Aula 3

Docupedia Export

Author:Goncalves Donathan (CtP/ETS)
Date:25-Aug-2023 19:35

Table of Contents

1 Introdução à programação python

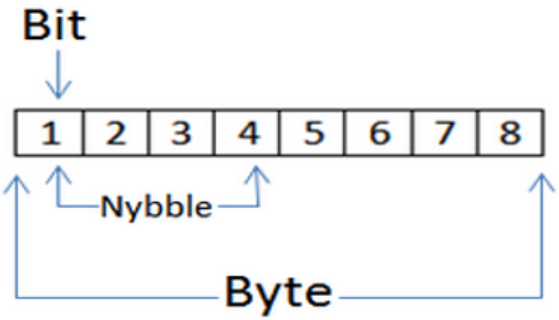
1.1 Apresentação da linguagem python e seu ambiente de desenvolvimento

1.1.1 Por que aprender a programar?

- Implementação de métodos computacionais para resolução de problemas
- Análise e comparação de métodos diferentes
- Conjunção de várias competências:
    - Matemática linguagens formais para especificar processos;
    - Engenharia juntar componentes para formar um sistema; avaliar prós/contras de alternativas
    - Ciências naturais observar comportamento de sistemas complexos; formular hipóteses; testar previsões

1.1.2 Bases Numéricas

BIT: é a sigla para **Binary Digit.** É a menor unidade de informação que pode ser armazenada ou transmitida. Um bit pode assumir somente 2 valores, como 0 ou 1.
BYTE: é uma **unidade de informação digital** equivalente a oito bits. O símbolo do byte é um (B) maiúsculo, para diferenciar de bit (b).
Nibble: é a metade de um byte (4 bits)



Bit, Nybble, Byte

O homem contemporâneo está familiarizado com a base **10** (decimal), no dia-a-dia, já os computadores atuais trabalham exclusivamente com a base **2** (binário), assim é preciso fazer conversões entre estas bases quando se pretende inserir algum valor para ser processado pelo computador. A diferença de interpretação se mostra a seguir:

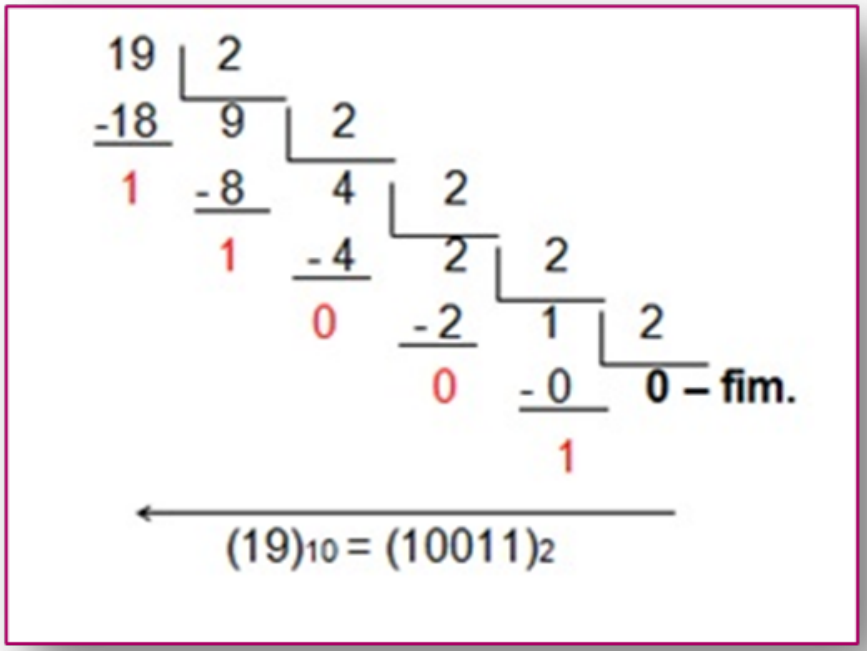
$$1010_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 0 + 2 + 0 = 10_{10},$$

$$10 = 1 \cdot 10^1 + 0 \cdot 10^0.$$

Onde se usou o índice **2** para indicar que o número inicial **1010** se encontra representado no sistema de numeração binário, ou base 2, enquanto que o número final **10** está representado no sistema de numeração decimal, ou base 10.

