a CPU não precisa buscá-los na memória e o processamento torna-se muito mais rápido.

|  |  |
| --- | --- |
| [Importante] |  |
| Lembre-se que as memórias são muito mais lentas do que os processadores! |

A CPU tenta ao máximo manter as variáveis mais utilizadas nos registradores. Ela faz isso guardando aquelas mais usadas nas últimas operações. Nem sempre isso funciona, mas no geral, é a melhor solução.

|  |  |
| --- | --- |
| [Nota] |  |
| Faça suas variáveis mais importantes serem bastante utilizadas. Usando-as em repetições, por exemplo. Isso aumentará as chances delas serem armazenadas em registradores, podendo acelerar a execução dos seus programas. |

### 2.2.3. Unidade Lógica e Aritmética (ULA)

A Unidade Lógica e Aritmética, ou ULA, se assemelha muito com uma calculadora convencional. Ela executa operações lógicas e aritméticas. As ULAs modernas executam operações tanto com inteiros, como com números reais. A ULA recebe como entrada dois diferentes dados que são trazidos para ela dos registradores (de propósito geral, ou específicos) (veja a [Figura 2.3, “Estrutura de uma CPU com registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#fig_arq_detalhada)). Quem decide que registradores passarão seus dados para a ULA é a Unidade de Controle baseada no tipo da instrução que está sendo executada. A Unidade de Controle também envia para a ULA qual operação será realizada (soma, multiplicação, divisão, AND, OR etc.). Assim que isso é feito, a ULA executa a operação e gera um resultado na sua saída. Esse resultado também é passado para um registrador escolhido pela Unidade de Controle, baseando-se na instrução em execução.

### 2.2.4. Unidade de Controle (UC)

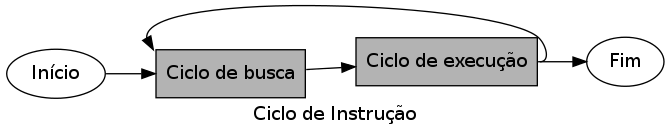
A Unidade de Controle, ao receber a instrução que está armazenada em IR, a decodifica e envia os sinais de controle para onde for necessário. Decodificar nada mais é do que ler um código em binário e interpretar a operação relativa a esse código. Dependendo da operação, os sinais de controle podem ser internos, por exemplo, para a ULA executar uma soma, ou para o conteúdo de um registrador ser transferido para a ULA. Ou pode ser externo, para um dispositivo de entrada e saída, por exemplo, ou mesmo para a Memória Principal. Tudo isso depende da instrução a ser executada.

Na próxima seção será apresentada a execução de instruções em mais detalhes, o que facilitará o entendimento do funcionamento das CPUs.

## 2.3. Ciclo de Instrução

Toda CPU trabalha em dois ciclos principais, o Ciclo de Busca e o Ciclo de Execução, como pode ser visto na [Figura 2.4, “Ciclo de Instrução”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s03.html#fig_ciclo_instrucao). Assim que o computador é iniciado, a CPU entra no Ciclo de Busca, em seguida passa para o Ciclo de Execução e depois volta para o Ciclo de Busca. Ela continua nesse processo até que precise ser desligada, saindo do Ciclo de Execução para o estado final.

**Figura 2.4. Ciclo de Instrução**



Durante o Ciclo de Busca, é a Unidade de Controle que atua. Uma nova instrução é buscada da Memória para que possa ser decodificada. Nessa fase os registradores [[PC]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#PC) e [[IR]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#IR) são utilizados, como apresentados na seção anterior. O PC é logo lido para se saber que instrução será executada, essa instrução é trazida para o IR e, finalmente, é decodificada pela Unidade de Controle. Assim que esse processo termina, caso a instrução não diga respeito à um laço, ou à uma repetição, o conteúdo de PC é incrementado. Ou seja, PC recebe PC + 1. Assim, no próximo Ciclo de Busca a instrução do endereço seguinte será carregada da memória e executada. Esse comportamento garante a característica de execução sequencial dos programas.

No passo seguinte a CPU entra em Ciclo de Execução. Nessa etapa atua a Unidade de Ciclo de Dados. Agora a Unidade de Controle já sabe exatamente que operação será executada, com quais dados e o que fazer com o resultado. Essa informação é passada para a ULA e os registradores envolvidos. Durante o Ciclo de Execução há cinco possíveis tipos de operação que podem ser executadas:

Processador e memória

trata simplesmente da transferência de dados entre CPU e memória principal;

Processador e Entrada e Saída

diz respeito à transferência de dados entre a CPU e um dispositivo de Entrada e Saída, como teclado, mouse, monitor, rede, impressora etc.;

Processamento de Dados

são operações simplesmente de processamento dos dados, como operação aritmética ou lógica sobre os registradores da CPU;

Controle

são instruções que servem para controlar os dispositivos do computador, como para ligar um periférico, iniciar uma operação do disco rígido, ou transferir um dado que acabou de chegar pela Internet para a Memória Principal;

Operações compostas

são operações que combinam uma ou mais instruções das outras em uma mesma operação.

### 2.3.1. Busca de Dados

Em operações entre Processador e Memória, é necessário que dados sejam trazidos da memória para servirem de entrada para a ULA, e/ou o resultado seja levado para armazenamento na memória no final da execução. Para isso acontecer, é executada uma Busca de Dados. Isso é determinado durante a decodificarão da instrução, no ciclo de Busca de Instrução. Isso acontece quando um dos parâmetros de uma operação aritmética é um endereço de memória, e não um valor diretamente, nem um registrador. Para isso, parte do conteúdo de [[IR]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#IR) é transferido para o [[MAR]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#MAR). Essa parte é justamente o endereço do parâmetro da instrução. Em seguida a Unidade do Controle requisita à memória uma leitura. Assim, o endereço, agora em MAR, é passado para a memória e o conteúdo lido da memória é passado para o [[MBR]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#MBR). Agora o conteúdo é transferido para a ULA para que a operação seja executada (lógica ou aritmética).

Se a instrução tiver dois ou mais parâmetros de memória, serão necessárias outras Buscas de Dados. Como a memória é sempre mais lenta do que a CPU, instruções que necessitam Buscas de Dados são muito mais lentas do que instruções de Processamento de Dados.

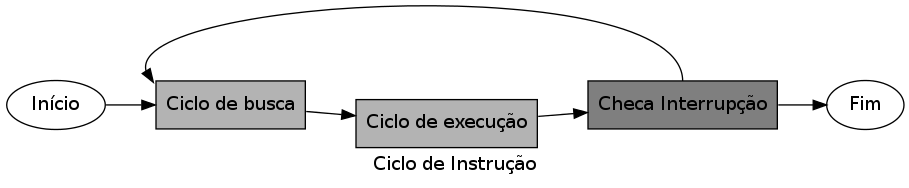
Perceba que cada instrução pode exigir mais tempo de execução do que outras. Isso depende de quantos acessos à memória ela exigirá. Quanto mais acessos à memória, mais lenta a instrução. O ideal é sempre usar registradores. Mas nem sempre é possível utilizar registradores. Eles estão sempre em poucas quantidades e em menores tamanhos. Principalmente por serem caros. O que os computadores sempre tentam fazer é passar os dados da memória para os registradores assim que puderem, para que as próximas instruções sejam aceleradas.

