Fundamentos de Arquiteturas de Computadores

Prof. Marcos Quinet
Universidade Federal Fluminense – UFF
Instituto de Ciência e Tecnologia – ICT

Motivação

- Através do estudo da estrutura funcional e organizacional de um ***sistema computacional***, o aluno será capaz de identificar os componentes que o compõe, a função de cada um deles, e a interação necessária para seu funcionamento
  - Um sistema computacional é o mesmo que um computador?

- Todas as áreas da Ciência da Computação baseiam-se no processamento de informações, que são realizados através de operações básicas, extremamente simples, que serão estudadas no decorrer da disciplina

Objetivo do Curso

- Toda a base da Ciência da Computação está na compreensão do funcionamento básico de um computador, que é realizado através de operações extremamente simples, mas que são utilizadas na implementação de poderosos e complexos sistemas computacionais

- Estudaremos como funcionam e qual o propósito dos componentes que, operando em conjunto, definem o que classificamos como um computador

Bibliografia

- Introdução à Organização de Computadores – Mário A. Monteiro; ed. LTC
- Introdução à Ciência da Computação – Guimarães/Lages; ed. LTC
- Arquitetura e Organização de Computadores – Willian Stallings; ed. Prentice Hall
- Organização Estruturada de Computadores – Andrew S. Tanenbaum; ed. LTC
- Arquitetura de Computadores – Hennessy e Patterson; ed. Campus

- Material das aulas: http://www.professores.uff.br/mquinet

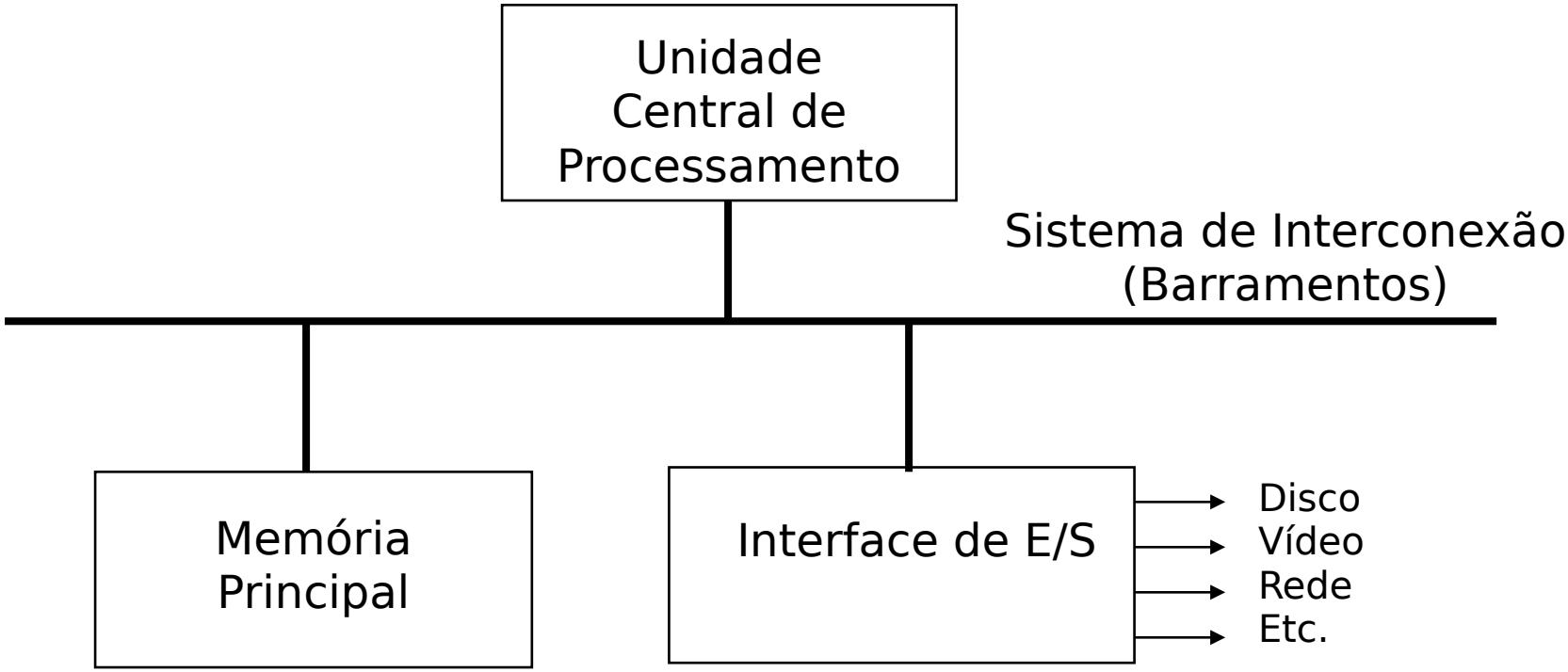
Introdução

- A arquitetura de um sistema computacional estabelece um modelo da organização e funcionamento de um sistema de processamento, com todas suas partes, divididas em **seções**, interagindo entre si

- Os componentes e suas relações são representados através de sistemas hierárquicos, modelo ideal para o estudo de conjuntos complexos e que atuam em diferentes níveis

  - Classificados de acordo com suas características, estudaremos o funcionamento de cada um dos componentes

Introdução – Componentes Básicos



Componentes Básicos

- Todo sistema de computação é constituído, pelo menos, dos seguintes componentes:
  - Um módulo que realize as operações necessárias, a unidade central de processamento;
  - Uma área de trabalho para o armazenamento das informações que serão processadas, a memória principal;
  - Dispositivos para o recebimento de informações e retorno/armazenamento dos resultados, os dispositivos de entrada e saída de dados;
  - E finalmente, um meio através do qual os dispositivos possam se comunicar e transmitir dados, os barramentos de comunicação.

Abstrações

- Tanto o *hardware* quanto o *software* são constituídos de níveis hierárquicos; o nível mais alto esconde os detalhes de funcionamento dos níveis mais baixos

- Utilização dessas camadas → **abstrações**

- A Interface entre os dois níveis de abstração, *hardware* e *software* básico caracteriza a arquitetura do conjunto de instruções da máquina

- Uma interface abstrata permite que várias implementações executem o mesmo *software*

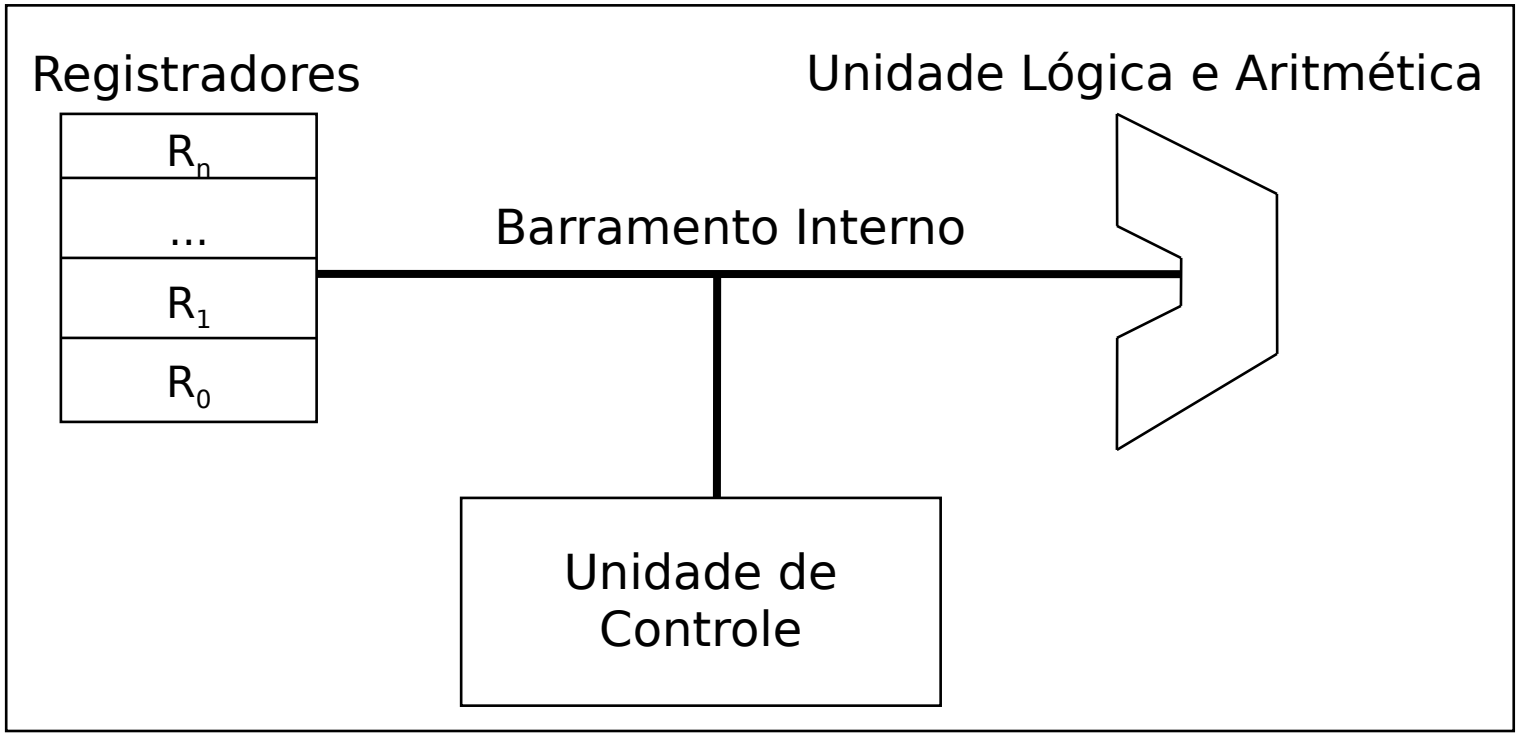
Subsistemas

- Ao estudarmos qualquer assunto complexo e/ou com grande riqueza de detalhes, a estratégia mais natural para facilitar o processo de aprendizado é subdividir o tema principal em partes menores, e então, preocupar-se somente com uma porção por vez

- Após estudadas todas as partes, o passo final é enxergar como estas se encaixam, tratando agora o objeto de estudo como um todo e abstraindo, se possível, os detalhes mais específicos de cada uma das partes

Exemplo de Subsistema

- Um exemplo (ainda bastante simplificado) de alguns componentes presentes na Unidade Central de Processamento (UCP ou CPU):

