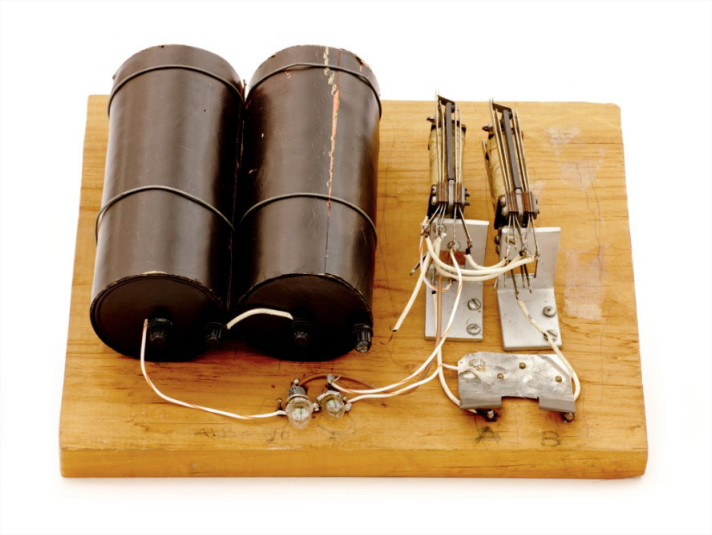
Sistema binário

De bits ao que interessa

Programas: fazendo o computador trabalhar para nós

Sistema binário

A calculadora Model K (George Robert Stibitz 1937)



A calculadora Model K (George Robert Stibitz 1937)

Trata-se de um circuito composto por dois interruptores dois relays e duas lâmpadas. Quando nenhum dos interruptores encontra-se ligado, nehuma das lâmpadas acende. Quando apenas um dos interruptores encontra-se ligado, apenas a lâmpada da direira acende. Quando os dois interruptores encontram-se ligados, apenas a lâmpada da esquerda acende.

A calculadora é alimentada com dois *sinais* — os estados dos interruptores (ligado / desligado) —, processa esses sinais, e produz dois resultados (*outputs*) — os estados das lâmpadas (acesa / apagada).

Tal tipo de circuito é chamado um *half adder*.

Os sinais de entrada são chamados binários, pois correspondem a dois estados possíveis: ligado e desligado. Ele podem ser associados aos valores zero (desligado) e um (ligado). O sinais de saída, podem ser interpretados como representando a soma dos dois números da seguinte maneira:

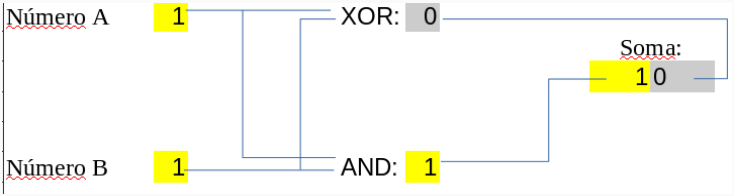
- primeiramente, definimos a variável $s0$ que assume valor 1 caso a lâmpada da direita esteja acesa e valor zero caso contrário;
- do mesmo modo, definimos a variável $s1$ que assume valor 1 caso a lâmpada da esquerda esteja acesa e valo zero caso contrário;
- a combinação dos dois sinais é interpretada como o número $s0 \times 2^0 + s1 \times 2^1 = s0 + 2s1$ e é igual à soma dos dois números.

Half adder, circuitos xor e and

Um *half adder* pode ser construído combinado dois circuitos:

- um circuito *ou exclusivo*, xor, que tem por *inputs* dois sinais binários e retorna como *output* um sinal binário que será ligado caso um, e apenas um dos sinais de entrada, seja ligado, e desligado caso contrário.
- um circuito *e*, and, que também tem por *inputs* dois sinais binários e retorna como *output* um sinal binário que será ligado caso os dois *inputs* sejam *ligados* e *desligado* caso contrário.

Representação esquemática de um *half adder*



O resultado deve ser interpretado como o valor do sinal à esquerda (1) vezes 2 mais o valor do sinal à direita (0) vezes 1.

Sistemas numéricos posicionais

A interpretação que demos ao resultado de nosso half adder está associada ao que chamamos de um *sistema numérico posicional* com *base* 2.