## 2.4. Interrupções

Além do ciclo básico de instrução apresentado anteriormente, a CPU pode ainda executar outro tipo de tarefa. Ela diz respeito ao processamento de pedidos oriundos dos dispositivos de Entrada e Saída. Como o Ciclo de Instrução da CPU que vimos até o momento é fechado, ou seja, a CPU sempre fica em estado de repetição até que seja desligada, ela não pode atender a nenhum evento externo que não seja a execução de um programa. Por exemplo, quando um usuário pressiona uma tecla do teclado, ou faz um movimento com o mouse, ou mesmo, quando uma mensagem chega pela Internet através da placa de rede. O que a CPU deve fazer? Se ela estiver em um Ciclo de Instrução fechado como mostrado anteriormente, nada. Ela precisa parar o que está fazendo para atender ao evento ocorrido e, só então, voltar ao Ciclo de Instruções. Esse processo de parar o Ciclo de Instrução para atender a um evento externo é chamado de Interrupção.

O Ciclo de Instrução pode agora ser visto modificado na [Figura 2.5, “Ciclo de Instruções com interrupções”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s04.html#fig_ciclo_com_interrupcao) para atender às Interrupções. Todas as interrupções são recebidas e armazenadas internamente por um dispositivo chamado Gerenciador de Interrupções. Esse dispositivo é um chip, semelhante à uma CPU, mas bem mais simples.

**Figura 2.5. Ciclo de Instruções com interrupções**



Na maioria dos computadores eles vêm soldados na Placa-Mãe, mas podem também vir dentro do chip da CPU. Toda interrupção possui um código de identificação. Sempre que uma nova interrupção chega nesse gerenciador, ele armazena esse código em sua memória e manda um sinal para CPU através do Barramento de Controle. Durante seu Ciclo de Instrução, sempre que uma instrução é executada, antes de voltar para o Ciclo de Busca, a CPU checa se algum sinal de interrupção foi enviado pelo Gerenciador de Interrupção.

Quando não há uma interrupção, a execução volta ao Ciclo de Busca e o programa em execução continua a ser executado. Mas se houver uma interrupção, a CPU agora vai parar a execução do programa atual para atender a interrupção. Por exemplo, vamos supor que o usuário tenha pressionado uma tecla do teclado. O código armazenado pelo Gerenciador de Interrupção indica que a interrupção veio do teclado. A CPU pára sua execução do programa anterior e vai iniciar a execução de um programa especial, o Tratador de Interrupção. O código do dispositivo (aqui seria o teclado) serve para a CPU saber o endereço do Tratador de Interrupção que ela vai buscar da memória. Então, ao sair da Checagem de Interrupção, a CPU muda o endereço do PC para o endereço do Tratador de Instrução. Assim, no Ciclo de Busca a próxima instrução a ser trazida da memória e posteriormente executada será a do tratador do teclado.

Cada tipo de interrupção precisa de um tratamento específico a ser feito. No caso do teclado, o tratador vai checar que tecla foi pressionada. Isso é feito através de uma leitura à memória do teclado (sim, todos os dispositivos possuem uma pequena memória) para saber que tecla foi pressionada. Dependendo da tecla, uma operação diferente será executada. Geralmente, a CPU adiciona o código da tecla pressionada num endereço específico de memória. Cada programa, lendo essa informação, tomará sua própria decisão sobre o que deve ser feito. O que acontece é que apenas o programa ativo no momento, vai ler esse conteúdo, executar a ação da tecla e limpar essa área de memória. Se o programa for um editor de texto, por exemplo, o código pode representar escrever a letra pressionada na posição atual do cursor dentro do texto.

Quando esse processo encerra, o tratamento é encerrado, e a CPU deve voltar à execução do programa que havia sido interrompido. Isso só é possível porque, antes de passar à execução do Tratador de Interrupção, a CPU salva os conteúdos de todos os registradores da CPU (inclusive o PC e o IR). Então, antes de devolver a execução para o programa, a CPU restaura todos os valores dos registradores antes salvos. Dessa forma, o programa retoma exatamente do ponto em que parou.

As interrupções também ocorrem se o próprio programa em execução executar uma operação ilegal. Isso é feito para evitar que a CPU entre em erro. Por exemplo, se um programa tentar acessar uma área da memória que é proibida para ele, como a área de outro programa ou do Sistema Operacional. Nesse caso, o programa é interrompido e não volta mais a executar, ele é finalizado e a execução é devolvida ao Sistema Operacional. Algo semelhante ocorre em caso de defeitos em alguns dispositivos. Por exemplo, se um programa estiver lendo um arquivo que está em um pendrive, e esse pendrive é removido subitamente, uma interrupção é lançada e o programa é encerrado, já que ele não faz mais sentido estar em execução.

## 2.5. Sobre o desempenho

É possível agora perceber que o desempenho das CPUs depende de muito outros fatores além da velocidade do seu clock. O computador precisa ter memórias rápidas para reduzir o tempo dos Ciclos de Busca, precisam de mais registradores para usar menos a memória e também que poucas interrupções ocorram. Cada vez que uma interrupção ocorre, o programa deve ser interrompido e a chamada deve ser atendida. Isso vai atrasar demais o tempo de execução dos programas, dando a impressão de baixo desempenho.

Basicamente, há dois tipos programas, os orientados à CPU e os orientados a Entrada e Saída. Na [Figura 2.6, “Execução com várias interrupções”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s05.html#fig_entrada_saida) o comportamento dos primeiros é mostrado na parte a) e o dos segundos na parte b).

**Figura 2.6. Execução com várias interrupções**

|  |
| --- |
| images/cap2/fig_entrada_saida.jpg |

Quando um programa é orientado à CPU, há momentos longos de processamento de CPU e curtos momentos de espera por um evento de Entrada e Saída. É o exemplo de programas que fazem muitos cálculos matemáticos, como ferramentas de simulação, projetos de engenharia, computação gráfica e planilhas de cálculos. Inicialmente os dados de entrada são passados por um dispositivo de entrada, há longos momentos de cálculos e depois os resultados são passados para um dispositivo de entrada e saída.

Já nos programas orientados à Entrada e Saída (b), que são aqueles chamados também de interativos, há muitos momentos de interação e uso de dispositivos de Entrada e Saída, e poucos momentos de uso de CPU. Esse é o caso de programas que utilizam muito o mouse e o teclado, como os jogos e a própria navegação na internet.

O que temos que ter em mente é que o desempenho de um computador está muito ligado ao perfil de cada usuário. Os Sistemas Operacionais são os responsáveis por escolher que tarefa colocar para executar a cada momento e por quanto tempo ela deve executar até que uma nova tarefa entre em execução. Assim, o papel do Sistema Operacional também é fundamental e determinante no desempenho do sistema. O que ele tenta fazer no máximo que pode, é tentar ocupar os tempos de espera de um programa com a execução de outro. Tarefa nada fácil!