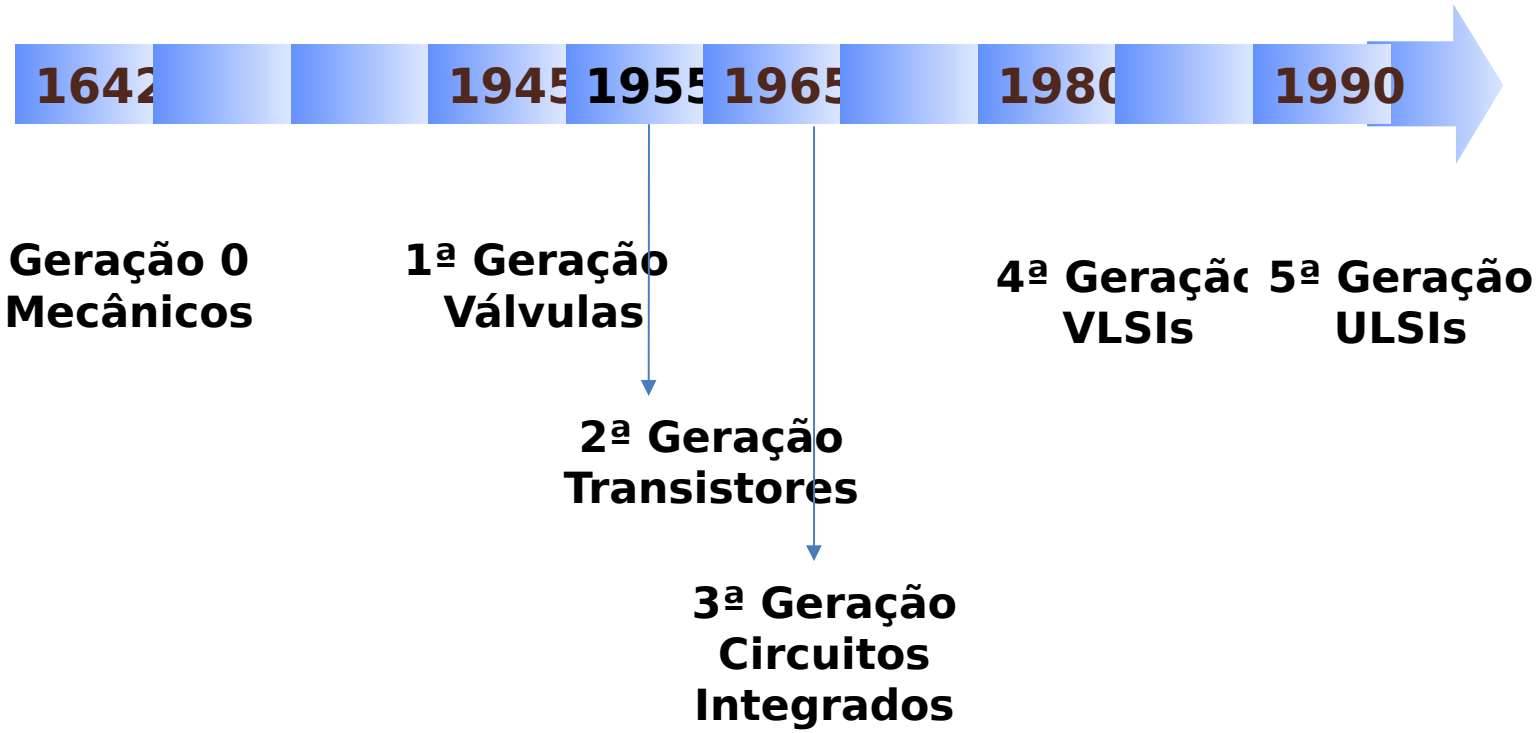


História dos sistemas computacionais

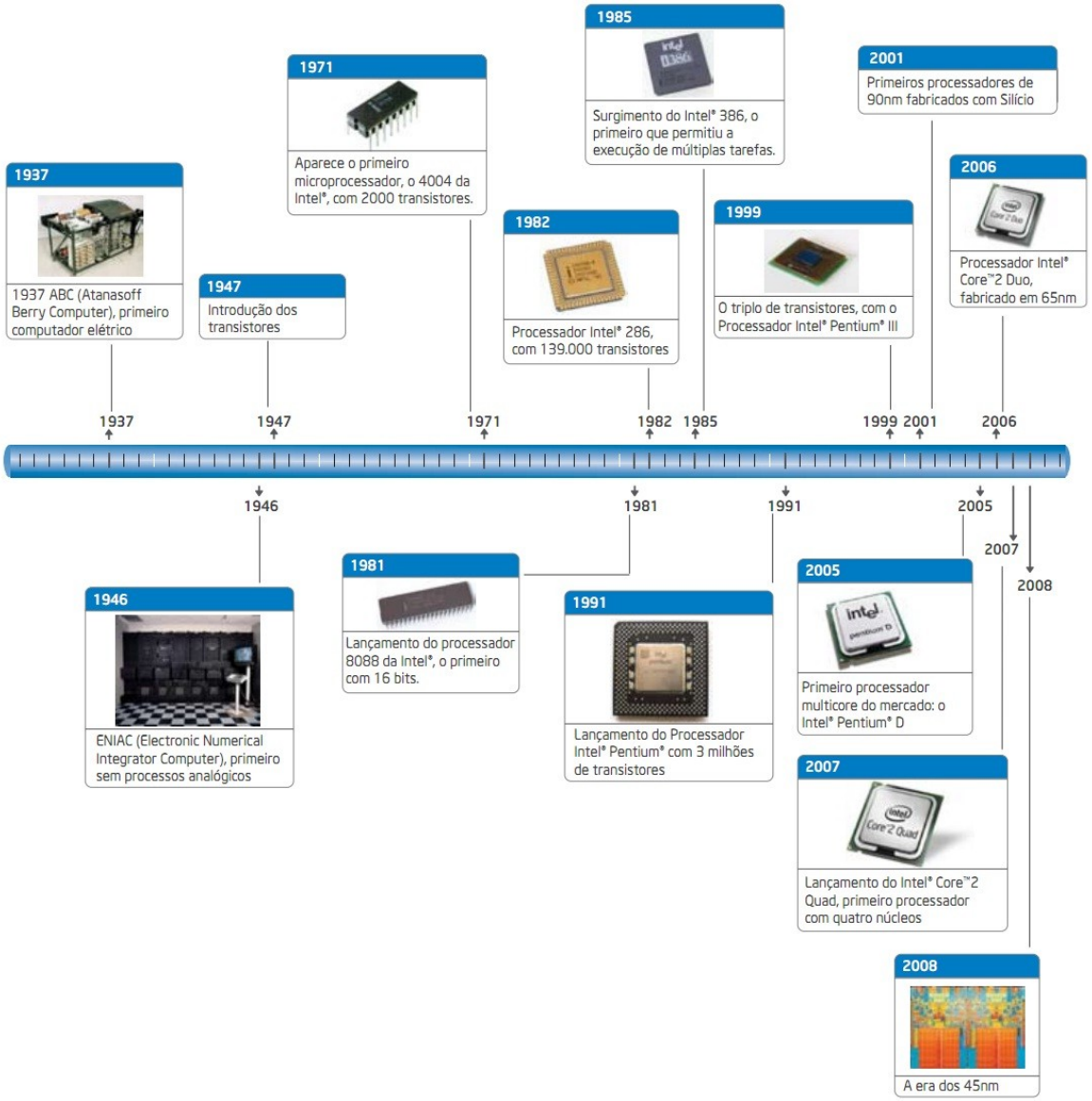
- Ao estudar a história da computação, nossa meta não deve ser somente situar historicamente o momento de criação destas “máquinas de calcular”, e sim observar a linha evolutiva que trilharam (além de ser extremamente interessante observar os avanços tecnológicos colossais em curtíssimos períodos de tempo).

- O mais importante é condicionar a mente a desenvolver IDEIAS!!! Devemos aprender a pensar, a criar recursos para atender nossas necessidades e contribuir, melhorando o que já existe.

Histórico


1642
1945
1955
1965
1980
1990
Geração 0
Mecânicos
1ª Geração
Válvulas
2ª Geração
Transistores
3ª Geração
Circuitos
Integrados
4ª Geraçã
VLSIs
5ª Geração
ULSIs

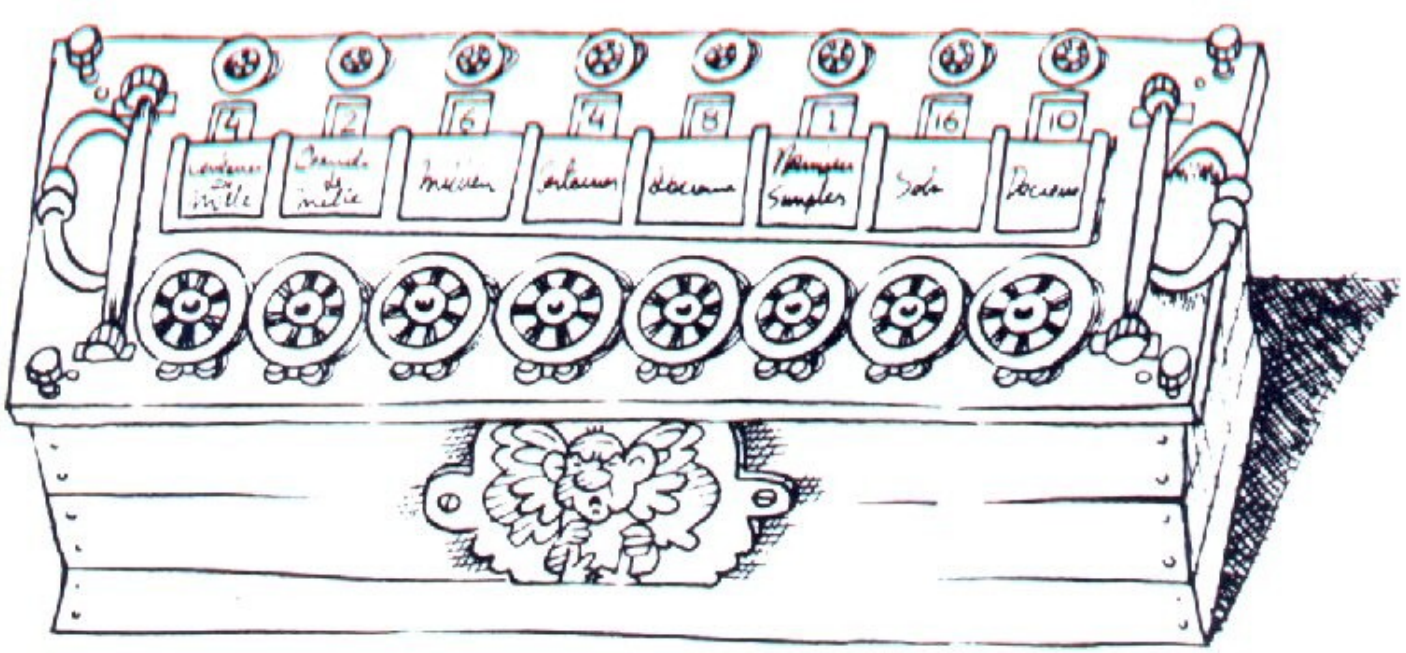
Histórico dos processadores



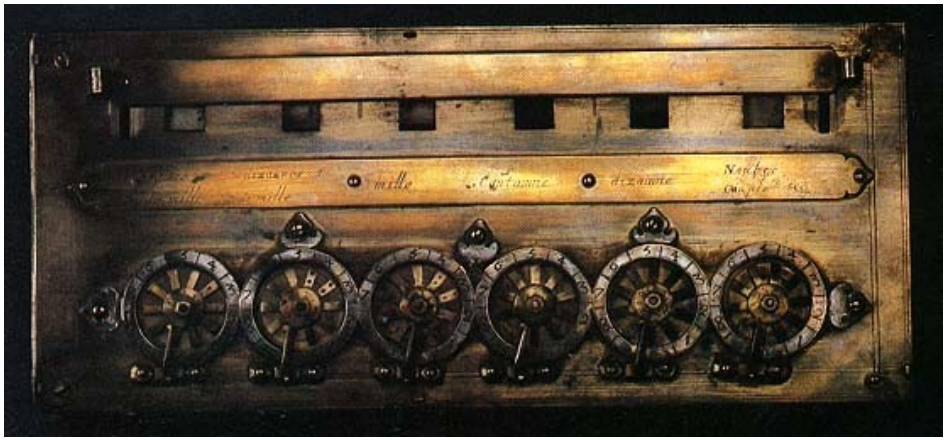
Geração 0 - Mecânicos

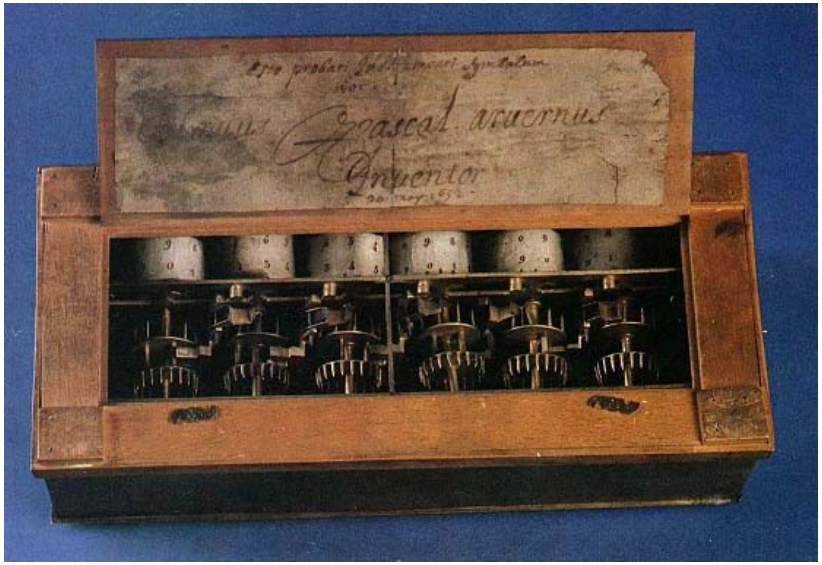
- Máquina de calcular de Pascal (1642)
  - permitia apenas adição e subtração;
  - uso de engrenagens e funcionava manualmente com manivela;
- Leibniz (~1672) → multiplicação e divisão
  - Evolução da máquina de Pascal, com pequenas melhorias e novos recursos, sendo mais rápida;
- Babbage (~1822) → Máquina de Diferenças
  - cálculo de tabelas de números úteis à cálculos navais;
  - executava apenas um algoritmo e permitia só adição e subtração;
  - método de saída: perfuração dos resultados em uma placa de cobre com um buril de aço;