1.1.3 Conversão de Decimal Para Binário

A técnica de **divisões sucessivas** é utilizada para conversão de números inteiros do sistema decimal para o binário. Esta técnica consiste em dividir o número original pela base 2, o resto da divisão será um dígito e o resultado da divisão é novamente dividido por 2. Esta última etapa se repete até que o resultado da divisão seja zero. Para melhor compreensão do método, a imagem ao lado mostra um exemplo de conversão do número decimal 19 para binário.

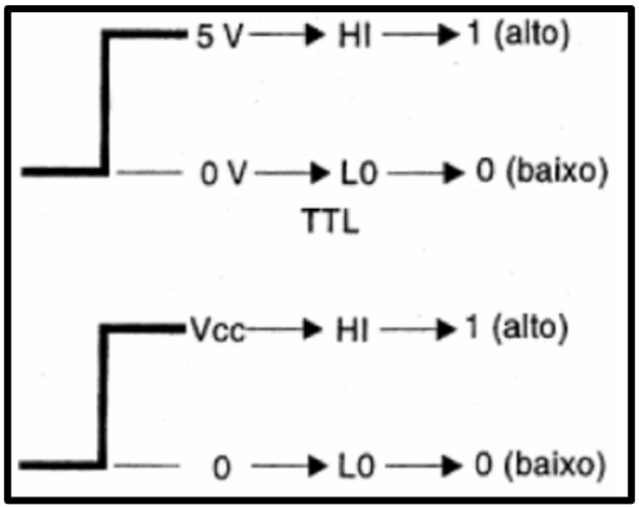


Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binário	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

1.1.4 Outras Bases Numéricas

Convertendo 111010110_2 para a base 16

Para cada algarismo em hexadecimal são necessários 4 algarismos para realizar sua representação em binário. Então o primeiro passo é separar o valor em base 2 em blocos de 4 algarismos:
111010110_2 = 1.1101.0110
Depois, consultando a tabela podemos converter o valor de cada bloco para seu equivalente hexadecimal. Assim teremos:
111010110_2 = 1.D.6= 〖1D6〗_16