Em todos os sistemas numéricos, os números são representados por símbolos. No sistema numérico posicional, o valor atribuído a cada símbolo empregado na representação de um número depende da posição desse símbolo em relação aos outros símbolos. Isso ocorre, por exemplo, em nossa notação numérica tradicional que usa um sistema numérico posicional com base 10.

Sistema decimal

No sistema numérico decimal, empregamos 10 símbolos, denominados dígitos — 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 — para representar os números. A quantidade representada por um dígito depende da posição que ele ocupa. A contar da direita para a esquerda, o primeiro dígito de um número indica uma quantidade de unidades, o segundo dígito, indica dezenas, o terceiro, centenas e, de um modo geral o *n*-ésimo dígito indica multiplos de 10^{n-1}.

Por exemplo, o número 1237 é igual a

$$1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 7 \times 7^0.$$

De um modo geral, um número seja representado em uma base b qualquer por xyz, em que x, y e z são dígitos dessa base, então

$$xyz = x \times b^2 + y \times b^1 + z \times b^0.$$

Assumindo que w também seja um dígito dessa base, então

$$wxyz = w \times b^3 + x \times b^2 + y \times b^1 + z \times b^0.$$

Sistema numérico binário

Os computadores trabalham com sinais binários, basicamente, circuitos fechados (ligados) e abertos (desligados). Assim, o sistema numérico natural para interpretar seus *outputs* é o sistema numérico posicional de base 2, também chamado de sistema numérico binário.

Na notação usual do sistema binário, usamos os dígitos 0 e 1. É comum para indicar que um número é representado na base binária anteceder sua representação por `0b`. Usaremos essa convenção aqui. Além disso, os números binários serão representados com tipo de máquina de escrever.

Exemplo:

$$\texttt{0b1101} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1 = 13.$$

Exercício

Encontre as representações decimais dos seguintes números binários:

1. `0b10`
2. `0b100`
3. `0b1000`
4. `0b10000`
5. `0b100000`
6. `0b1000000`
7. `0b10000000`
8. `0b100000000`
9. `0b1111011`
10. `0b100101`
11. `0b101111011`

Exercício — respostas

Encontre as representações decimais dos seguintes números binários:

1. `0b10` $= 2$
2. `0b100` $= 4$
3. `0b1000` $= 8$
4. `0b10000` $= 16$
5. `0b100000` $= 32$
6. `0b1000000` $= 64$
7. `0b10000000` $= 128$
8. `0b100000000` $= 256$
9. `0b1111011` $= 123$
10. `0b100101` $= 37$
11. `0b101111011` $= 379$

Do sistema decimal para o binário

Para converter um número decimal inteiro em binário

1. Faça a divisão inteira do número por 2, coloque o resto na casa mais à direita, caso nenhuma posição tenha sido preenchida, ou na próxima posição caso alguma posição já tenha sido preenchida;
2. repita sucessivamente a operação anterior usando o quociente como número inicial até que este seja igual a zero.

Exemplo

Vamos encontrar a representação de 137:

Divisão por 2		Representação binária
resto	quociente	
$137 \mod 2 = 1$	$137 \div 2 = 68$	`_________1`
$68 \mod 2 = 0$	$68 \div 2 = 34$	`________01`
$34 \mod 2 = 0$	$34 \div 2 = 17$	`_______001`
$17 \mod 2 = 1$	$17 \div 2 = 8$	`______1001`
$8 \mod 2 = 0$	$8 \div 2 = 4$	`_____01001`
$4 \mod 2 = 0$	$4 \div 2 = 2$	`____001001`
$2 \mod 2 = 0$	$2 \div 2 = 1$	`___0001001`
$1 \mod 2 = 1$	$1 \div 2 = 0$	`0b10001001`

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

```
  1 1 0 1 0
    1 1 1 1
  ---------
```

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

```
  1 1 0 1 0
    1 1 1 1
-------------
          1
```

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

$$
\begin{array}{r}
\,{\scriptstyle 1} \\
1\,1\,0\,1\,0 \\
+\quad 1\,1\,1\,1 \\
\hline
0\,1
\end{array}
$$

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

$$
\begin{array}{r}
\,{\scriptstyle 1}\;{\scriptstyle 1} \\
1\;1\;0\;1\;0 \\
1\;1\;1\;1 \\
\hline
0\;0\;1
\end{array}
$$

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

$$
\begin{array}{r}
{\scriptstyle 1\ \ 1\ \ 1\ \ \ \ \ } \\
1\ 1\ 0\ 1\ 0 \\
\quad 1\ 1\ 1\ 1 \\
\hline
\quad 1\ 0\ 0\ 1
\end{array}
$$

Operações aritméticas com números binários

As operações aritméticas podem ser realizadas empregando algorítimos similares aos empregados na realização das operações aritméticas com números decimais.