Exemplo de uma máquina de cálculo

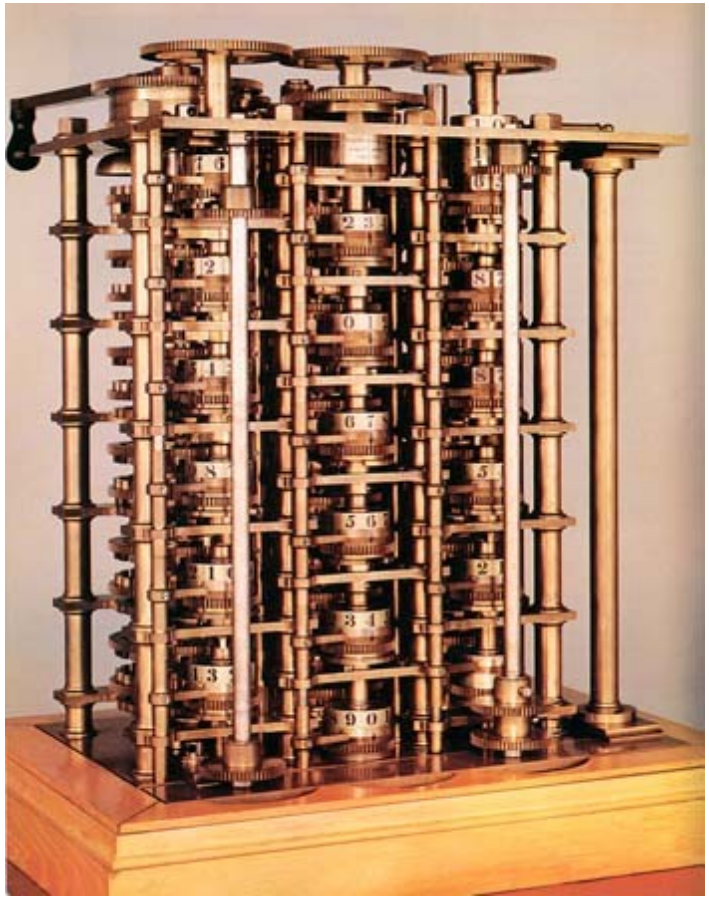


Máquina de Cálculo de Pascal





Máquina de Diferenças, de Charles Babbage (1822)



Geração 0 - Mecânicos

- Babbage (~1834) → Máquina Analítica
  - avanço: máquina de uso geral;
  - ainda era inteiramente mecânica (rodas dentadas e engrenagens);
  - 4 componentes:
    - armazenamento (memória);
    - engenho (unidade de cálculo);
    - seção de entrada (leitora de cartões perfurados);
    - seção de saída (saída perfurada e impressa).
  - tecnologia de '*hardware*' da época ainda era muito imprecisa.

Geração 0 - Mecânicos

- Projeto de calculadoras automáticas
  - uso de relés eletromagnéticos.
- Aiken (1944, Harvard)
  - Mark I → computador de uso geral construído com relés.
    - 72 palavras de 23 dígitos decimais;
    - Baseado nos trabalhos de Babbage;
  - Mark II → Não saiu do projeto, pois a tecnologia de relés tornou-se obsoleta em pouco tempo, finalizando a geração dos dispositivos mecânicos, dando lugar ao **início da era eletrônica**.
  - Partes mecânicas → baixa velocidade de processamento e pouca confiabilidade.

1ª Geração - Válvulas

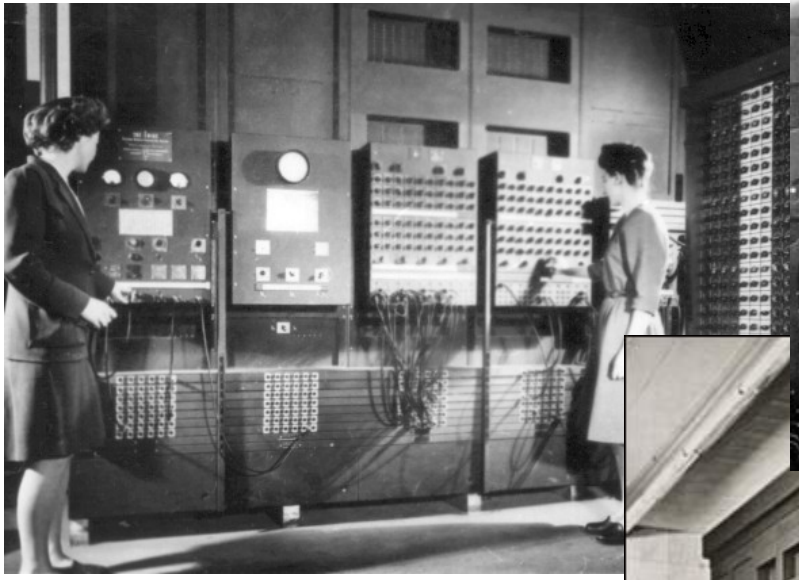
- ENIAC (*Electronic Numerical Integrator And Computer*)
  - Primeiro computador digital de propósito geral (1946)
  - Criado inicialmente para a realização de cálculos balísticos
  - 18 mil válvulas, 10 mil capacitores, 70 mil resistores, um peso de 30 toneladas, consumo de 140 quilowatts e 800 km de cabos
  - Programação feita em painéis (aproximadamente 6000 chaves multiposicionais), com redistribuição de cabos → conhecimento profundo do hardware
  - Máquina decimal, com 20 registradores capazes de armazenar um valor numérico de 10 dígitos
  - Rápida → 5.000 operações/segundo

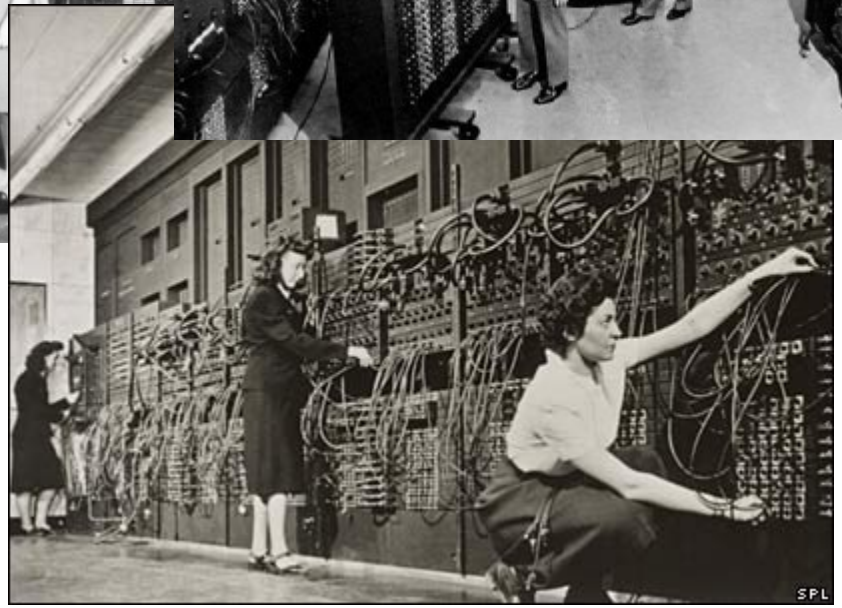
Válvula

- Os primeiros sistemas computacionais tinham como componente principal as **válvulas**

- A função de uma válvula eletrônica abrir ou fechar um circuito, dependendo da intensidade da corrente que passa através dela em determinado momento

ENIAC



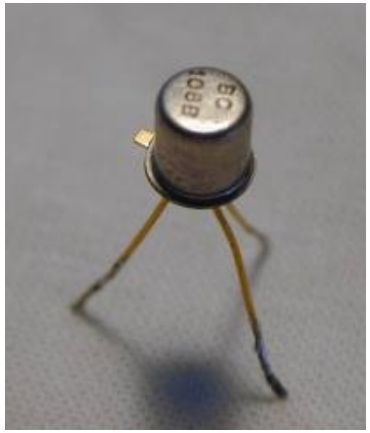


1ª Geração - Válvulas

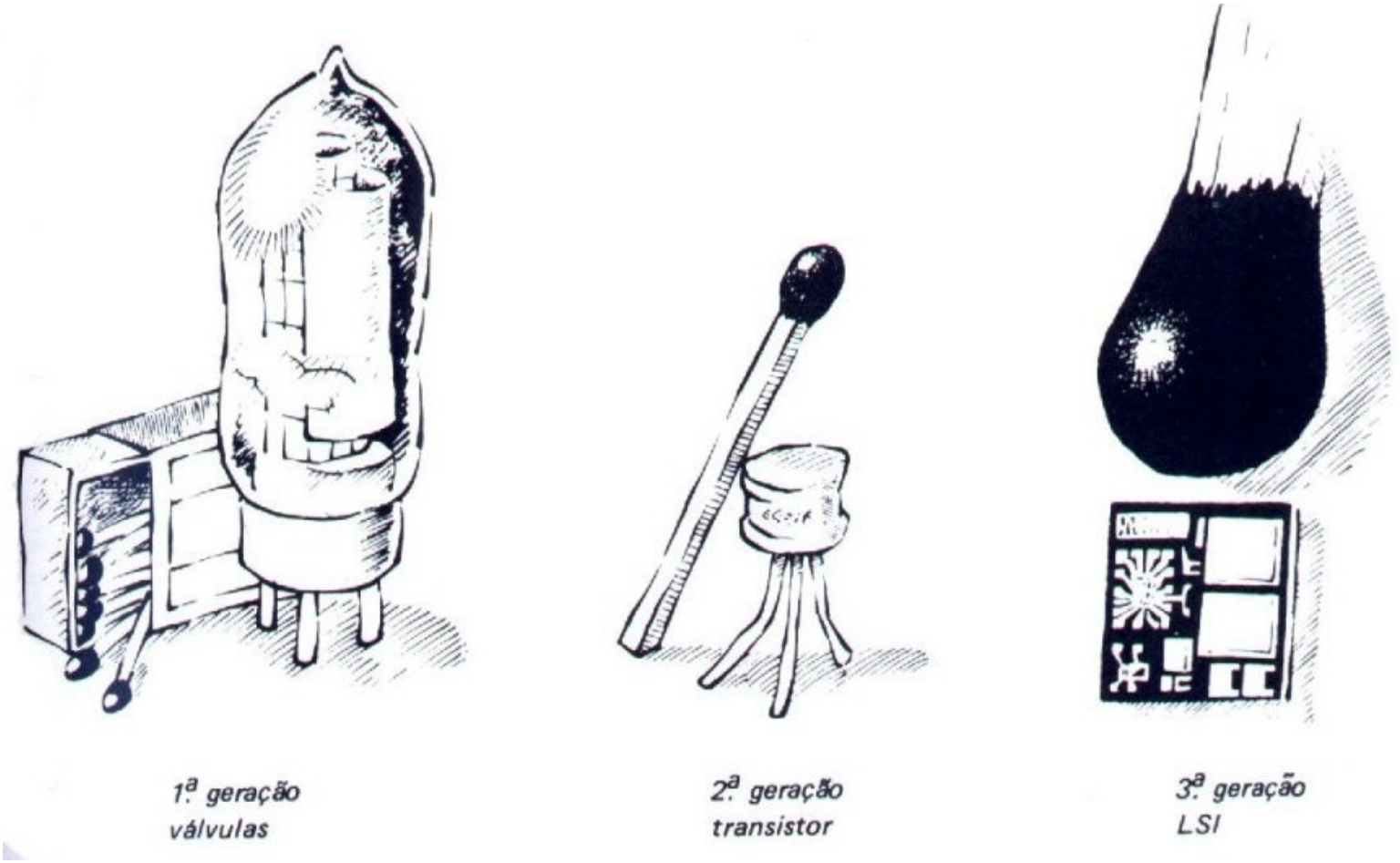
- EDVAC (*Electronic Discrete Variable Automatic Computer*)
- Máquina IAS (*Institute of Advanced Studies,*1946)
  - desenvolvida por John von Neumann;
  - aritmética binária ao invés da decimal;
  - definição de uma arquitetura de computadores com programa armazenado;
    - **Máquina de Von Neumann** → ainda hoje é base de quase todos os computadores digitais;
- UNIVAC I (1949, Mauchly e Eckert)
  - primeiro computador para fins comerciais;
- IBM-701 (1953), 704 (1956) e 709 (1958)