## 2.6. Exemplo de execução de um programa

Suponha que queiramos executar uma instrução de máquina que soma dois números que estão na memória e salve o resultado em outro endereço de memória. Para tal, vamos indicar que a memória (M) se comporta como um vetor (um array) e entre colchetes indicaremos o endereço do dado, ou da instrução. Sendo assim, a instrução que gostaríamos de executar seria:

200: M[100] = M[101] + M[102]

Nesse caso, vamos ler que no endereço 200 da memória há uma instrução que precisa somar o conteúdo do endereço 101, com o conteúdo do endereço 102 e salvar o resultado no endereço 100 da memória. Supondo que M[101] contenha o valor 10, e M[102] contenha o valor 20, ao final da execução, o endereço 100 de memória (M[100]) deverá conter o valor 30.

Como uma instrução como essa será executada depende de cada arquitetura. Aqui vamos utilizar uma abordagem que quebra as instruções em pequenos passos simples, que facilitam o trabalho de decodificarão da CPU.

Sendo assim, esse programa seria transformado na seguinte sequência de instruções e executado.

PC = 200;

//Envia comando de leitura de instrução para a memória

IR <- (M[100] = M[101] + M[102]) // Busca instrução da memória

PC = PC + 1

//Instrução é passada do IR para a Unidade de Controle

A primeira ação seria realizar o **Ciclo de Busca**, visando trazer a instrução a ser executada da memória para o processador. O endereço da instrução (200) seria passado para o PC e um comando de leitura de instrução seria passado para a memória. Baseada no endereço trazido por PC, a memória localizaria a instrução e a enviaria para o processador, que a armazenaria no registrador IR. Antes de passar a instrução para a Unidade de Controle para dar início à execução, o registrador PC é atualizado para o próximo endereço de memória, no caso, 201.

O próximo passo será iniciar o **Ciclo de Execução**:

//O primeiro dado é trazido da memória para o registrador R1

MAR = 101

//Envia comando de leitura de dado para a memória

MBR <- 10

R1 = MBR // valor lido da memória é passado para o registrador R1

Como os dados a serem operados estão também na memória, é antes necessário executar uma operação de **Busca de Operando**, ou **Busca de Dado**. O primeiro operando está no endereço 101. Sendo assim, o endereço 101 é passado para o registrador de endereço (MAR). Esse endereço é passado para a memória e é enviado um comando de leitura de dado. O conteúdo, o valor 10, é então localizado pela memória e enviado para o processador, que o armazena no registrador de dados (MBR). Como o MBR será utilizado nas próximas etapas de execução, seu conteúdo é salvo em um registrador de propósito específico, o R1.

Em seguida, a Unidade de Controle passa para a busca do segundo operando, contido no endereço 102:

//O segundo dado é trazido da memória para o registrador R1

MAR = 102

//Envia comando de leitura de dado para a memória

MBR <- 20

R2 = MBR // valor lido da memória é passado para o registrador R2

Essa etapa ainda faz parte do **Ciclo de Execução**, e também diz respeito à uma **Busca de Dado**. A busca é mesma do passo anterior, mas agora o endereço buscado é o 102, e o conteúdo é o 20, que é repassado para o registrador R2.

O próximo passo do **Ciclo de Execução** é executar a operação aritmética propriamente dita. Isso geralmente é feito entre registradores de propósito geral, por serem mais rápidos do que se fosse tratar dados da memória. Os conteúdos de R1 e R2 são somados e armazenados em R3:

R3 = R1 + R2

Para finalizar o processo, o resultado deve ser armazenado de volta na memória:

MAR = 100 // Endereço é passado para MAR

MBR = R3 // Resultado da operação é passado para MBR

// Comando de escrita é passado para a memória

M[100] <- 30 // Endereço 100 da memória recebe o valor 30

Para isso ser realizado, é preciso executar uma operação de escrita na memória. O endereço 100 é então passado para MAR e o resultado da operação, salvo em R3 é passado para MBR. Quando o comando de escrita é enviado pela Unidade de Controle para a memória, ela lê o endereço 100 pelo Barramento de Endereço e o valor 30 pelo Barramento de Dados e salva, então, o valor 30 no endereço 100.

Com isso a operação é finalizada. Essa operação foi executada em aproximadamente 14 passos. Esse valor é aproximado porque alguns deles são apenas o envio de sinal para a memória, e isso geralmente é feito em paralelo com o passo seguinte. Se cada passo for executado dentro de uma batida do relógio (ou **ciclo de clock**), teremos 14 ciclos de clock para uma única instrução. Mas perceba que o acesso à memória é sempre mais lento do que a execução do processador. Se cada acesso à memória levar 3 ciclos de clock, teremos um total de 20 ciclos de clock.

|  |  |
| --- | --- |
| [Nota] |  |
| Apenas uma memória tipo Cache poderia ser acessada com apenas 3 ciclos de clock. Uma memória principal convencional precisa de entre 10 e 15 ciclos de clock para ser lida. Depende de sua tecnologia (e preço!). |

Parece bastante, mas algumas instruções podem levar muito mais ciclos do que isso, como operações com Ponto Flutuante (números reais), ou de acesso a um periférico, como o disco rígido. Isso depende muito de como o projeto do computador é elaborado.

Apesar do computador parecer pouco efetivo na execução de uma simples soma, como ele executa numa frequência de clock muito alta, ele acaba executando muitas operações por segundo. Então, utilizar apenas a frequência de clock como medida de desempenho não é uma boa ideia. O mais utilizado é medir a quantidade de operações aritméticas que o processador é capaz de executar por segundo. Hoje em dia um computador pessoal está na escala dos alguns Milhões de Instruções por Segundo (ou MIPS). Mais a seguir vamos estudar como essas e outras medidas de desempenho podem ser calculadas.

|  |  |
| --- | --- |
| [Nota] | **O que vem por aí** |
| Até o momento vimos como um processador básico trabalha. Nas próximas seções deste capítulo vamos ver como o desempenho pode ser aumentado, ainda mais, aumentando adicionando técnicas avançadas de execução paralela e de análise de programas. |

## 2.7. Aumentando o desempenho com Pipeline

Pelo o que foi visto, até o momento, a execução de um programa é essencialmente sequencial, ou seja, uma instrução só é executada quando a anterior termina. Ao longo do nosso curso veremos que há dois modos de paralelismo que podem ser utilizados para melhorar ainda mais o desempenho do processador. O primeiro deles é através do chamado **Paralelismo em Nível de Hardware**, que é obtido quando replicamos unidades do processador para que elas funcionem em paralelo, reduzindo assim o tempo de execução dos programas. A segunda forma é através do **Paralelismo em Nível de Instruções**, ou ILP (do inglês, *Instruction Level Parallelism*). Nesse caso, as unidades do processador não são duplicadas, mas melhores organizadas para que não fiquem ociosas. Há duas formas principais de implementar o ILP, uma delas é através do **Pipeline** e a outra é através de **processadores Superescalares**. Aqui vamos tratar do Pipeline, e no capítulo sobre Processamento Paralelo, vamos tratar as outras formas de paralelismo.

Imagine que possamos dividir o Ciclo de Instrução de um determinado processador nas cinco etapas seguintes:

Carregar instrução (FI)

Traz a instrução da memória para o processador, armazena em IR (essa etapa também é chamada de *Fetch de Instrução*) e a decodifica para execução no passo seguinte.

Carregar operandos (FO)

Traz os operandos da operação dos registradores para a ULA, para que a operação seja realizada sobre eles, também chamada de *Fetch de Operandos*.

Executar instruções (EI)

Executa operação lógica ou aritmética propriamente dita.