Exemplo: Calcule `0b11010` + `0b1111`

$$\begin{array}{r} {\scriptstyle 1\ \ 1\ \ 1\ \ \ \ \ } \\ 1\ 1\ 0\ 1\ 0 \\ \quad 1\ 1\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 0\ 1 \end{array}$$

bite, byte, kilobyte, etc

Um sinal binário simples (do tipo 0 ou 1 / ligado ou desligado) é chamado um *bit*. Um bit pode assumir apenas dois valores.

Um *byte* é a combinação de 8 bits. Um byte pode assumir $2^8 = 64$ valores;

Um *kilobyte* (KB) corresponde a 2^{18} bits ou $2^{10} = 1024$ bytes.

Um *megabyte* (MB) corresponde a 2^{28} bits ou 2^{10} bytes.

Um *gigabyte* (GB) corresponde a 2^{38} bits ou 2^{10} megabytes.

Um *terabyte* (TB) corresponde a 2^{48} bits ou 2^{10} terabytes.

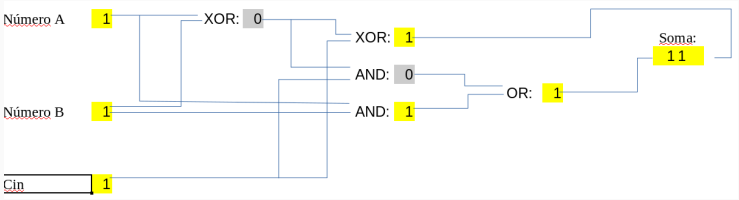
De modo análogo, definimos, sucessivamente, Petabyte (PB), Exabyte (EB), Zettabyte (ZB) e Yottabyte (YB).

Full Adder

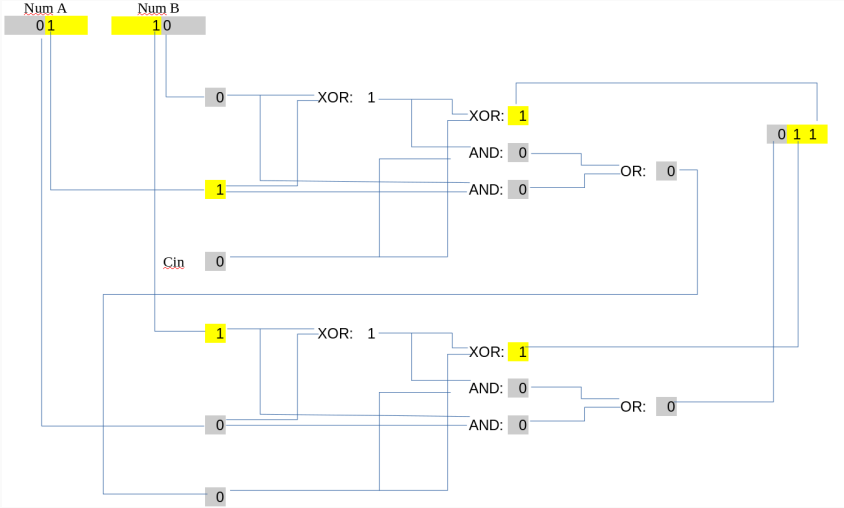
O half adder é capaz de somar 2 bits. Se quisermos somar um número representado por mais do que 2 bits, podemos usar o algorítimo da soma, descrito anteriormente. Porém, esse algorítimo requer a capacidade de somar até três bits.

O *full adder* é um circuito capaz de somar até 3 bits.