2ª Geração - Transistores

- Transistor
  - melhor custo, tamanho e desempenho do que as válvulas;
  - base da lógica digital → ligar e desligar a corrente elétrica em um dispositivo (2 estados).
- TX-0 (Lincoln Laboratory do M.I.T., 1957)
  - primeiro computador transistorizado, apenas experimental.
- PDP-1 (DEC - Digital Equipment Corporation)
  - marco inicial da indústria de minicomputadores.
- IBM 7090 e 7094 - série transistorizada



Comparação entre válvulas e transistores



1ª geração
válvulas

2ª geração
transistor

3ª geração
LSI

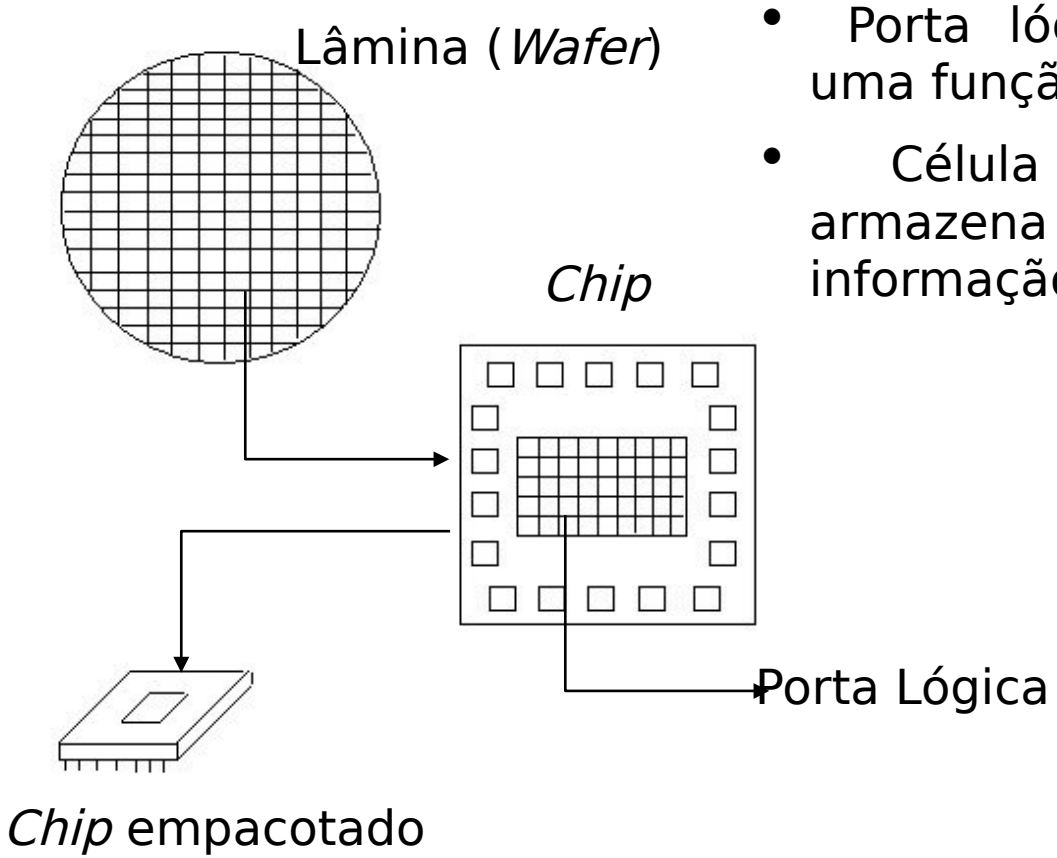
3ª Geração - Circuitos Integrados

- LSI (*Large Scale Integration*)
  - dezenas de transistores colocados em uma única pastilha;
  - surgimento da **microeletrônica**;
  - computadores menores, mais rápidos e mais baratos.

- Série 360 da IBM (1964)
  - "família" de máquinas com mesma linguagem de montagem, mas com tamanhos e potências diferentes;
  - surgimento da técnica de MULTIPROGRAMAÇÃO;
  - sistema operacional OS/360 para gerenciar os recursos do hardware.

- PDP-8 da DEC (1965)
  - Minicomputador de baixo custo (U$ 16.000!);
  - A partir do modelo PDP-8/E foi introduzida a estrutura de barramento.

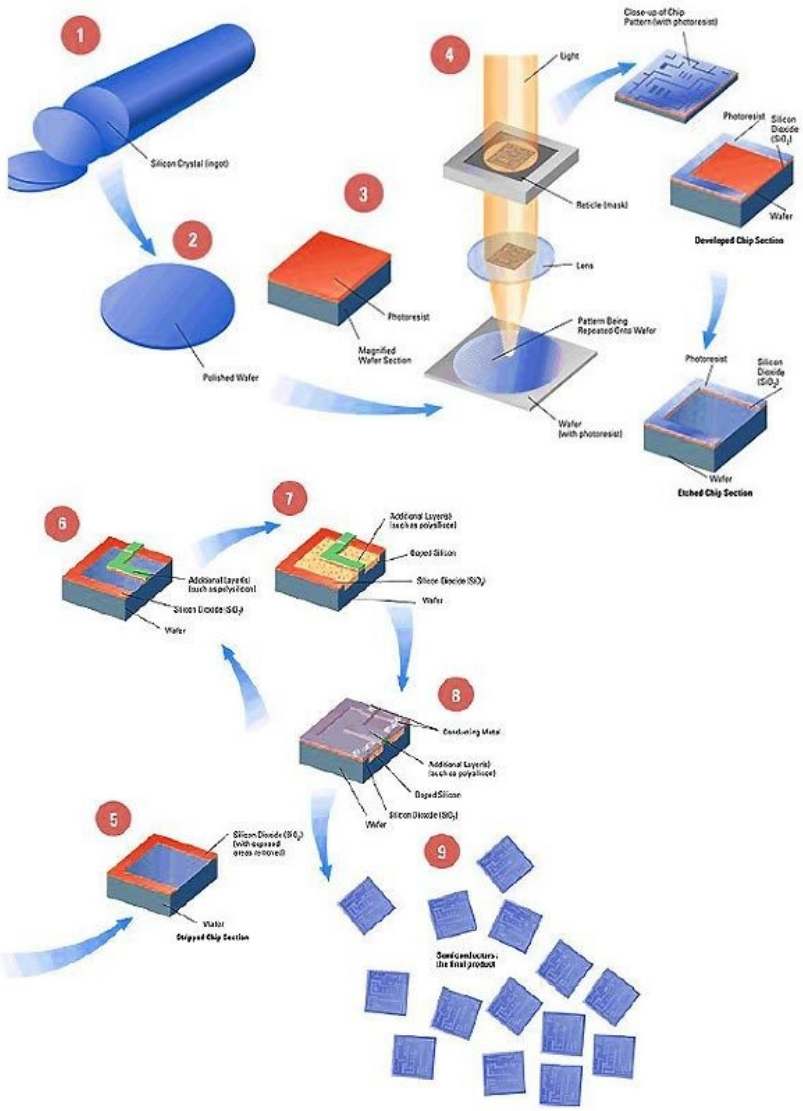
3ª Geração - Circuitos Integrados



- Porta lógica: implementa uma função booleana.
- Célula de memória: armazena um bit de informação.

Produção de *Chips*

- O silício é cortado em lâminas, posteriormente cortadas em *chips*
- Uma máscara fotossensível é usada como molde da impressão dos circuitos
- Uma considerável parte dos chips produzidos a partir de um *wafer* é descartada, por problemas de contaminação, funcionamento e encapsulamento



4ª Geração - VLSIs

- VLSI (*Very Large Scale Integration*)
  - milhões de transistores armazenados em uma única pastilha;
  - computadores cada vez menores e mais rápidos;
  - A miniaturização de componentes eletrônicos levou ao surgimento dos computadores pessoais, criando um novo segmento no mercado;
    - Intel 4004 (1971): o primeiro microprocessador;
    - Intel 8080 (1974): microprocessador de 8 bits de propósito geral.
- Surgimento dos computadores pessoais
  - Série Intel de “chips” - tornou-se padrão;
  - Exs: 8086, 8088, 80286, 80386, 80486, Pentium.
  - IBM PC adotou o chip Intel para UCP.

5ª Geração - ULSIs

- ULSI (*Ultra Large Scale Integration*)
  - Componentes cada vez menores de de mais baixo custo, o que permitiu a evolução de aplicações computacionais mais complexas.
- Evolução das aplicações:
  - Sistemas especialistas, sistemas multimídia, banco de dados distribuídos, inteligência artificial, redes neurais, etc.
  - Necessidade de maior capacidade de processamento e armazenamento de dados.

5ª Geração - ULSIs

- Novos paradigma no projeto de computadores:
  - Arquiteturas Paralelas;
  - Processamento Distribuído nos Sistemas Operacionais;
  - Redes de Alta Velocidade;
  - Linguagens e metodologias de programação concorrentes;
  - Linguagens naturais: interface homem/máquina.

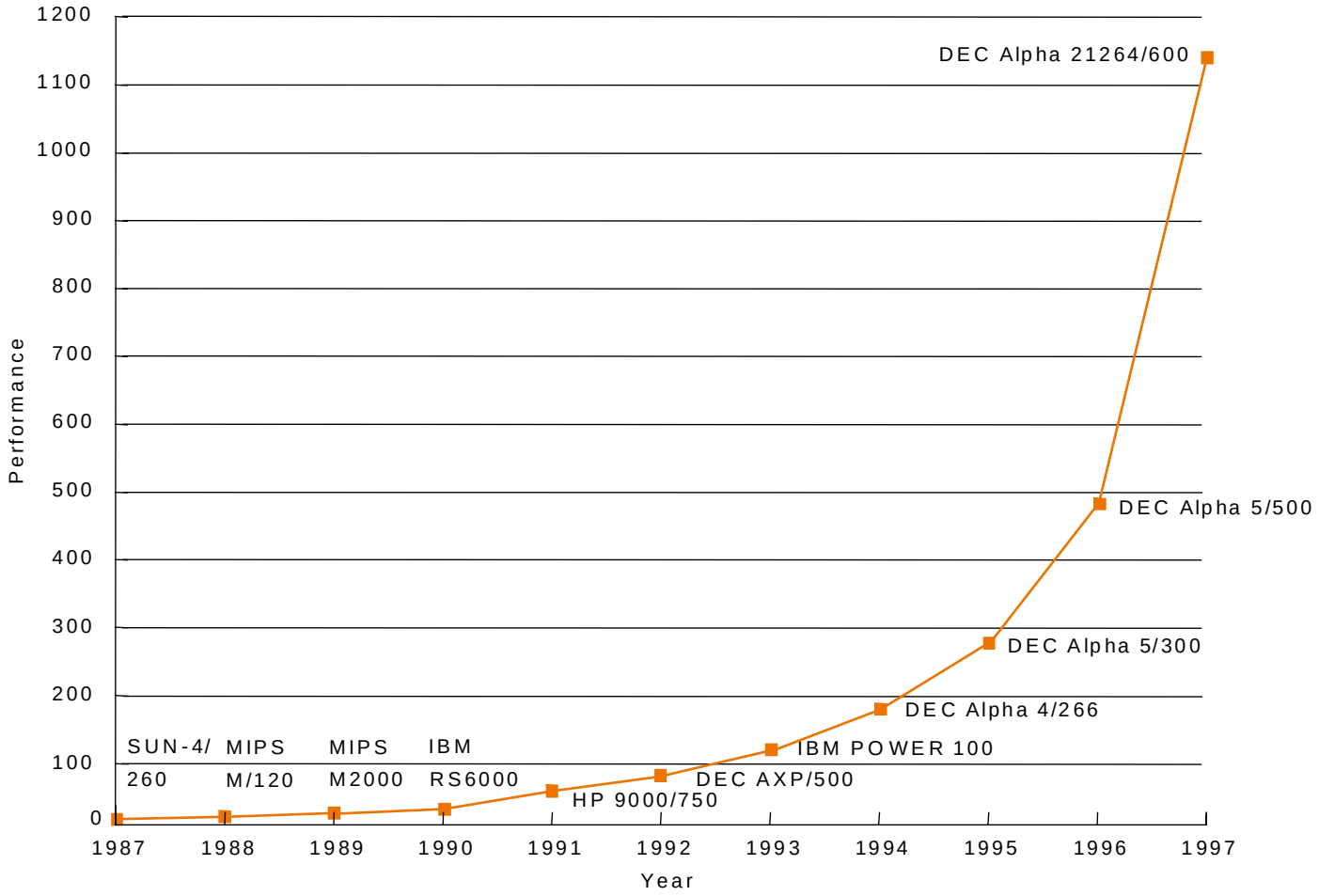
- Novas aplicações:
  - Caixas eletrônicos;
  - Computadores em automóveis;
  - Notebooks;
  - Projeto genoma;
  - *World Wide Web;*
  - Computação distribuída em escala mundial.