Escrever em memória (WM)

Escreve o resultado da operação em memória, se necessário.

Escrever em registrador (WR)

Escreve o resultado da operação em um dos registradores, se necessário.

Esse é um dos Ciclos de Instrução mais simples que poderíamos imaginar, organizado em apenas 5 etapas.

|  |  |
| --- | --- |
| [Nota] |  |
| Processadores convencionais, como os da Intel que usamos em nossos computadores, executam instruções em cerca de 18 etapas. |

Cada instrução deve passar pelos cinco passos descritos para ser executada. Suponha que cada etapa necessite de apenas 1 ciclo de clock para ser executada. Quantos ciclos seriam necessários para executar um programa de 20 instruções? Essa conta é simples. Cada instrução deve passar pelas cinco etapas, e cada etapa leva 1 ciclo de clock, sendo assim, o programa levará 20 vezes 5 ciclos, ou seja, 100 ciclos de clock.

Agora vamos analisar o que acontece com cada etapa a medida em que o programa é executado. A primeira instrução vai passar pela etapa **FI**, que a leva para o IR e a decodifica. Em seguida ela é passada para a etapa **FO**, e os dados necessários para a operação são levados dos respectivos registradores para a ULA. Agora, observe. Neste exato momento, a segunda instrução do programa está parada na memória, aguardando sua vez para ser executada. Ao mesmo tempo, a etapa **FI** está ociosa. Por que a etapa FI não pode entrar em ação e trabalhar com a segunda instrução do programa, enquanto a primeira está na etapa FO?

O mesmo vai ocorrer com todas as etapas de execução da primeira instrução do programa. Ela vai ser executada na etapa **EI**, depois vai passar para a etapa **WM** que checará se há necessidade de copiar o resultado para a memória e, finalmente, para a **WR**, que copiará o resultado para um dos registradores. Quando a primeira instrução estiver na etapa **WR**, as etapas anteriores estarão todas ociosas. Por que não aproveitar o tempo ocioso para colocar as etapas anteriores para irem adiantando a execução das próximas instruções? É isso que propõe o Pipeline!

O Pipeline vai separar as etapas de execução de instruções em unidades físicas independentes, assim, uma etapa pode trabalhar com uma instrução, ao mesmo tempo em que uma outra unidade trabalha com uma outra instrução. É a mesma estratégia utilizada pela indústria de produção em massa para fabricar carros, por exemplo. Enquanto um chassi está sendo montado, outro está recebendo a carroceria, outro o motor, outro sendo pintado e outro recebendo o acabamento interno. Todas etapas trabalhando em paralelo e vários carros sendo tratados ao mesmo tempo. Esta estratégia aumenta o desempenho da execução das instruções de forma grandiosa.

Na [Figura 2.7, “Processador adaptado para trabalhar com Pipeline de cinco estágios”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html#fig_pipeline) são apresentadas as adequações necessárias no processador para que as etapas possam ser organizadas em Pipeline. Dizemos então que esse processador trabalha com cinco Estágios de Pipeline.

**Figura 2.7. Processador adaptado para trabalhar com Pipeline de cinco estágios**

|  |
| --- |
| images/cap2/fig_pipeline.jpg |

A primeira mudança necessária é a separação da memória em duas partes independentes (ou duas memórias mesmo). Uma parte será utilizada apenas para instruções (representadas na figura pela palavra Fetch), e outra apenas para os dados (representada por DMem). Isso é necessário para que a etapa **FI** acesse a memória para buscar a próxima instrução, ao mesmo tempo em que a **WM** acessa a memória para salvar o resultado de outra instrução anterior. Se houvesse apenas uma memória para dados e instruções, isso não seria possível. Essa mudança vai contra o que foi projetado na ((Arquitetura de von Neumann)), e foi considerado um grande avanço. Ela foi batizada de Arquitetura Harvard.

Outra mudança importante foi a adição de memórias intermediárias entre cada etapa. Na [Figura 2.7, “Processador adaptado para trabalhar com Pipeline de cinco estágios”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html#fig_pipeline) essas memórias são representadas pelos retângulos preenchidos e sem nenhuma palavra sobre eles. Essas memórias são utilizadas para armazenar o resultado da etapa anterior e passá-lo para a etapa posterior no ciclo seguinte. Elas são necessárias porque as etapas não executam necessariamente sempre na mesma velocidade. Se uma etapa for concluída antes da etapa seguinte, seu resultado deve ser guardado nessas memórias para aguardar que a etapa seguinte conclua o que estava fazendo. Só então ela poderá receber o resultado da etapa anterior.

O mesmo ocorre na produção de um carro. A etapa de instalação do motor pode ser mais rápida do que a de pintura, por exemplo. Então, se um carro acabou de receber um motor, ele deve ser guardado num local temporário até que o carro anterior tenha sua pintura concluída. Assim, a etapa de instalação do motor pode receber um novo carro.

Qual o benefício da execução em Pipeline? Para isso, vamos analisar a [Figura 2.8, “Execução em pipeline de cinco estágios”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html#fig_pipeline2).

**Figura 2.8. Execução em pipeline de cinco estágios**

|  |
| --- |
| images/cap2/fig_pipeline2.jpg |

Nesse exemplo a dimensão horizontal (eixo X) representa o tempo, e a dimensão vertical (eixo Y) representa as instruções a serem executadas (I1, I2, I3, I4 e I5). Nessa imagem, a instrução I1 já passou por todas as etapas e está em WR, enquanto isso, I2 está em WM, I3 está em EI, I4 está em FO e I5 ainda está em FI. Como o Pipeline possui 5 estágios, ele precisa, no mínimo, de 5 instruções para encher o Pipeline e, a partir daí, inicia-se o ganho de desempenho.

Voltando ao exemplo anterior, considerando que cada etapa leve 1 ciclo de clock para ser concluída. Quantos ciclos são necessários para executar 20 instruções agora com Pipeline? No início, o Pipeline não está cheio, então a instrução I1 deve passar por todas as 5 etapas para ser concluída, levando então 5 ciclos de clock. Mas, a instrução I2 acompanhou I1 durante toda execução e terminou no ciclo seguinte, ou seja, em 6 ciclos de clock. Em seguida, a instrução I3 foi concluída em 7 ciclos de clock, I4 em 8 ciclos, I5 em 9 ciclos, assim em diante, até a conclusão da vigésima instrução, que ocorreu em 24 ciclos.

Comparado com o exemplo sem Pipeline, que executou o mesmo programa em 100 ciclos, o ganho foi de 4,17 vezes. Se o programa tivesse 200 instruções, levaria 1000 ciclos de clock sem Pipeline e 204 ciclos com Pipeline, o resultaria num ganho de 4,9 vezes. Onde queremos chegar com isso?

|  |  |
| --- | --- |
| [Importante] |  |
| A medida em que a quantidade de instruções aumenta, o ganho de desempenho com Pipeline vai se aproximando da quantidade de estágios, que foi 5 nesse exemplo. Então, quanto mais instruções forem executadas e mais estágios de Pipeline tiver o processador, maior será o benefício de usar o Pipeline. |

## 2.8. Limitações do Pipeline

Infelizmente, nem sempre o processador consegue usufruir do ganho máximo de desempenho ao usar o Pipeline. Há vários riscos que podem fazer com que o Pipeline seja interrompido e precise ser reiniciado, ou impossibilitado até de iniciar. Os riscos são:

* Riscos Estruturais
* Riscos de Dados
* Riscos de Controle

Os **Riscos Estruturais** são limitações físicas do processador. O exemplo mais simples é a separação da memória em Memória de Dados e Memória de Instruções. Se isso não ocorrer, as etapas de FI e WM não podem ser executadas ao mesmo tempo.