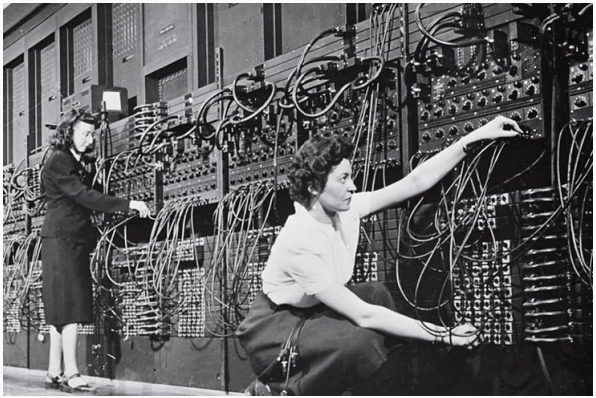
Representação esquemática de um full adder



Realizando uma soma de dois números de 2 bits com full adders encadeados

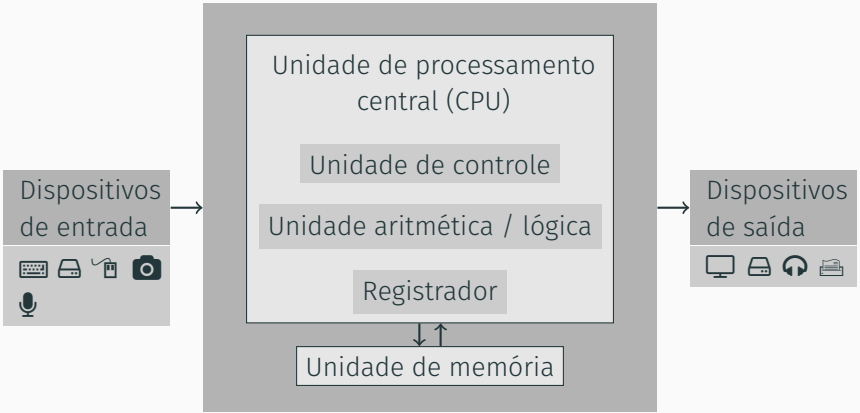


Programação dos primeiros computadores

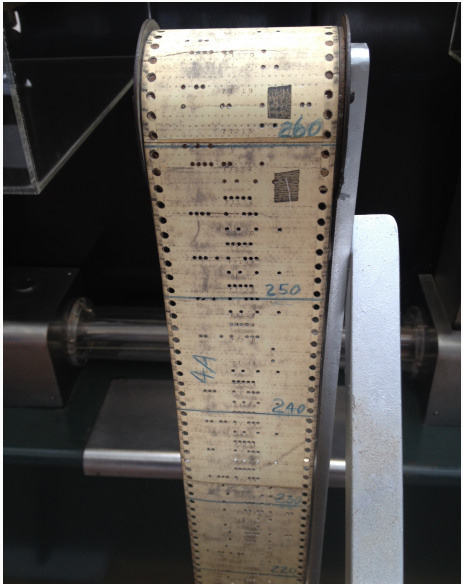


Os primeiros computadores eram uma coleção de circuitos como o anterior. Eles não eram capazes de ler instruções previamente gravadas. A programação era feita ajustando-se cabos, botões, e interruptores.