Novas tecnologias

- Para o desenvolvimento de aplicações mais eficientes, programadores devem se familiarizar com novos aspectos da organização de computadores.

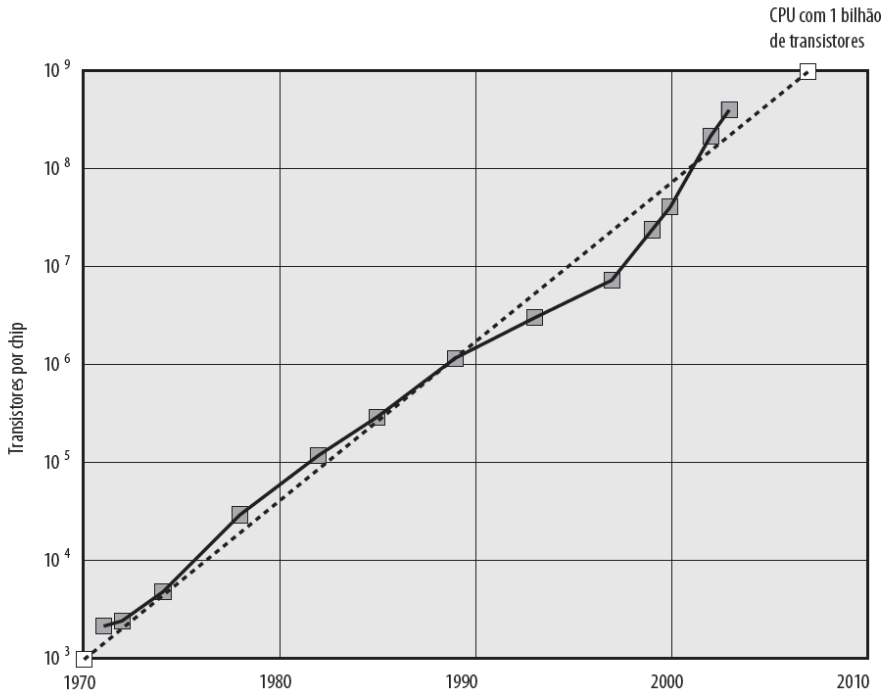
  - Hierarquia de memória (memória principal, memória cache em vários níveis, etc.);
  - Paralelismo de execução de instruções;
  - Novas tecnologias de processamento (processadores *multicore*, etc.).

Histórico do Desempenho das Estações de Trabalho



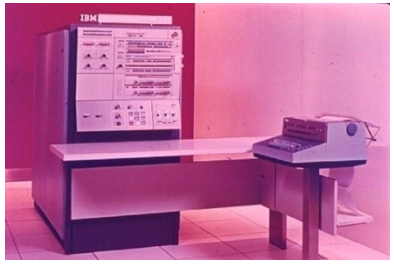
Aumento no número de transistores das UCPs

Figura 2.8 Crescimento na contagem de transistores da CPU (BOHR, 2003[9])



Lei de Moore (1965): “O número de transistores que pode ser colocado em único chip dobra a cada ano” - o ritmo diminuiu para dobrar a cada 18 meses a partir dos anos 1970

Histórico do Desempenho das Estações de Trabalho



1965 IBM 360/50	1977 DEC VAX 11/780	1999 IBM PC 300
0.15 MIPS	1 MIPS	750 MIPS
64 KB	1 MB	64 MB
$1M	$200 K	$2060
$ 6.6M por MIPS	$ 200K por MIPS	$2.75 por MIPS

Evolução dos processadores Intel

Tabela 2.6 Evolução dos microprocessadores Intel

(a) Processadores da década de 1970

	4004	8008	8080	8086	8088
Introduzido	1971	1972	1974	1978	1979
Velocidades de clock	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Largura do barramento	4 bits	8 bits	8 bits	16 bits	8 bits
Número de transistores	2 300	3 500	6 000	29 000	29 000
Dimensão mínima da tecnologia de fabricação (μm)	10		6	3	6
Memória endereçável	640 bytes	16 KB	64 KB	1 MB	1 MB

(b) Processadores da década de 1980

	80286	386TM DX	386TM SX	486TM DX CPU
Introduzido	1982	1985	1988	1989
Velocidades de clock	6–12,5 MHz	16–33 MHz	16 –33 MHz	25–50 MHz
Largura do barramento	16 bits	32 bits	16 bits	32 bits
Número de transistores	134.000	275.000	275.000	1,2 milhão
Dimensão mínima da tecnologia de fabricação (μm)	1,5	1	1	0,8–1
Memória endereçável	16 MB	4 GB	16 MB	4 GB
Memória virtual	1 GB	64 TB	64 TB	64 TB
Cache	–	–	–	8 kB

Evolução dos processadores Intel

(c) Processadores da década de 1990

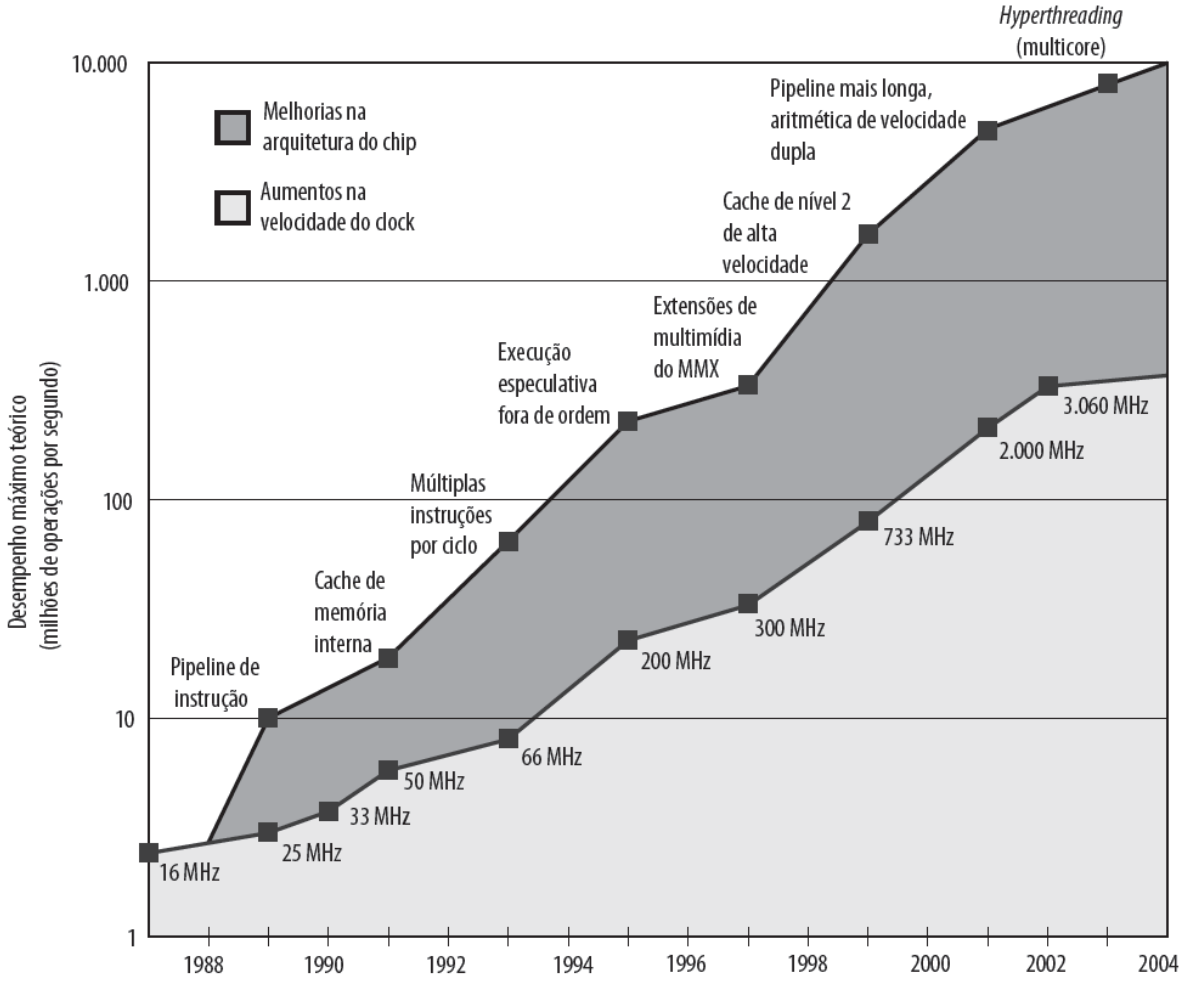
	486TM SX	Pentium	Pentium Pro	Pentium II
Introduzido	1991	1993	1995	1997
Velocidades de clock	16–33MHz	60–166 MHz	150–200 MHz	200–300 MHz
Largura do barramento	32 bits	32 bits	64 bits	64 bits
Número de transistores	1,185 milhão	3,1 milhões	5,5 milhões	7,5 milhões
Dimensão mínima da tecnologia de fabricação (μm)	1	0,8	0,6	0,35
Memória endereçável	4 GB	4 GB	64 GB	64 GB
Memória virtual	64 TB	64 TB	64 TB	64 TB
Cache	8kB	8kB	512 kB L1 e 1 MB L2	512 kB L2

(d) Processadores recentes

	Pentium III	Pentium 4	Core 2 Duo	Core 2 Quad
Introduzido	1999	2000	2006	2008
Velocidades de clock	450–660 MHz	1,3–1,8 GHz	1,06–1,2 GHz	3 GHz
Largura do barramento	64 bits	64 bits	64 bits	64 bits
Número de transistores	9,5 milhões	42 milhões	167 milhões	820 milhões
Dimensão mínima da tecnologia de fabricação (nm)	250	180	65	45
Memória endereçável	64 GB	64 GB	64 GB	64 GB
Memória virtual	64 TB	64 TB	64 TB	64 TB
Cache	512 KB L2	256 KB L2	2 MB L2	6 MB L2

Desempenho

Figura 2.12 Desempenho do microprocessador Intel (Gibbs, 2004[n])



Sistemas de numeração - Introdução

- Os computadores eletrônicos tem como base para seu funcionamento a utilização de corrente elétrica. Diferente de outras máquinas que a presença ou ausência de corrente apenas significam se estão ligadas ou desligadas, um computador deve utilizá-la para manipular e armazenar informações.

- A partir dos dois estados representados pela presença ou ausência de corrente, todo o trabalho realizado pelo computador será baseado no sistema binário, onde trabalha-se somente com os símbolos ‘0’ e ‘1’ para a representação de informações.

Sistemas de Numeração

- Os sistemas de numeração surgiram com a evolução da civilização para atender a necessidade de registrar informações sobre quantidades.

- Sistema decimal: o sistema de numeração mais comum de ser empregado, o que estamos mais habituados a trabalhar e pensar (mas existem exceções! Ex.: dúzia, grosa).
  - Surgimento a partir da analogia com a contagem utilizando os dedos da mão;
  - Criação da notação posicional (números em posições diferentes representam valores diferentes - unidade, dezena, centena, etc.).