Já o **Risco de Dados** ocorre quando uma instrução depende do resultado de uma instrução anterior que está no Pipeline e ainda não está pronta. Imagine o trecho de programa a seguir:

I1: r1 = r2 + r3

I2: r4 = r1 - r3

A instrução I1 inicia primeiro e logo avança nas etapas do Pipeline. Logo depois dela vem I2. Quando I2 for buscar o valor de r1 na etapa FO, ele ainda não estará pronto, porque I1 ainda está em EI. A instrução I1 precisaria concluir a última etapa (WM) para que I2 pudesse executar FO. Nesses casos, dizemos que há uma *Dependência de Dados*. Isso cria uma bolha no Pipeline, o processador tem que avançar I1 até a conclusão de WM, e parar a execução de I2 e todas instruções seguintes até então. Só depois da conclusão de I1 é que o I2 e as próximas instruções seriam liberadas para continuar a execução.

O último tipo de risco é o **Risco de Controle**. Esse ocorre quando qualquer mudança no fluxo de controle do processador. Ou seja, quando a execução deixa de ser meramente sequencial. Como vimos anteriormente, o Pipeline vale a pena quando temos uma grande sequência de instruções sendo executadas. O processador confia que depois da instrução I1 ele executará a I2, depois a I3, e assim sucessivamente. Mas o que acontece, por exemplo, se o processador estiver executando a instrução I10, e essa instrução ordenar que o programa salte para a instrução I30? Isso pode ocorrer se a instrução se tratar de uma repetição, ou uma chamada a uma função. A mudança de controle também pode ocorrer por meio de interrupção, provocada por um dispositivo de entrada e saída, ou pelo pelo próprio Sistema Operacional, quando determina que um programa seja interrompido para passar a execução para um outro. Quando há uma mudança no fluxo de execução desta maneira, todas instruções que estão no Pipeline são removidas, o fluxo é modificado, e o Pipeline começa a ser preenchido todo novamente.

As técnicas de Pipeline avançaram bastante e várias medidas foram tomadas para amenizar o prejuízo causado pelos riscos mencionados. Entretanto, nenhuma técnica é capaz de evitar todas as possíveis perdas de desempenho. Até boas práticas de programação podem ajudar a otimizar a execução dos programas, e os compiladores também ajudam bastante neste aspecto.

Na próxima sessão vamos estudar um pouco como podemos realmente medir o desempenho dos processadores e entender melhor o que faz um processador mais eficiente do que outro.

|  |  |
| --- | --- |
| [Dica] | **O desempenho dos computadores** |
| O desempenho dos processadores e dos computadores é muito valorizado pelas empresas por agregarem muito valor a elas. Como se costuma dizer, tempo é dinheiro. Então quanto menos tempo se espera para um computador realizar uma tarefa, mais tempo resta para a empresa se dedicar a outras atividades. O desempenho é tão importante, que há uma corrida silenciosa entre empresas, universidades e governos para saber quem é capaz de produzir o computador mais rápido do mundo. A organização chamada Top 500 organiza uma competição para conhecer quem são esses campeões de desempenho e anualmente geral uma lista com os 500 computadores mais velozes. Atualmente a China e os Estados Unidos disputam o topo da lista do Top 500. Nas décadas após a Segunda Guerra Mundial os países disputavam uma guerra silenciosa (a Guerra Fria) para saber quem era o país mais poderoso em poder bélico e em tecnologias, como a corrida espacial.  Hoje a Guerra Fria já terminou, mas a corrida pela liderança tecnológica e econômica mundial continua e possui um novo concorrente forte disputando de igual para igual com os Estados Unidos, a China. Nessa corrida o poder computacional é importantíssimo! Ter um computador poderoso não significa apenas ser capaz de realizar tarefa mais rapidamente, mas ser também capaz de realizar certas tarefas que seriam impossíveis em computadores menos poderosos. Um exemplo disso é a construção de um computador que haja de forma semelhante ao cérebro humano. Chegar a esse ponto significa ser capaz de construir sistemas que possam substituir o homem em várias tarefas complexas, como dirigir máquinas e até mesmo operar computadores. Outro exemplo seria simular o comportamento perfeito da reação do corpo humano a drogas. Assim, não seria mais necessário o uso de cobaias para testar medicamentos. Esse avanço traria um poder incalculável a quem o dominasse!  Acesso o site da Top 500 em [http://www.top500.org](http://www.top500.org/) e conheça as super máquinas da computação! |

### 2.8.1. Medidas de desempenho

Para medir o desempenho dos computadores, três métricas principais são mais usadas:

Ciclos de Clock por Instrução (CPI)

determina quantos ciclos de clock são necessários para executar uma determinada instrução. Vimos que o Ciclo de Instrução é organizado em várias etapas e que isso depende de instrução para instrução. Se uma instrução acessar mais memória do que outra, ela será mais lenta. Instruções que operam com Pontos Flutuantes são mais lentas do que as operações com números inteiros. É fácil perceber a razão disso. Operações com números reais são mais complexas de serem resolvidas, porque devem ser realizadas para a parte fracionária e para a inteira, e depois o resultado deve ser consolidado. Assim, simulações são realizadas com um processador e são calculados quantos ciclos de clock cada tipo de instrução necessita em média para ser completada. Este é o CPI!

Milhões de Instruções por Segundo (MIPS)

o CPI é uma medida utilizada para medir o desempenho do processador para cada tipo de instrução, mas não é muito boa para medir o desempenho para a execução de programas, que é o objetivo de todo computador. Isso porque os programas são geralmente formados por instruções de todos os tipos, com inteiros, ponto flutuante, acessando muita ou pouca memória. Outro fator é que fatores como, Pipeline, tecnologia de memória e tamanho da memória Cache, podem fazer com que uma instrução seja executada lenta agora, e rápida logo em seguida. Para contornar isso, uma métrica muito utilizada é o MIPS. Ela determina quantos Milhões de Instruções são executadas pelo computador a cada segundo. Nesse caso, programas que demandam muito esforço do computador são executados, a quantidade de instruções é contada e depois dividida pela quantidade de segundos da execução. Caso o CPI médio (também chamado de CPI Efetivo) do computador já tenha sido calculado anteriormente, o MIPS pode ser calculado pela fórmula:

Onde, Clock é a frequência do relógio do processador em Hertz, CPI é o CPI médio e M é um milhão (). É necessário dividir o resultado por M porque a medida do MIPS é sempre dada na escala de milhões. Por exemplo, se um processador de 2 GHz e CPI médio de 4 ciclos por instrução, o MIPS desse processador será:

O resultado dessa operação será 0,5M, ou 500K. Isso porque 2 dividido por 4 é 0,5, e 1 Giga dividido por 1 Mega, resulta em 1 Mega.

|  |  |
| --- | --- |
| [Nota] |  |
| Lembre-se sempre de considerar a grandeza do Clock no cálculo. Um giga Hertz é muito diferente de um kilo, ou um mega Hertz! |

Milhões de Instruções em Ponto Flutuante por Segundo (MFLOPS)

uma alternativa para o MIPS é o MFLOPS. O MIPS é muito eficiente, mas não para comparar programas diferentes. Para calcular o MFLOPS, são executadas apenas instruções que operam com Ponto Flutuante e são calculados quantos segundos se passaram para cada milhão delas. Assim, ela pode ser definida da seguinte forma:

A única diferença para o cálculo do MIPS é que apenas as instruções que operam com Ponto Flutuante são consideradas. Assim, diz respeito a quantos ciclos de clock em média são necessários para executar uma instrução de ponto flutuante.