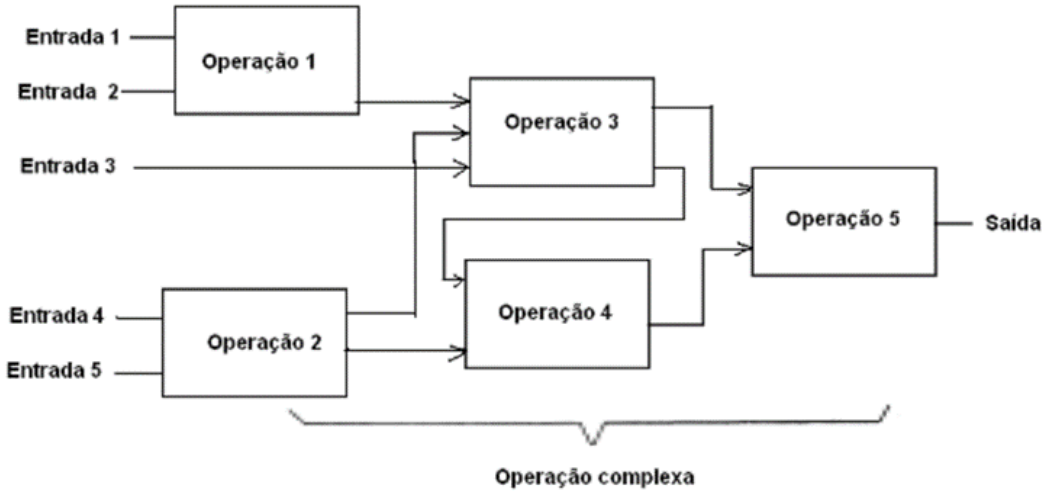


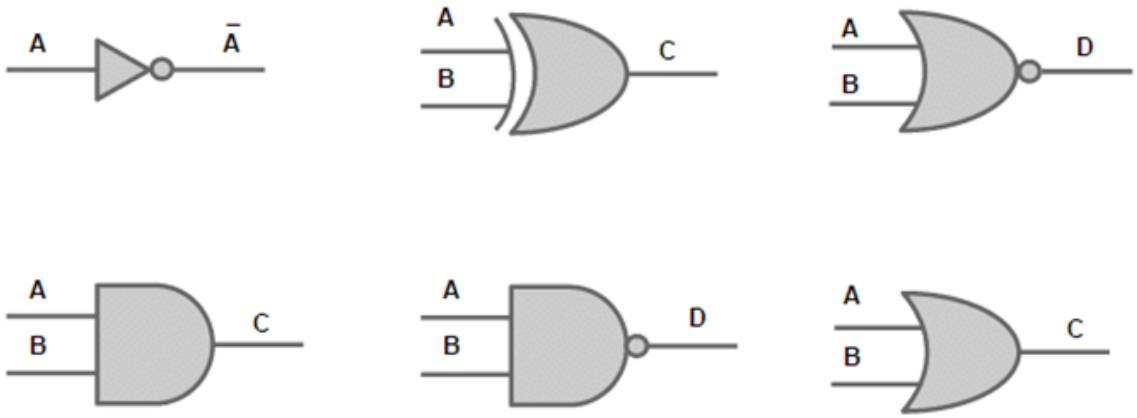
Na Álgebra de Boole, as variáveis lógicas só podem adquirir dois estados: 0 ou 1, Verdadeiro ou Falso, Aberto ou Fechado, Alto ou Baixo (HI ou LO), Ligado ou Desligado.
Um circuito digital só pode trabalhar com dois estados possíveis: presença de sinal ou ausência de sinal o que faz com que ela se adapte perfeitamente aos princípios da Álgebra de Boole.
Nos circuitos digitais, a presença de uma tensão será indicada como 1 ou HI (de HIGH ou Alto), enquanto que a ausência de uma tensão será indicada por 0 ou LO (de LOW ou baixo).
O 0 ou LO será sempre uma tensão nula, ou ausência de sinal num ponto do circuito, mas o nível lógico 1 ou HI pode variar conforme o circuito considerado, conforme mostra a figura.

1.1.5 Álgebra de Boole

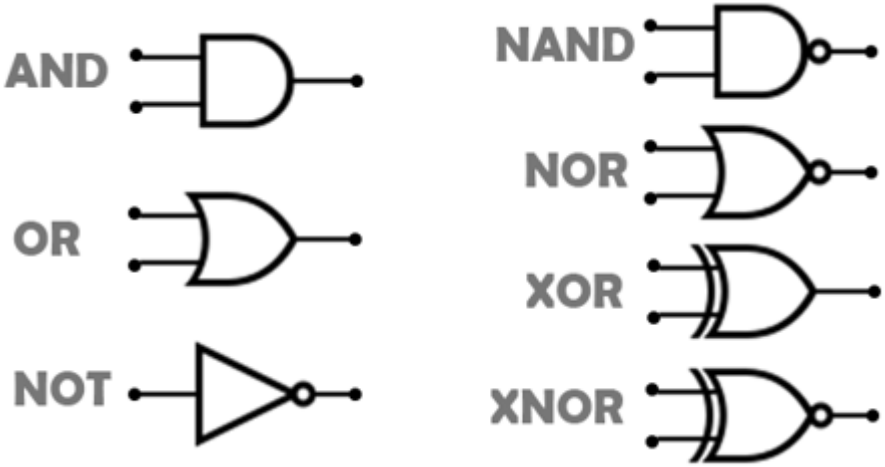
Podemos representar uma soma como:
Y = A + B
Onde o resultado de Y depende dos valores que vamos atribuir às variáveis A e B.
Neste caso, uma função algébrica em que Y é a variável dependente enquanto A e B são as variáveis independentes.

Na eletrônica digital, entretanto, existem operações mais simples do que a soma, e que podem ser implementadas levando em conta a utilização da Álgebra Booleana.
A saída do circuito digital é determinado pela função e pelo estado das entradas. A resposta/saída de cada circuito lógico depende da "regra booleana" implementada por ele.