Sistemas de Numeração

- Base de um sistema de numeração: quantidade de algarismos disponíveis para a representação.
  - Ex: base decimal: 0 1 2 3 4 5 6 7 8 9 → base 10
- A posição ocupada por um algarismo em um número altera seu valor de uma potência de 10 para cada casa à esquerda.
  - Ex: $125 = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$

Níveis de Programação

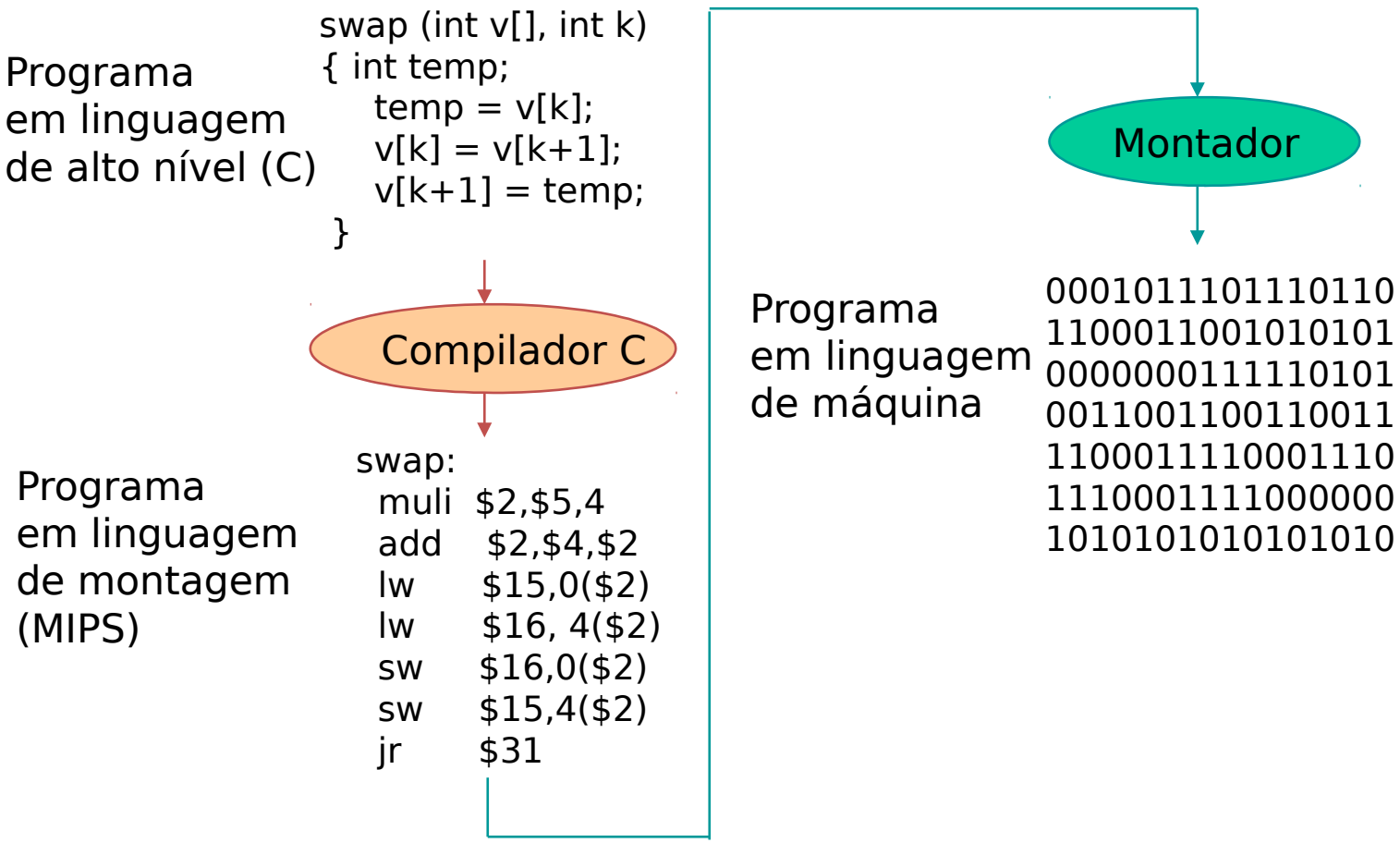
- Linguagem de montagem ainda está longe da notação utilizada pelas pessoas que querem gerar programas para resolver problemas.
- Solução: criar um conjunto de instruções (uma linguagem de mais alto nível) que seja mais dirigido às pessoas do que à máquina.
- Linguagem de programação de alto nível traduzida pelo compilador.
  - A+B → add A,B → 1000110010100000

Representação Simbólica

- No programa em linguagem de montagem, as instruções são representadas através de “abreviações”, chamadas de mnemônicos, que associam o nome da instrução à sua função, como por exemplo:

ADD	Soma
SUB	Subtração
MUL	Multiplicação
DIV	Divisão
LOAD	Carregar dados da memória
STOR	Armazenar dados na memória

Exemplo de instruções em diferentes níveis



Representação da Informação

- Sistema Binário: bit (*binary digit*): 1 ou 0 → sistema de numeração conhecido como base 2.
- Sinal elétrico mais simples tem dois estados, representados por sim e não.
- Não confundir bases: $10_2 = 2_{10}$; dez igual a dois???
- Instruções: conjunto de bits inteligível pelo computador.
  - Ex: 1000110010100000 (deve somar dois números).

Representação da Informação

Repr. Binária	Potência	Repr. Decimal
1	2^0	1
10	2^1	2
100	2^2	4
1000	2^3	8
10000	2^4	16
100000	2^5	32
1000000	2^6	64
10000000	2^7	128
100000000	2^8	256
1000000000	2^9	512
10000000000	2^{10}	1024

Representação de Informação

- Representação binária: perfeitamente adequada para computadores, mas para seres humanos...

- Solução: trabalhar com bases que utilizem a potência de 2; as mais utilizadas são octal (2^3) e hexadecimal (2^4).

  - Octal: oito algarismos para representação; um algarismo octal representa 3 bits.
    - 0 1 2 3 4 5 6 7

  - Hexadecimal: dezesseis algarismos para representação; um algarismo hexadecimal representa 4 bits.
    - 0 1 2 3 4 5 6 7 8 9 A B C D E F

Representação de Informação

- A princípio, as bases binária, octal e hexadecimal podem parecer um conceito totalmente novo, mas não são. Sua formação e comportamento funcionam exatamente como na base decimal; a lógica de construção é exatamente a mesma!

- Lembre-se da base decimal, como temos um conjunto de símbolos ou algarismos, que associamos a certas quantidades de alguma coisa que queremos contar. Quando não existem mais algarismos a serem utilizados sozinhos, começamos a combiná-los, seguindo um critério estabelecido pela notação posicional

Tabela de Representação de Sistemas de Numeração

Decimal	Binário	Hexadecimal	Octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Conversões entre Bases

- Conversão binário-octal: como 1 algarismo octal representa 3 algarismos binários, separar os bits de um número binário em grupos de 3 bits e converter cada um destes grupos para o algarismo octal equivalente.

Ex.: 001111110101_2 =

001	111	110	101
1	7	6	5

= 1765_8

Conversões entre Bases

- Conversão binário-hexadecimal: análoga a utilizada na conversão binário-octal, a única diferença é que os grupos de algarismos do binário serão formados por 4 bits.

Ex.: 001111110101_2 =

0011	1111	0101
3	F	5

= $3F5_{16}$

Conversões entre Bases

- Para as conversões octal-binário e hexadecimal-binário basta aplicar a operação inversa, ou seja, para cada algarismo, obter a representação em binário, respeitando a ordem posicional.

- MUITO IMPORTANTE!!! Não esquecer de incluir zeros não-significativos!

Ex: $3F5_{16}$ → $3_{16} = 11_2$, $F_{16} = 1111_2$, $5_{16} = 101_2$ **$= 001111110101_2$**

e não $111111101_2 = 1FD_{16}$

Exercícios

- Efetue as seguintes conversões de base:
  - $10100101101_2 = (\ \ )_{16} = (\ \ )_8$
  - $5FB7_{16} = (\ \ )_2 = (\ \ )_8$
  - $74325_8 = (\ \ )_2 = (\ \ )_{16}$
- Obs: para as conversões octal/hexadecimal e hexadecimal/octal, não há fórmula direta, estes devem ser convertidos primeiro para binário e depois convertidos novamente.

Conversões entre Bases

- Conversão de uma base qualquer para decimal: seja um número em uma base *b* não-decimal composto pelos algarismos a_n, a_{n-1}, ... a_1, a_0, obteremos seu equivalente decimal da forma:
  - $N_b = a_n \times b^n + a_{n-1} \times b^{n-1} + ... + a_1 \times b^1 + a_0 \times b^0$

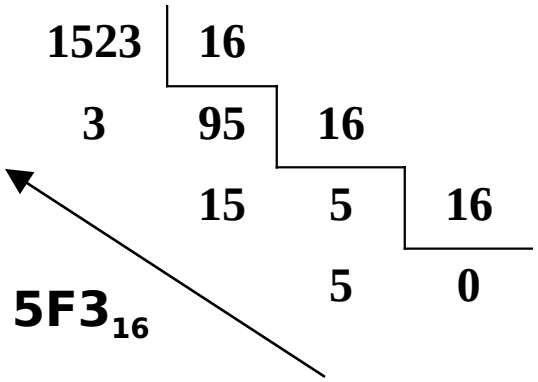
Exemplos:

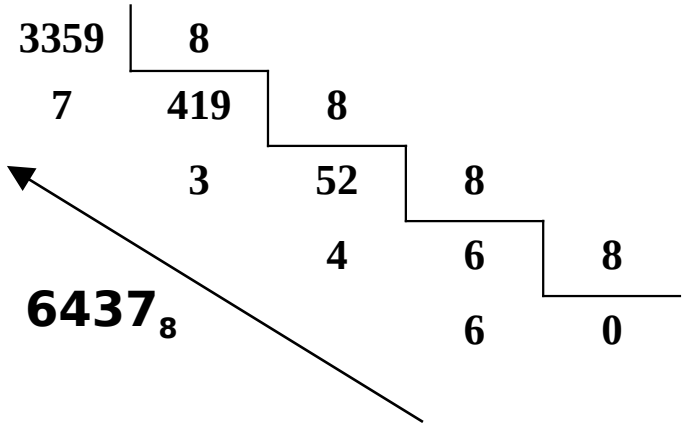
- $5F3_{16} = 5 \times 16^2 + 15 \times 16^1 + 3 \times 16^0 = 1280 + 240 + 3 = 1523_{10}$
- $6437_8 = 6 \times 8^3 + 4 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 = 3072 + 256 + 24 + 7 = 3359_{10}$

Conversões entre Bases

- Conversão da base decimal para uma base qualquer: o número decimal é dividido sucessivas vezes pela base que se deseja a conversão, até que não possa mais ser dividido. O resto de cada divisão será o número na base desejada, indo do último para o primeiro resto obtido.

Exs.:





Exercícios

- Efetue as seguintes conversões de base:
  - $1D5_{16} = (\ \ )_{10}$
  - $100101101_{2} = (\ \ )_{10}$
  - $5341_{8} = (\ \ )_{10}$
  - $155_{10} = (\ \ )_{2}$
  - $63_{10} = (\ \ )_{8}$
  - $119_{10} = (\ \ )_{16}$

Álgebra de Boole

- No século XIX, o matemático inglês George Boole desenvolveu um estudo estabelecendo o conjunto de regras e estruturas a serem aplicadas a símbolos lógicos, ficando conhecida como Álgebra de Boole (ou Álgebra Booleana)

- A motivação por trás deste estudo surgiu dos problemas constantes nos projetos de circuitos de chaveamento com relés; o planejamento lógico reduziria a complexidade e os erros na implementação