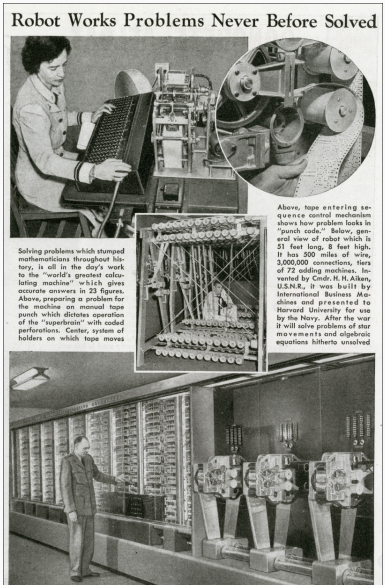
A arquitetura de Von-Neumann



Midia de entrada de dados e programa



Midia de entrada de dados e programa

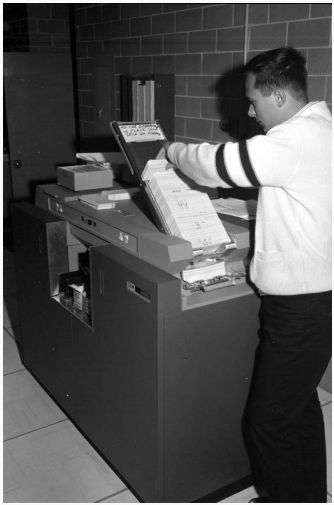


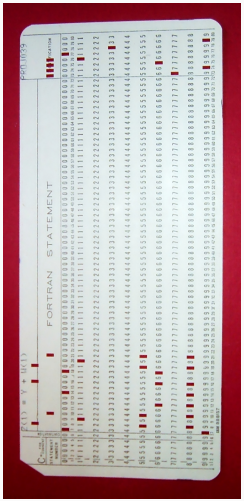
Robot Works Problems Never Before Solved

Solving problems which stumped mathematicians throughout history, is all in the day's work to the "world's greatest calculating machine" which gives accurate answers in 23 figures. Above, preparing a problem for the machine on manual tape punch which dictates operation of the "superbrain" with coded perforations. Center, system of holders on which tape moves

Above, tape entering sequence control mechanism shows how problem looks in "punch code." Below, general view of robot which is 51 feet long, 8 feet high. It has 500 miles of wire, 3,000,000 connections, tiers of 72 adding machines. Invented by Cmdr. H. H. Aiken, U.S.N.R., it was built by International Business Machines and presented to Harvard University for use by the Navy. After the war it will solve problems of star movements and algebraic equations hitherto unsolved

Cartão perfurado e leitora de cartões





Método dominante de entrada de dados até início da década de 1980.

Principal dispositivo de saída no século passado

Linguagem de máquina

Os programas passam à unidade de controle uma sequência de instruções que são lidas sequencialmente, a menos que uma instrução indique o contrário. Cada instrução é uma combinação de bits. A depender da arquitetura, o número de bits das instruções pode ser constante ou variável. Alguns bits da instrução têm por função “dizer” à unidade de controle o que ela deve fazer com os bits restantes contidos na instrução, os outros bits podem conter dados, instruções adicionais e/ ou endereços de memória.

Exemplo de linguagem de máquina — arquitetura MIPS

- Instruções de 32 bits;
- Os seis primeiros bits informam o tipo de instrução que pode ser uma operação, uma instrução de salto ou imediata;
- O significado dos outros bits depende do código do operador.

Exemplo: A instrução abaixo é do tipo operação (6 primeiros bits = `000000`) e faz a CPU adicionar (seis último bits, `100000`) os conteúdos dos registro 1 (`00001`) e 2 (`00010`) no registro 6(`00110`).

```
000000 00001 00010 00110 00000 100000
```

Sistema hexadecimal

Frequentemente, a notação de máquina é representada em um sistema hexadecimal, isso é um sistema numérico com base 16. A correspondência de símbolos é descrita abaixo

Bin.	Dec.	Hex.
0b0	0	0x0
0b1	1	0x1
0b10	2	0x2
0b11	3	0x3
0b100	4	0x4
0b101	5	0x5
0b110	6	0x6
0b111	7	0x7

Bin.	Dec.	Hex.
0b1000	8	0x8
0b1001	9	0x9
0b1010	10	0xa
0b1011	11	0xb
0b1100	12	0xc
0b1101	13	0xd
0b1110	14	0xe
0b1111	15	0xf

Exemplos

A instrução

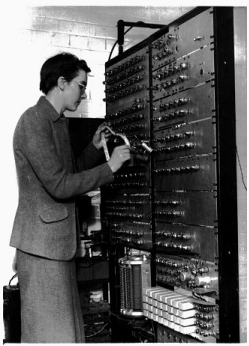
```
00000000001000100011000000100000
```

é escrita no sistema hexadecimal como

```
223020
```

O número decimal 62285 tem representação binária `0b1111001101001101` e representação hexadecimal `0xf34d`.

Linguagem *assembly* (ou *assembler*)



Katleen Elizabeth Booth foi uma pioneira em programação de computadores. Ela desenvolveu a primeira linguagem assembly ou assembler. A linguagem estabelece uma correspondência entre os comandos da linguagem de máquina e *mneumônicos*, palavras curtas, mais fáceis de serem lidos e reconhecidos por humanos.