Nos primórdios da eletrônica, todos os problemas eram solucionados por meio de sistemas analógicos. Com o avanço da tecnologia, os problemas passaram a ser solucionados pela eletrônica digital.
Na eletrônica digital, os sistemas (computadores, processadores de dados, sistemas de controle, codificadores, decodificadores, etc) empregam um pequeno grupo de circuitos lógicos básicos, que são conhecidos como portas lógicas.
Em qualquer bloco (porta ou função) lógico somente esses dois estados (0 ou 1) são permitidos em suas entradas e saídas.
Uma variável booleana também só assume um dos dois estados permitidos (0 ou 1)



Nesta apresentação trataremos dos seguintes blocos lógicos

- E (AND)
- OU (OR)
- NÃO (NOT)
- NÃO E (NAND)
- NÃO OU (NOR)
- OU EXCLUSIVO (XOR)

Por fim, veremos a correspondência entre expressões, circuitos e tabelas verdade

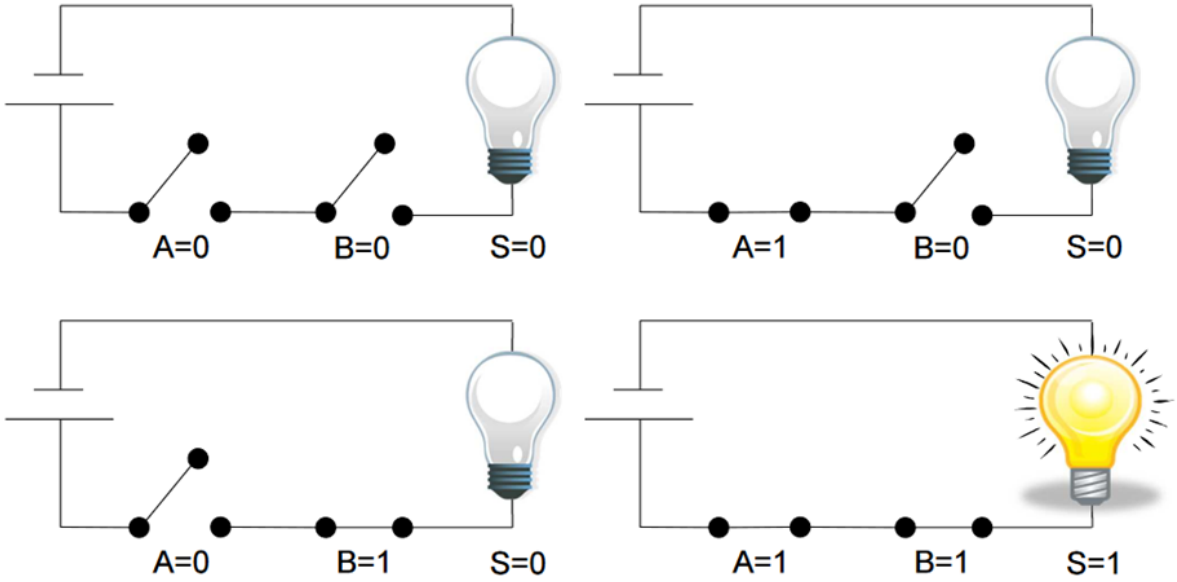
1.1.6 Porta E (AND)

Executa a multiplicação (conjunção) booleana de duas ou mais variáveis binárias. Por exemplo, assuma a convenção no circuito

Chave aberta = 0; Chave fechada = 1

Lâmpada apagada = 0; Lâmpada acesa = 1

Para representar a expressão
S = A e B
Adotaremos a representação S = A.B, onde se lê S = A e B.
Porém, existem notações alternativas:
S = A & B
S = A, B
S = A ∧ B