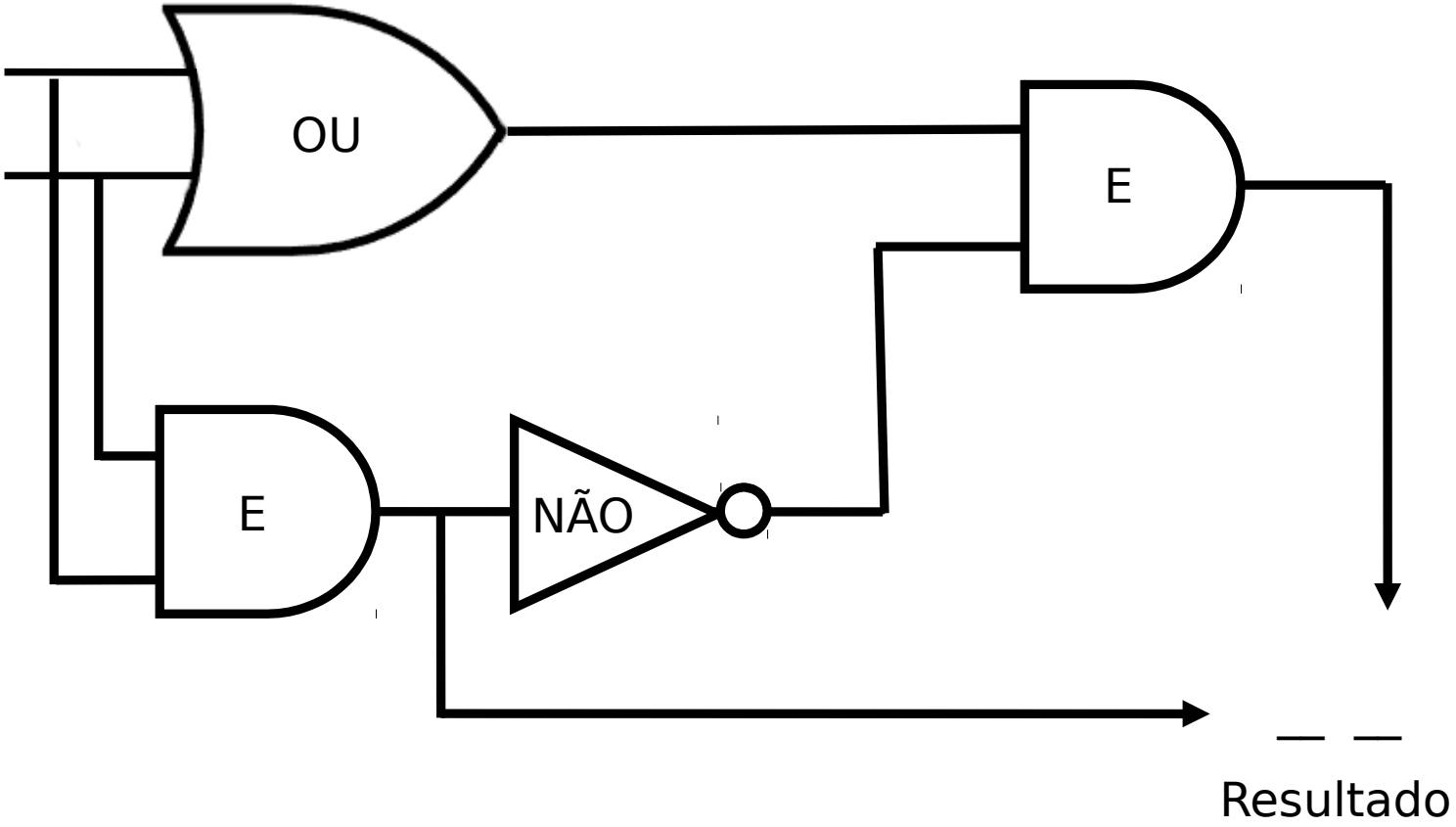
Álgebra de Boole

- Os circuitos eletrônicos modernos mantém uma estrutura de funcionamento análoga a utilizada com relés, portanto, as mesmas técnicas desenvolvidas para circuitos com relés ainda são utilizadas no projeto de modernos computadores de alta velocidade;

- A álgebra booleana, por meio de suas regras, proporciona um modo econômico e direto de descrição do conjunto de circuitos usado nos computadores;

Exemplo de Lógica em Circuitos

- Circuito somador parcial:



Álgebra de Boole

- As variáveis usadas nas equações booleanas podem assumir apenas um de dois valores possíveis, representados pelos símbolos ‘0’ e ’1’, chamados de estados lógicos

- Usualmente utilizamos ‘1’ para representar um circuito fechado ou no estado ligado ‘ON’, e ‘0’ um circuito aberto ou no estado desligado ‘OFF’

Operações no Sistema Binário

- Adição e multiplicação binária: o conjunto de operações básicas de adição e multiplicação no sistema binário pode ser representado de forma resumida pelas seguintes tabelas:

Adição

+	**0**	**1**
0	0	1
1	1	$^{1}0$

Leia-se ‘0’ e ‘vai 1’ para o dígito de ordem superior

Multiplicação

x	**0**	**1**
0	0	0
1	0	1

Operações no Sistema Binário

- Obs. 1: As operações de adição e multiplicação são realizadas operando-se as colunas da direita para a esquerda, da mesma forma que nas operações decimais.

- Obs. 2: Todas as operações aritméticas podem ser realizadas através da soma:
  - a multiplicação pode ser feita através de sucessivas somas (um número N vezes *'b' é igual a soma de N com N 'b'* vezes);
  - a subtração pode ser feita através do método de complemento a base (que veremos a seguir);
  - Finalmente, a divisão pode ser feita através de sucessivas subtrações.

Exemplos de operações binárias

$3_{10} + 5_{10} = 11_2 + 101_2 \longrightarrow$

$$\begin{array}{r} 101 \\ +\ \ 11 \\ \hline 1000 \end{array} \longrightarrow 1000_2 = 8_{10}$$

$3_{10} \times 5_{10} = 11_2 \times 101_2 \longrightarrow$

$$\begin{array}{r} 101 \\ \times\ \ 11 \\ \hline 101 \\ +\ 101 \\ \hline 1111 \end{array} \longrightarrow 1111_2 = 15_{10}$$

Exemplos de operações binárias

$37_{10} + 30_{10} = 100101 + 11110 \longrightarrow$

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1\,\,} \\ 100101 \\ +\;\underline{11110} \\ 1000011 \end{array} \longrightarrow 2^6+2^1+2^0 = 67_{10}$$

$7_{10} \times 7_{10} = 111 \times 111 \longrightarrow$

$$\begin{array}{r} 111 \\ \times\;\underline{111} \\ 111 \\ 111 \\ +\;\underline{111} \\ 110001 \end{array} \longrightarrow 2^5+2^4+2^0 = 49_{10}$$

Exercícios

- Converta para binário e efetue as seguintes operações:

  - $63_{10} + 34_{10}$
  - $32_{10} \times 6_{10}$
  - $7BA_{16} + 9C6_{16}$
  - $D2_{16} \times 5_{16}$
  - $73_{8} + 34_{8}$
  - $34_{8} \times 21_{8}$

Operações Binárias

- Subtração: o método mais simples de subtração entre dois valores binários é através do complemento a base, executado pela seguinte sequência de instruções (ei, é um algoritmo!):

  - Mantenha o minuendo na sua forma original;
  - Inverta o subtraendo (todo ‘1’ vira ‘0’ e todo ‘0’ vira ‘1’;
  - Some o minuendo e o subtraendo;
  - Some 1;
  - Ignore o algarismo mais significativo caso ele esteja numa posição notacional que os operandos não tenham um algarismo significativo.

- Obs.: não se esqueça de representar os zeros não-significativos (pois este serão importantes na inversão)!

Complemento a 2

- Esta forma de representação que vimos é chamada **complemento a 2**; pode ser usada para outras operações particulares, além da transformação de uma subtração em adição.

- Em qualquer situação, a conversão é feita da mesma forma: invertem-se os bits (0 → 1 e 1 → 0) e soma ‘1’.

- É importante sempre lembrar dos zeros não-significativos para realizar uma conversão, pois zeros à esquerda se tornarão ‘1’s.

Exemplo de subtração binária

$37_{10} - 12_{10} = 100101 - 001100$

A quantidade de casas foi igualada com zeros à esquerda!

O minuendo foi mantido!

```
    1   111
  100101
+ 110011
 1011000
+      1
 1011001
```

O subtraendo foi invertido!
12 = 001100 → 110011

Soma 1

Este algarismo é desprezado!

$= 011001 = 2^4+2^3+2^0 = 25_{10}$

Exercícios

- Efetue as seguintes operações:
  - $37_{10} - 30_{10}$
  - $83_{10} - 82_{10}$
  - $63_{8} - 34_{8}$
  - $77_{8} - 11_{8}$
  - $BB_{16} - AA_{16}$
  - $C43_{16} - 195_{16}$

Divisão de números binários

- Como nas demais operações aritméticas, a divisão binária é similar à divisão decimal, sendo:
  - 0 / 1 = 0
  - 1 / 1 = 1
  - Divisão por zero → erro
- Divisões binárias podem ser realizadas pelo método tradicional (dividendo / divisor = quociente e um resto) ou através de sucessivas subtrações

Passos para a divisão binária

- A partir da esquerda, avançam-se tantos algarismos quantos sejam necessários para obter-se um valor maior ou igual ao divisor;
- Encontrado este valor, registra-se 1 para o quociente;
- Subtrai-se do valor obtido no dividendo o valor do divisor;
- Ao resultado da subtração, acrescentam-se mais algarismos do dividendo (se ainda houver algum que não fora utilizado) até obter-se um valor maior ou igual ao divisor.
- Se não houverem mais algarismos a serem utilizados e o valor do dividendo for menor do que o divisor, encerra-se a operação e temos um resto não-nulo;

Passos para a divisão binária

- Se os algarismos ainda não utilizados do dividendo tiverem valor igual a zero, a cada um deles utilizado deve se acrescentar um zero ao quociente

- Repita o processo até que não existam mais algarismos do dividendo.

- Ex.: 100_2 / 10_2

10’0	**10**
-10	**10**
00	

Exemplo de divisão binária

- $18_{10} / 3_{10} \rightarrow 10010_2 / 11_2$

100’1’0	**11**
- 11	**110**
011	
00	

Exercícios

- Efetue as seguintes operações:
  - $120_{10} / 5_{10}$
  - $63_{10} / 8_{10}$
  - $145_{8} / 4_{8}$
  - $B2_{16} / 8_{16}$

Deslocamento de bits

- Operação *shift*: faz o deslocamento de bits dentro de um número, equivalendo às operações de multiplicação ou divisão por potência de 2
  - Multiplicação: deslocamento à esquerda;
  - Divisão: deslocamento à direita;
- Exemplos:
  - 2 x 2 = 4 (10_2 x 10_2 = 100_2) → bit 1 se deslocou 1 casa para à esquerda
  - 4 (2^2) x 2= 8 (100_2 x 100_2 = 1000_2) → bit 1 se deslocou 2 casas para à esquerda
  - 16 (2^4) / 4 (2^2) = 4 (10000_2 / 10_2 = 100_2) → bit 1 se deslocou 2 casas para à direita

Números binários fracionários

- É possível trabalhar com números binários com representação fracionária, que da mesma maneira que os valores decimais, utiliza uma vírgula para separar a parte inteira da parte fracionária
  - Ex: $101,11_2$
- Como é feita a conversão?
  - A parte inteira é convertida da maneira tradicional, enquanto a parte fracionária utiliza expoentes negativos

Números binários fracionários

- Para o exemplo apresentado, temos:

  - $101{,}11_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 4 + 0 + 1 + 0{,}5 + 0{,}25 = 5{,}75_{10}$

- Recordando potências negativas:

  - $2^{-1} = (1/\ 2^1) = 0{,}5$
  - $2^{-2} = (1/\ 2^2) = 0{,}25$
  - $2^{-3} = (1/\ 2^3) = 0{,}125$
  - $2^{-4} = (1/\ 2^4) = 0{,}0625$
  - e assim por diante

Números binários fracionários

- Como é a conversão de um decimal fracionário para binário?
  - Para a parte inteira, segue a regra de conversão já estudada;
  - Para a parte fracionária, inicialmente multiplica-se o seu valor pela base que se deseja converter, neste caso, 2;
  - Em seguida, o valor da parte inteira gerada representa o primeiro bit da parte fracionária;
  - Repete-se o processo com a parte fracionária até a precisão desejada ou até que a parte fracionária seja igual a zero.

Números binários fracionários

- Para o exemplo apresentado anteriormente, temos:
  - $5{,}75_{10} = 5\ (101_2) + 0{,}75$
  - $0{,}75 \times 2 = 1{,}5 = 1 + 0{,}5$
  - $0{,}5 \times 2 = 1{,}0 = 1 + 0{,}0$
- Logo, a representação binária é igual a $101{,}11_2$
- Em alguns casos, dificilmente encontramos a parte fracionária igual a zero. Isto é causado por diferenças de precisão entre as bases.

Números binários fracionários

- Quando encontramos uma dízima periódica em binário com uma quantidade ilimitada de bits, o número fica da forma:
  - 90,8 = 1011010,1100110011001100110011001100110
  - 0110011001100110011001100110011001100110011001100...
- Se a variável utilizada for do tipo *float*, são reservados 23 bits para representar a parte fracionária, ou seja, só teremos espaço para armazenar uma parte da resposta, e será feito um arredondamento, acarretando um pequeno erro.