Exemplo: a instrução

```
000000 00001 00010 00110 00000 100000
```

é escrita na linguagem assembly do MPIS como

```
add $t1, $t2, $t6.
```

Assembler

Um assembler é um programa de computador que traduz um código escrito em assembly para a linguagem de máquina.

Cada arquitetura de CPU tem uma linguagem assembly diferente.

Linguagens de alto nível

Linguagens de alto nível são linguagens cujas instruções não encontram correspondente direto na linguagem de máquina. Dizemos que elas *abstraem* da estrutura da CPU. Para ser implementada, uma linguagem de alto nível precisar ser traduzia em linguagem de máquina. Isso pode ser feito por uma compilador ou por um interpretador:

Compilador traduz um programa escrito em uma linguagem de alto nível em outro programa em linguagem de máquina, linguagens que usam compiladores são chamadas linguagens compiladas;

Interpretador traduz cada instrução em linguagem de alto nível em linguagem de máquina e a executa instantaneamente; linguagens que usam interpretador são chamadas linguagens interpretadas.

Linguagens de alto nível vs. linguagens de baixo nível

Linguagens de alto nível tornam a programação mais fácil. Os compiladores e interpretadores tornam automáticas tarefas repetitivas. Os códigos de linguagens de alto nível são compreendidos mais facilmente por seres humanos.

Por outro lado, os compiladores e, ainda mais, os interpretadores podem ser ineficientes no momento de realizar a tradução de instruções em linguagem de alto nível para linguagem de máquina. Assim, a programação em assembly pode ser vantajosa quando a necessidade de desempenho é fundamental.

Note, todavia, que um programador mediano terá muita dificuldade em gerar um código mais eficiente daquele que seria gerado por uma linguagem compilada, como, por exemplo, o C.

Linguagens interpretada vs. linguagens compiladas

O processo de compilar um programa é mais demorado do que o processo de interpretá-lo, isso faz com que linguagens interpretadas sejam mais adequadas para códigos que serão executados poucas vezes ou que se encontram em faze de elaboração.

Todavia, um programa compilado é mais rápido do que um programa interpretado. Isso faz com que as linguagens

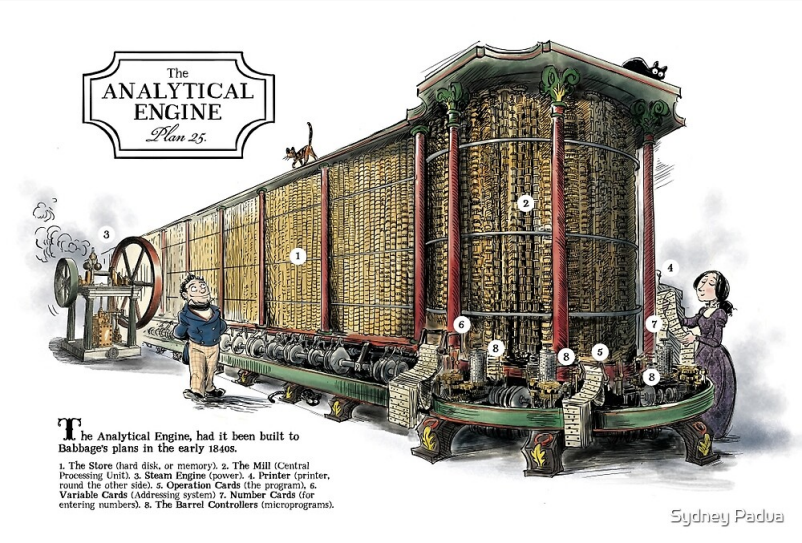
Para algumas aplicações, o ganho de velocidade advindo da compilação é imperceptível, dado que os computadores modernos são bastante velozes.

Linguagens mais populares (2018)

Nome	propósito	tipo
JavaScript	geral / web	interpretada
HTML	web	interpretada
SCC	web	interpretada
SQL	banco de dados	interpretada
Java	propósito geral	compilada para máquina virtual
Bash/Shell	scripts de console linux, unix	interpretada
Python	propósito geral	interpretada
C#	propósito geral	compilada
PHP	propósito geral	interpretada

De bits ao que interessa

Charles Babbage, Ada Lovelace e *the analytical engine* (1840)



A profecia de Ada Lovelace

[The analytical engine] pode agir sobre outras coisas além de números,nas quais se encontram objetos cujas relações mútuas fundamentais podem ser expressas por aquelas da ciência abstrata das operações, e as quais devem ser também suscetíveis de adaptação à ação da notação operacional e do mecanismo da máquina... Supondo, por exemplo, que as relações fundamentais de sons afinados na ciência da harmonia e da composição musical sejam suscetíveis de tal expressão e adaptações, a máquina pode compor peças musicais elaboradas de qualquer grau de complexidade e extensão.