Os computadores pessoais e comerciais de hoje trabalham na escala MFLOPS.

Já os supercomputadores trabalham na escala de GFLOPS (Giga FLOPS). Aqueles computadores que lideram a lista do Top 500, e são capazes até de mudar o PIB de um país, trabalham na escala do TFLOPS (Teta FLOPS).

### 2.8.2. Exemplos de calcular o desempenho de um processador

Suponha que um programa é executado num processador de 40MHz. A [Tabela 2.1, “Exemplo de configuração de um processador”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s08.html#tabela_exercicio1) apresenta os CPIs coletados para cada tipo de instrução, bem como sua quantidade de instruções para um determinado programa com 100.000 instruções.

**Tabela 2.1. Exemplo de configuração de um processador**

|  |  |  |
| --- | --- | --- |
| **Tipo de instrução** | **CPI** | **Número de instruções** |
| Aritmética com Inteiros | 1 | 45.000 |
| Operações de acesso à Memória | 4 | 32.000 |
| Operações com Ponto Flutuante | 2 | 15.000 |
| Instruções de salto e desvio | 5 | 8.000 |

Para este exemplo, vamos calcular o CPI efetivo, o MIPS e o MFLOPS.

O CPI Efetivo é simplesmente a média ponderada dos CPIs apresentados para o programa. Isso pode ser feito da seguinte forma:

Já o MIPS pode ser calculado como:

Ou seja, para o programa examinado, o processador teve um desempenho de 16,46 milhões de instruções por segundo. Se o objetivo for calcular o MIPS geral, e não específico para esse programa, deve-se utilizar a média aritmética de todos os CPI calculados, e não a média ponderada.

Para calcular o MFLOPS seguimos a mesma estratégia, mas dessa vez utilizamos apenas o CPI para instruções de ponto flutuante. Ou seja:

Isso significa que esse processador apresentou um desempenho de 20 milhões de instruções de ponto flutuante por segundo.

## 2.9. Recapitulando

Vimos em mais detalhe neste capítulo como os processadores executam nossos programas, e como alguns aspectos de organização e arquitetura são importantes. Não apenas a frequência do clock do processador é relevante, mas como o Pipeline é aplicado, se há muitas interrupções e se o programa é bem escrito a ponto de explorar melhor o Pipeline. Também foram apresentados três diferentes métricas para se avaliar o desempenho de um sistema, o CPI, o MIPS e o MFLOPS.

No próximo capítulo vamos estudar mais profundamente a CPU através do estudo de sua Unidade de Controle e como ela faz para decodificar e executar instruções.

**Capítulo 3. Unidade de Controle**

**Objetivos do capítulo**

Ao final deste capítulo você deverá ser capaz de:

* Conhecer o funcionamento da Unidade de Controle em mais detalhes
* Apresentar como as instruções são executadas pela CPU

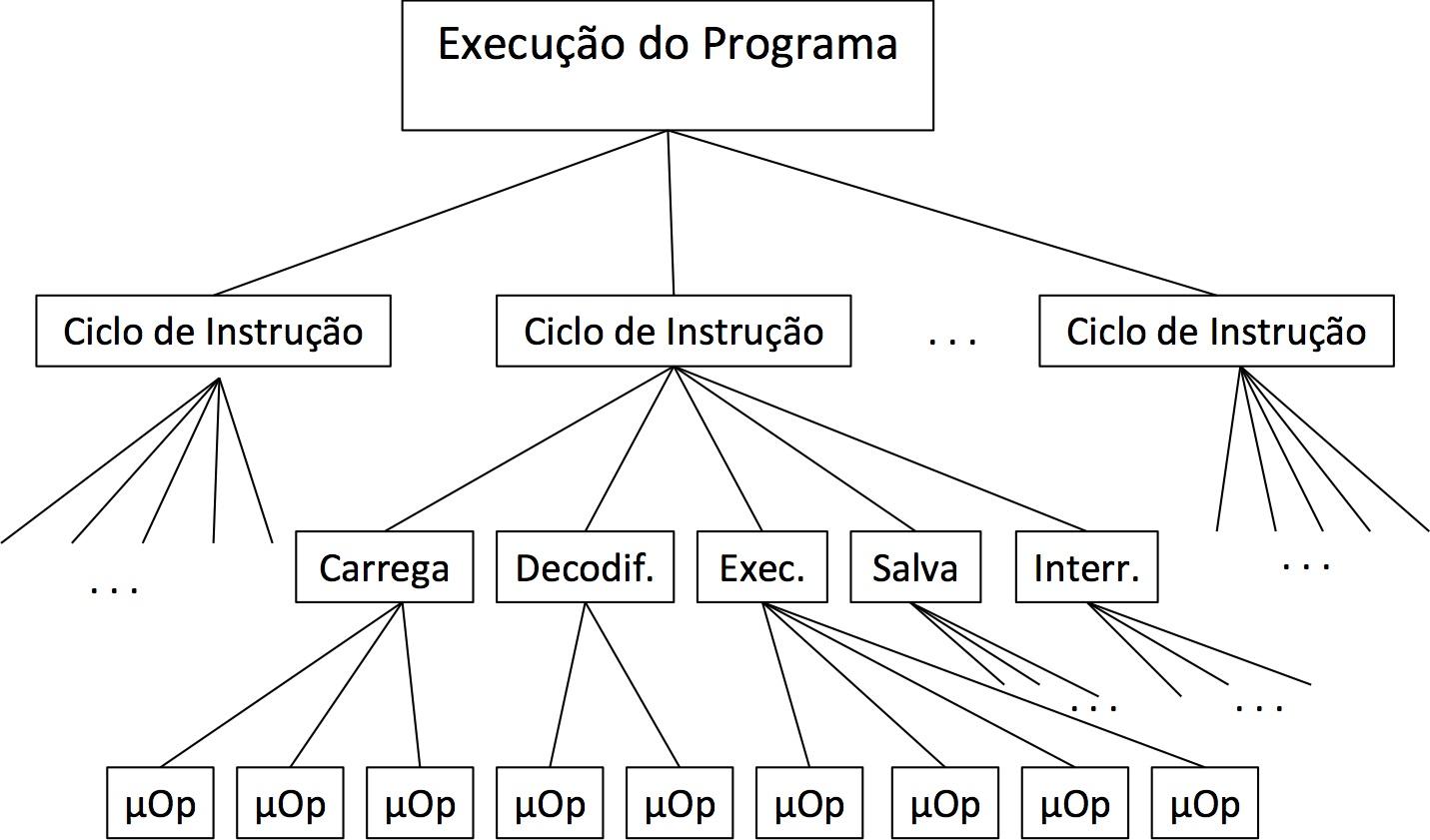
Neste capítulo vamos entrar mais a fundo no estudo das CPUs e vamos apresentar a Unidade de Controle, principal unidade dos processadores. Conhecer a Unidade de Controle é essencial para o entendimento de como as instruções são executadas pelo computador.