Números binários fracionários

- Como resolver este problema?
  - Toda linguagem de programação dispõe de um tipo de variável que dispõe de mais casas binárias para a representação da parte fracionária. Em ‘C’, é possível declarar o tipo da variável como *double*, que reserva 52 bits para armazenar os bits da fração

Exercícios

Realize as seguintes conversões fracionárias:

- $110{,}0110_2 = ?_{10}$
- $10{,}1110 = ?_{10}$
- $1111{,}010_2 = ?_{10}$
- $7{,}6_{10} = ?_2$
- $13{,}127_{10} = ?_2$
- $5{,}331_{10} = ?_2$

Representação de Dados

- Antes de prosseguirmos, precisamos entender os seguintes conceitos:
  - BIT (*BInary digiT*): é a representação do menor item de dado possível;
  - Byte: um conjunto de bits (é adotado como padrão que 1 byte é formado por 8 bits);
  - Palavra: um conjunto de bytes; um computador com palavra de 32 bits tem 4 bytes por palavra. A maioria das instruções de um computador opera sobre palavras, por exemplo, cada operação de um computador de 32 bits opera sobre palavras de 32 bits, deve ter registradores de 32 bits, instruções para 32 bits, etc.

- A limitação do número de bits é necessária para que possamos representar na forma binária diferentes tipos de dados (números, instruções, etc.)

Representação de Dados

- Da forma que vimos até agora todas as representações numéricas são de valores inteiros e sem um limite máximo, isto é, iríamos de 0 a ∞. Devemos estabelecer um domínio de valores dentro do qual o sistema irá operar. Será necessário estabelecermos uma forma de representar valores negativos

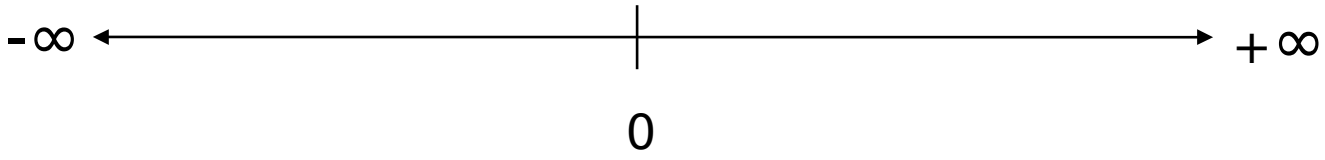
- Na prática utilizamos o bit na posição mais significativa (isto é, o bit mais a esquerda) para a representação do sinal, com a seguinte convenção:
  - Bit 0 → sinal positivo.
  - Bit 1 → sinal negativo.

Representação de Dados

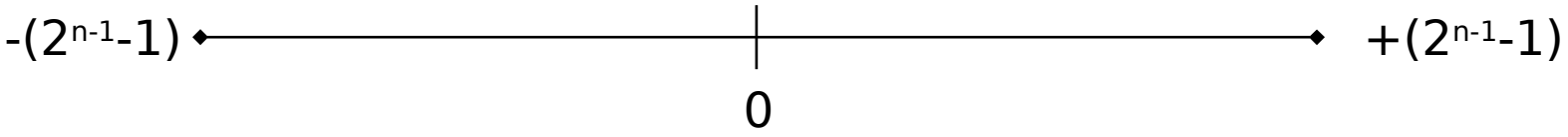
- Desta forma, uma representação em binário com *n* bits teria disponíveis para a representação do número *n-1* bits (o bit mais significativo representa o sinal). Este modelo de representação é chamado de representação em sinal e magnitude

Representação de Dados

- Teoria:



- Prática (limitado a *n* bits):



Representação em Sinal e Magnitude

- A magnitude é o valor absoluto de um número, que independe do sinal, representada em binário;
- O sinal é representado pelo bit mais significativo, sendo positivo se representado por ‘0’ e negativo se representado por ‘1’;
- O valor dos bits usados para representar a magnitude independe do sinal, ou seja, seja o número positivo ou negativo, a representação da magnitude será exatamente a mesma, variando somente o bit de sinal.

Representação em Sinal e Magnitude

Sinal	Magnitude

Ex.: $0110110 = +\ 54_{10}$

$1110110 = -\ 54_{10}$

o binário 110110 corresponde ao valor absoluto 54.

A faixa de representação de um valor binário em sinal e magnitude com *n* bits possui 2^n representações, representando os valores de **$-\ (2^{n-1}-1)$** (menor valor negativo) a **$+\ (2^{n-1}-1)$** (maior valor positivo).

Como é a representação do zero em sinal e magnitude?

Exemplo de Representação de Dados

Valor decimal	Valor binário com 8 bits (7 + bit de sinal)
+9	00001001 (bit inicial 0 significa positivo)
-9	10001001 (bit inicial 1 significa negativo)
+127	01111111 (bit inicial 0 significa positivo)
-127	11111111 (bit inicial 1 significa negativo)

Como podemos observar, o maior e menor valores que podemos representar com 8 bits são, respectivamente, +127 e -127.

Exercício

- Complete o quadro a seguir com os valores correspondentes para uma arquitetura de 32 bits, representados como uma potência de base 2, quando conveniente:

32 bits	Sem Sinal	Sinal e Magnitude
Menor Valor		
Maior Valor		
Quantidade de valores distintos		

Aritmética em Sinal e Magnitude

Soma em sinal e magnitude:

- Verificar o sinal das parcelas a serem somadas;
  - Se forem iguais, repetir o sinal e somar as magnitudes;
  - Se forem diferentes:
    - verificar qual parcela tem a maior magnitude;
    - repetir o sinal da maior magnitude;
    - subtrair a menor magnitude da maior magnitude.

- Os bits referentes ao sinal dos operandos, positivo ou negativo, não devem ser operados aritmeticamente!

Representação de operações

- Lembre-se que as instruções são interpretadas em linguagem de montagem (que será estudada com detalhes mais adiante), utilizando *mnemônicos* para representar as operações, desta forma, apenas os operandos possuem sinais, por exemplo:

  - +5 – (– 7) → SUB (5, -7)
  - +12 + (– 3) → ADD (12, -3)

Aritmética em Sinal e Magnitude

- Subtração em sinal e magnitude:
  - É calculada exatamente como uma soma entre duas parcelas de sinais diferentes
  - É importante lembrar que as operações aritméticas são realizadas somente com as magnitudes, então sempre o menor valor é subtraído do maior valor!
  - Quanto ao sinal, basta fazer uma análise lógica se o resultado será positivo ou negativo, da mesma forma que é feito com operações decimais.

Exemplos de operações em Sinal e Magnitude

$+54_{10} + (+43_{10})$ (em uma representação com palavras de 8 bits):
$= 00110110 + 00101011$

```
  11111
  0110110
+ 0101011
  1100001
```

$\longrightarrow$ $2^6 + 2^5 + 2^0 = +\ 97_{10}$

$+54_{10} - (+43_{10})$ (em uma representação com palavras de 8 bits):

```
  0110110
- 0101011
```

$\longrightarrow$

```
   111 1
  0110110
+ 1010100
 10001010
+        1
 10001011
```

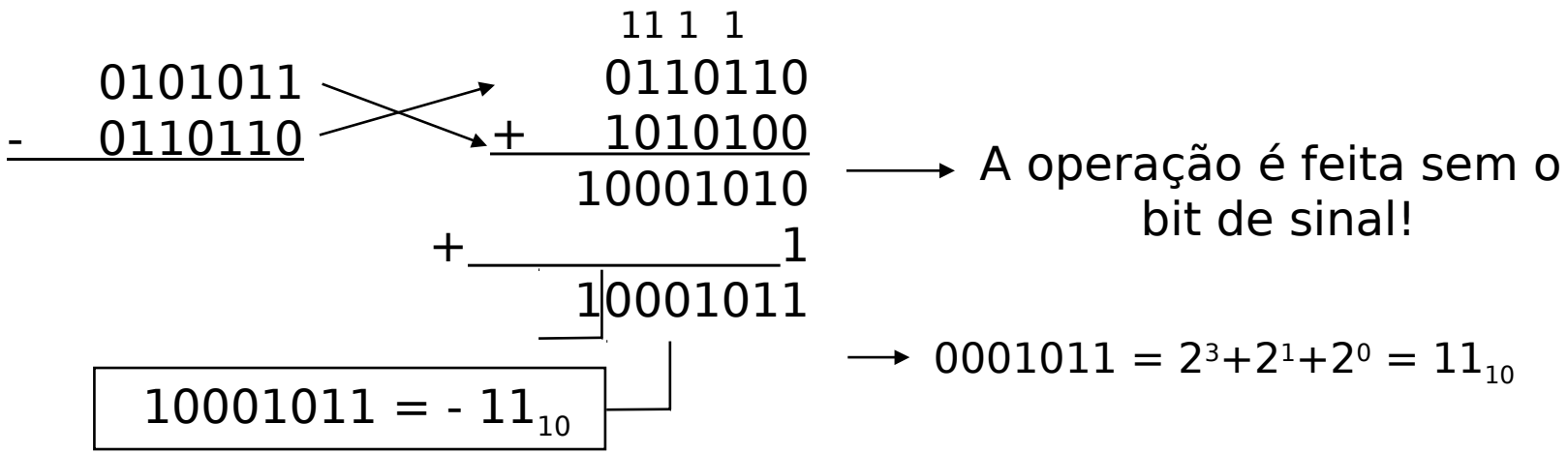
$\longrightarrow$ A operação é feita sem o bit de sinal!

$10001011 = 2^3+2^1+2^0 = 11_{10}$

$00001011 = +\ 11_{10}$

Exemplos de operações em Sinal e Magnitude

$+43_{10} - (54_{10})$ (em uma representação com palavras de 8 bits):



Por que os operandos foram invertidos? Lembre-se que as operações são feitas somente com a magnitude, que são valores positivos, portanto, o menor valor deve ser subtraído do maior valor.

Exemplo de Multiplicação

Multiplicação:

Na multiplicação são utilizadas as regras já conhecidas para a magnitude, e o sinal é manipulado da mesma forma que na aritmética tradicional.

$+19_{10} \times (-19_{10}) = 010011 \times 110011 = -361_{10}$

```
      10011
  x   10011
      10011
     10011
+   10011
    101101001
```

Operamos sem o bit de sinal (o mais significativo)

$2^8+2^6+2^5+2^3+2^0 =$

$256 + 64 + 32 + 8 + 1 = 361_{10}$

Exercícios

- Efetue as seguintes operações considerando a representação em sinal e magnitude, sem restrições de limite de bits:

  - $47_{10} - (+52_{10})$
  - $69_{10} - (+40_{10})$
  - $32_{10} \times (-14_{10})$
  - $54_{8} \times (-5_{8})$
  - $A2_{16} \times (-13_{16})$

Limites de memória

- Em sistemas reais, tanto os valores operados quanto os resultados produzidos podem exceder os limites de armazenamentos impostos pela arquitetura, ou seja, o número de bits que compõem a palavra (principalmente em multiplicações).

- Para contornar esta limitação, no caso de o número de bits da solução exceder o limite da palavra, podem ser utilizadas duas palavras para armazenar o resultado

- Uma palavra de n bits contém os *n*-1 bits do valor, precedidos pelo bit de sinal. A outra palavra conterá os bits mais significativos, ‘0’s complementares, se necessários, e o sinal do resultado.

Exemplo de uso de mais de uma palavra para expressar um valor

$+19_{10} \times (-19_{10}) = 010011 \times 110011$ **(Agora limitado a 6 bits)**

```
     10011
  x  10011
     10011
    10011
+  10011
   101101001
```

Operamos sem o bit de sinal (o mais significativo)

O bit de sinal será negativo

1	0	1	0	1	1		1	0	1	0	0	1

$= 1101101001_2 = -361_{10}$

bits mais significativos **bits menos significativos**

Limites de memória

- No caso de não ser possível armazenar um valor mesmo usando o limite de palavras do sistema para a representação de um valor, ocorre um erro chamado *OVERFLOW*, que pode ser traduzido livremente como ‘estouro de memória’. Isto significa que tentamos armazenar mais bits do que uma capacidade pré-estabelecida para uma variável.
- Ex.:

Tipo	Tamanho (em bytes)	Faixa Mínima
unsigned char	1	0 a 255
signed char	1	-128 a 127
long int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295