Informações digitais

Informações digitais são informações representadas de modo discontínuo empregando-se um conjunto contável e finito de sinais (símbolos) armazenados (com repetição) de modo ordenado.

Exemplos:

Língua escrita.

Código morse.

Vídeos e audios digitais.

Dados em computadores, etc.

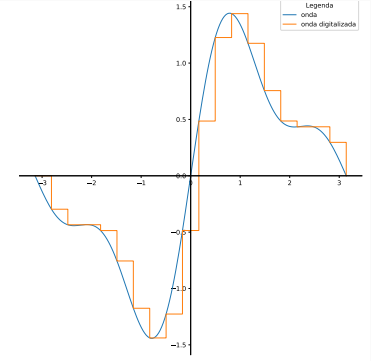
Qualquer informação digital pode ser armazenada sem perda de conteúdo em um sistema binário. Basta que haja um código de conversão.

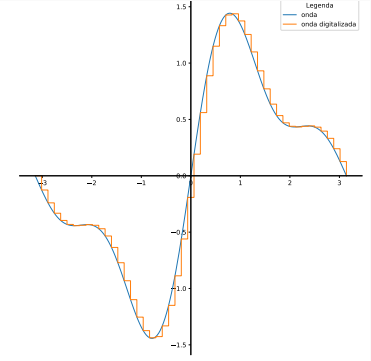
Exemplo: conversão de letras em números

Decimal	Binário	Hexadecimal	Símbolo
65	0b1000001	0x41	A
66	0b1000010	0x42	B
67	0b1000011	0x43	C
68	0b1000100	0x44	D
69	0b1000101	0x45	E
70	0b1000110	0x46	F
71	0b1000111	0x47	G
72	0b1001000	0x48	H
73	0b1001001	0x49	I
74	0b1001010	0x4a	J
75	0b1001011	0x4b	K
76	0b1001100	0x4c	L

Representação de dados contínuos

A transformação de dados contínuos em digitais implica alguma perda de informação.





À esquerda a frequência de amostragem para a geração do dado digitalizado é menor, gerando um ajuste pior.

Exemplo: foto digital

Programas: fazendo o computador trabalhar para nós

Hardware e software

Hardware: Elementos físicos dos computadores — placa, circuitos, dispositivos, etc. — também chamados equipamentos ou maquinário.

Software: O conjunto de programas que fazem o computador trabalhar em nosso benefício. É frequente classificar os softwares em três categorias: software de sistema, aplicativos e ferramentas de programação.

Classificação do software

Software de sistema: conjunto de programas que organizam a execução de tarefas pelo computador, o armazenamento de dados e o acesso a periféricos.

Aplicativos: programas usados para executar tarefas bem delimitadas pelos usuários finais: processadores de texto, planilhas de cálculo, criadores de apresentação, etc.

Ferramentas de programação: programas que auxiliam desenvolvedores a elaborar outros programas.

A divisão entre as três categorias nem sempre é precisa. Por exemplo, um editor de texto pode ser usados tanto para produzir um documento quanto para escrever um código de computador.

Computadores sem sistema operacional.

Os primeiros computadores rodavam um programa por vez. Cada programa deveria controlar todo o computador e seus dispositivos.

O carregamento dos programas era feito sequencialmente por operadores humanos.

Tarefas comuns tais como buscar dados em um dispositivo de entrada ou enviar dados a um dispositivo de saída deveriam ser definidas em cada programa.

Fase intermediária

Com o tempo foram introduzidos novos recursos tais como:

- Programas que poderiam ser chamados pelo programa do usuário para executar tarefas comuns tais como operações de entrada e saída e compilação de códigos em linguagem de alto nível (anos 50);
- Automatização do carregamento dos programas que eram lidos e armazenados na memória e executados seguindo critérios de prioridade (anos 50);
-