## 3.1. Introdução

A Unidade de Controle, como visto no capítulo anterior, é a unidade interna à CPU responsável (como o próprio nome já diz) pelo controle da execução das instruções. Como a CPU é uma máquina de executar instruções, a Unidade de Controle é quem controla o principal funcionamento do computador.

O projeto de uma Unidade de Controle varia de arquitetura para arquitetura, mas há alguns pontos que são comuns a todas elas. Toda Unidade de Controle trabalha com a execução de Microoperações. Como pode ser visto na [Figura 3.1, “Execução em Microoperações”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s01.html#fig_execucao), um programa é sempre executado instrução por instrução. Essas seriam instruções de máquina, compiladas a partir de Assembly, ou antes a partir de uma linguagem de alto nível e depois compiladas para Assembly. Cada instrução é executada através de um Ciclo de Instrução, como visto no capítulo anterior. Nessa ilustração o ciclo de instrução foi apresentado em cinco estágios (Carrega instrução, decodifica, executa, salva resultados e checa interrupção). Mas ele pode variar de acordo com a arquitetura da Unidade de Controle e também de acordo com o tipo de instrução. Uma instrução com vários parâmetros que estão na memória, por exemplo, pode necessitar de um estágio a mais antes da execução para buscar os dados na memória. Já outra que utiliza apenas dados de registradores pode omitir esse passo e executá-lo no próprio estágio de execução.

**Figura 3.1. Execução em Microoperações**



## 3.2. Microoperações

Ainda na [Figura 3.1, “Execução em Microoperações”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch03s01.html#fig_execucao) pode ser visto que cada estágio do Ciclo de Instrução é quebrado em sub-estágios, chamados de Microoperações. A Microoperação é a operação atômica (indivisível) realizada pela Unidade de Controle para a execução de cada estágio. Cada vez que um estágio do Ciclo de Instrução for executado, as devidas Microoperações são executadas pela Unidade de Controle.

Essa organização é necessária para melhorar a organização da CPU e facilitar o projeto das Unidades de Controle. Além disso, nos projetos de Unidades de Controle modernas, é possível reutilizar Microoperações de um estágio em outro. Por exemplo, para executar uma instrução, é necessário executar uma Microoperação para buscar um dado que está em um registrador (ou memória) para a ULA. Se houver vários dados envolvidos, a única tarefa a ser realizada é pedir para a Unidade de Controle executar a mesma Microoperação de buscar dado, mas agora com endereço diferente.

Dessa maneira, todas instruções que chegam à Unidade de Controle são quebradas em estágios (lembre-se do Pipeline), que por sua vez, separados em sequências de Microoperações e, só então, são executadas. Agora vamos apresentar alguns exemplos de como os principais estágios de execução são organizados em Microoperações.

### 3.2.1. Busca de Instrução

Neste estágio uma nova instrução deve ser buscada da memória e trazida para a CPU para que possa ser decodificada e executada em seguida. O endereço da instrução a ser buscada está sempre no Contador de Programa (PC) e a instrução buscada da memória é armazenada no Registrador de Instrução (IR). Vamos então apresentar como seriam as Microoperações para realizar a Busca de Instrução.

A seguir cada Microoperação é apresentada ao lado de uma determinação do tempo em que ela será realizada. Nesse caso, três unidades de tempo são necessárias (t1, t2 e t3). Imagine que cada unidade de tempo é 1 ciclo de clock (as microoperações mais complexas levam mais tempo do que isso).

t1: MAR <- (PC)

Memória <- fetch;

t2: MBR <- (memória)

PC <- (PC) + 1

t3: IR <- (MBR)

No tempo t1 o endereço guardado no registrador PC é copiado para o registrador MAR, que é o registrador conectado ao Barramento de Endereço. Ao mesmo tempo, a CPU envia para a memória um sinal de “fetch”, indicando que precisa ser trazida uma nova instrução. No tempo t2 a memória lê o endereço contigo no Barramento de Endereço (que é sempre o mesmo de MAR), busca a instrução e a escreve no Barramento de Dados. Como o MBR sempre reflete aquilo que está no Barramento de Dados, o MBR agora contém a instrução trazida da memória. Esta tarefa é realizada essencialmente pela memória, a ULA está livre para executar outra Microoperação, e aproveita para adicionar 1 ao endereço do PC. Esta operação de incremento vai garantir que a próxima instrução a ser buscada será a do endereço seguinte da memória. Finalmente, no tempo t3, a instrução que foi trazida da memória e salva em MBR pode ser agora salva em IR.

A instrução precisa sempre estar em IR porque na faze de decodificarão, a Unidade de Controle vai buscar lá que instrução deve executar.

Observe que a mesma Microoperação poderia ser executada de forma diferente, como mostrada a seguir.

t1: MAR <- (PC)

Memória <- fetch;

t2: MBR <- (memória)

t3: PC <- (PC) +1

IR <- (MBR)

Nesta segunda opção, o incremento de PC foi passado para o tempo t3, ao invés de t2, o que gera o mesmo resultado.

A quantidade de Microoperações que podem ser executadas no mesmo ciclo de clock depende da arquitetura do processador. Por exemplo, se há apenas uma ULA, então só é possível executar uma única operação lógica, ou aritmética a cada ciclo de clock. O mesmo serve para o uso acesso à memória com MAR e MBR. Se houver um barramento exclusivo para instruções, separado do barramento de dados e uma memória de instrução separada da memória de dados (quase todos processadores hoje possuem), e então é possível buscar uma instrução no mesmo ciclo de clock que se busca um dado.

### 3.2.2. Busca indireta

Outro tipo de Microoperação muito utilizado é a Busca Indireta. Ela trata de buscar um dado que está na memória e trazer para a CPU para ser utilizado pela instrução. O termo “indireta” indica que o dado não está diretamente na CPU (em um registrador) mas na memória.

Imagine que a instrução em questão seja a soma de 2 números A e B, ambos na memória. Esta instrução foi buscada no estágio anterior, portanto no ciclo seguinte ela estará no registrador IR. Então, os endereços de A e B estão presentes na instrução que agora está em IR. A Busca Indireta deve então ser realizada para A e depois para B, da seguinte forma:

t1: MAR <- (IRendereco)

Memória <- read

t2: MBR <- (memória)

t3: ACC <- (MBR)

No primeiro instante t1 o endereço do dado contido em IR é passado para o registrador de endereço MAR. Ao mesmo tempo a CPU envia para a memória um sinal de leitura (read), avisando que deve ser feita uma busca indireta. No instante seguinte, t2, o conteúdo do dado é trazido da memória para o MBR através do Barramento de Dados, e no último passo, em t3, o conteúdo agora em MBR é levado para um registrador para que seja utilizado na operação, geralmente o Acumulador (ACC).

### 3.2.3. Execução

Após a busca da instrução e dos dados necessários, é hora de executar a instrução na Microoperação de execução. Mantendo o exemplo da soma A e B apresentado anteriormente, a Unidade de Controle terá que fazer a Busca Indireta de A e B para depois realizar a soma. Supondo que A seja salvo em ACC, ele deve ser transferido para outro registrador, digamos R1 antes da Busca Indireta por B. Assim, a execução seria:

t1: R1 <- ACC

// Busca indireta por B:

t2: MAR <- (IRendereco)

Memória <- read

t3: MBR <- (memória)

t4: ACC <- MBR

t5: ACC = R1 + ACC

Em t1 o conteúdo de A salvo em ACC será transferido para o registrador R1. Nos intervalos de t2 até t4 seria feita a Busca Indireta por B. E, finalmente, no instante t5 a soma de R1 e ACC seria realizada e salva no acumulador ACC.

### 3.2.4. Salvar resultado

Após esse último passo, o conteúdo de ACC com o resultado da operação deve ser transferido para o local de destino. Isso é feito no estágio de Salvar Resultado. Se o resultado for salvo num registrador, a operação é direta e feita num único ciclo de clock. Mas se precisar salvar o resultado na memória, uma escrita indireta deverá ser realizada para salvar o conteúdo de ACC na memória.

t1: MAR <- (IRendereco)

t2: MBR <- ACC

Memória <- write

Para tal, inicialmente em t1 o endereço da variável de memória que precisa ser salva é passado para o MAR. O conteúdo de ACC é passado para o MBR no ciclo seguinte (t2), ao mesmo tempo em que a CPU envia para a memória um sinal de escrita. Ao receber esse sinal, a memória traz o conteúdo de MBR e o salva no endereço representado por MAR.

### 3.2.5. Salto condicional

Uma instrução muito comum executada pelos computadores são os saltos condicionais. Ela indica que se uma determinada condição for satisfeita, a execução não deve ser a da instrução seguinte, mas a indicada pela instrução. Imagine uma instrução “Salta se zero”, com dois parâmetros, X e Y. O X seria a variável a ser testada e o Y o endereço para onde a execução deve saltar caso o valor de X seja 0.

Desta forma, as microoperações seriam as seguinte:

// Busca indireta por X:

t1: MAR <- (IRenderecoX)

Memória <- read

t2: MBR <- (Memória)

t3: ACC <- (MBR)

t4: se ACC == 0, PC = (IRenderecoY)

Inicialmente, de t1 a t3, seria buscado o conteúdo de X na memória. No último ciclo t4, o conteúdo de ACC seria comparado com 0 e se forem iguais, o conteúdo de PC será o endereço da variável Y, também presente em IR. Observe que, caso contrário, nada precisa ser feito, o PC continuará como antes e a próxima instrução depois da atual será executada.

**3.3. Tipos de Microoperações**

Como já foi possível observar através dos exemplos apresentados, há quatro tipos básicos de Microoperações de uma Unidade de Controle. São eles:

* Transferência de dados entre registradores
* Transferência de dados de registrador para o exterior da CPU
* Transferência de dados do exterior da CPU para um registrador
* Operação lógica e aritmética

A transferência de dados de um registrador para outro é a mais simples das Microoperações e geralmente é feita num único ciclo de clock. Já a movimentação de dados de ou para o exterior da CPU pode ser mais complexa. Para facilitar muitos computadores mapeiam todos dispositivos de Entrada e Saída com se fossem memória. Ou seja, para a CPU, enviar um dado para um dispositivo seria como escrever um dado na memória, bastando usar um endereço diferente. Isso facilita bastante a operação da Unidade de Controle, mas pode limitar a quantidade de endereços de memória disponíveis para os programas. As operações de transferência de dados são complexas também porque levam um tempo não conhecido para serem executadas. Se o dado estiver na memória Cache o acesso é mais rápido, se estiver na Memória Principal levará mais tempo, e se tiver num dispositivo externo, um Disco Rígido, por exemplo, pode levar ainda mais.

As operações lógica e aritméticas podem ser mais rápidas ou mais lentas dependendo de cada uma delas. Operações com números de Ponto Flutuante tendem a levar mais tempo do que aquelas com números inteiros. Já as operações trigonométricas são as mais lentas que o computador pode operar.

## 3.4. Decodificação

A execução das Microoperações é sempre ordenada pela Unidade de Controle. Isso é feito no estágio de Decodificação, a partir da leitura da instrução presente em IR. O primeiro passo da decodificação é ler o código da instrução para conhecer seu tipo. Dependendo do tipo, a instrução é quebrada numa quantidade específica de Estágios e cada estágio no seu respectivo grupo de Microoperações. Cada vez uma que Microoperação se encerra, a Unidade de Controle checa qual será a próxima e envia os sinais para os devidos registradores, para ULA e dispositivos envolvidos, como memória ou dispositivos de Entrada e Saída.

Dessa forma, podemos dizer que a Unidade de Controle possui duas funções principais, a execução e o sequenciamento das instruções. Nessa última função a Unidade de Controle deve saber o exato momento que uma Microoperação concluiu para executar a próxima, e quando a última Microoperação for executada, iniciar um novo Estágio, e quando o último Estágio for concluído, executar a próxima instrução do programa.

## Glossário

**PC (Program Counter)**

Contador de Programas ([Seção 2.2.2, “Os registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_registradores)).

**IR (Instruction Register)**

Registrador de Instrução ([Seção 2.2.2, “Os registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_registradores)).

**MAR (Memory Address Register)**

Registrador de Endereço ([Seção 2.2.2, “Os registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_registradores)).

**MBR (Memory Buffer Register)**

Registrador de Dados ([Seção 2.2.2, “Os registradores”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_registradores)).

**ULA**

Unidade Lógica e Aritmética ([Seção 2.2.3, “Unidade Lógica e Aritmética (ULA)”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_ULA)).

**UC**

Unidade de Controle ([Seção 2.2.4, “Unidade de Controle (UC)”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s02.html#sec_UC)).

**FI**

Ciclo carregar instrução. Traz a instrução da memória para o processador, armazena em [[IR]](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/go01.html#IR) (essa etapa também é chamada de *Fetch de Instrução*) e a decodifica para execução no passo seguinte. ([Seção 2.7, “Aumentando o desempenho com Pipeline”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html))

**FO (Fetch operands)**

Clico carregar operandos. Traz os operandos da operação dos registradores para a ULA, para que a operação seja realizada sobre eles, também chamada de *Fetch de Operandos*. ([Seção 2.7, “Aumentando o desempenho com Pipeline”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html))

**EI (Execute Instructions)**

Ciclo executar instruções. Executa operação lógica ou aritmética propriamente dita. ([Seção 2.7, “Aumentando o desempenho com Pipeline”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html))

**WM (Write to memory):: Ciclo escrever em memória**

Escreve o resultado da operação em memória, se necessário. ([Seção 2.7, “Aumentando o desempenho com Pipeline”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html))

**WR (Write to register)**

Ciclo escrever em registrador. Escreve o resultado da operação em um dos registradores, se necessário. ([Seção 2.7, “Aumentando o desempenho com Pipeline”](http://producao.virtual.ufpb.br/books/edusantana/old-arq/livro/livro.chunked/ch02s07.html))