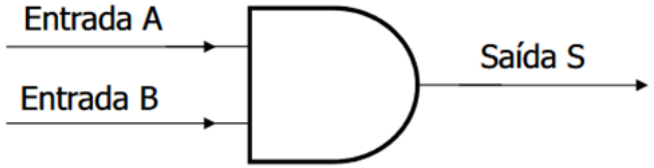


A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

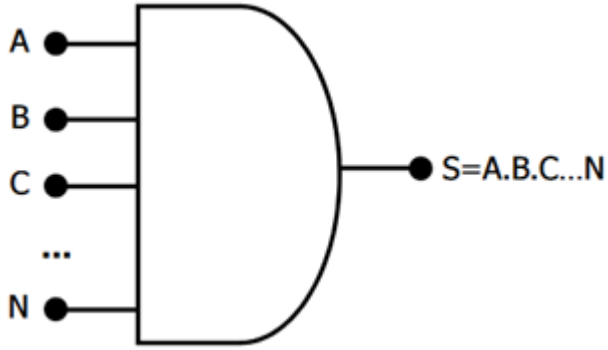
A tabela verdade é um mapa onde são colocadas todas as possíveis interpretações (situações), com seus respectivos resultados. Como visto no exemplo anterior, para 2 variáveis booleanas (A e B), há 4 interpretações possíveis. Para N variáveis booleanas de entrada, há 2^N interpretações possíveis

A porta E é um circuito que executa a função E, isto é, a tabela verdade da função E
Portanto, a saída será 1 somente se ambas as entradas forem iguais a 1; nos demais casos, a saída será 0.

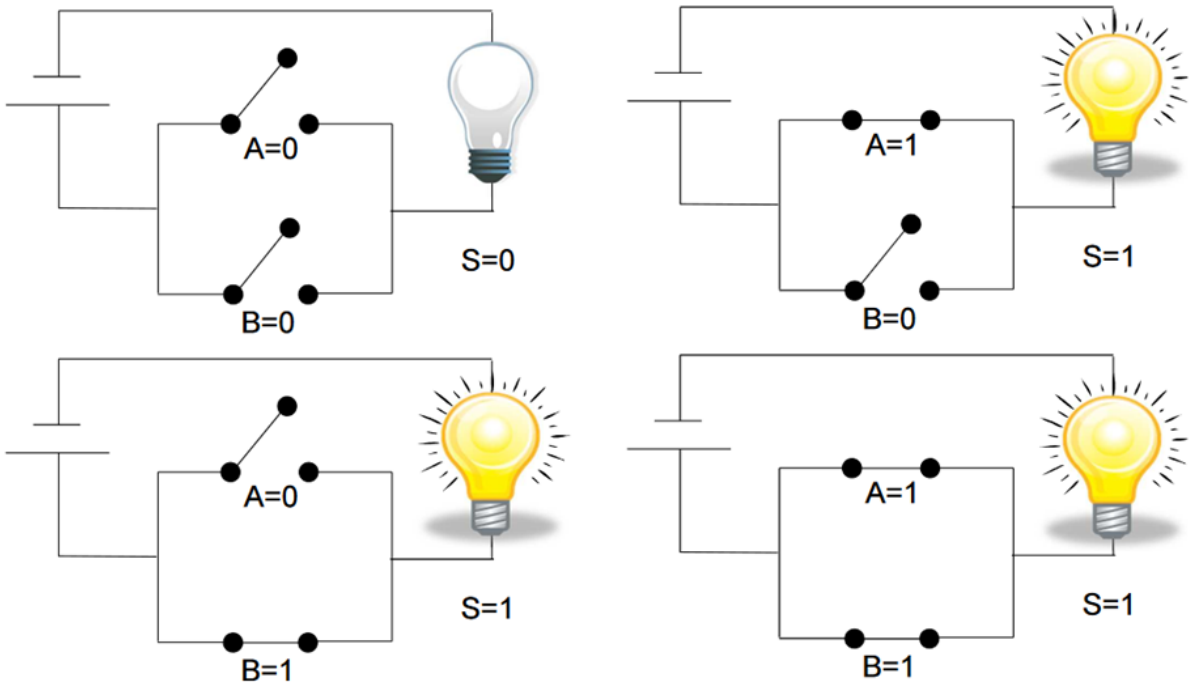
Porta **E**
(**AND**)



É possível estender o conceito de uma porta E para um número qualquer de variáveis de entrada Nesse caso, temos uma porta E com N entradas e somente uma saída
A saída será 1 se e somente se as N entradas forem iguais a 1; nos demais casos, a saída será 0



1.1.7 Porta OU (OR)



Executa a soma(disjunção) booleana de duas ou mais variáveis binárias. Por exemplo, assuma a convenção no circuito:
Chave aberta = 0; Chave fechada = 1
Lâmpada apagada = 0; Lâmpada acesa = 1

Para representar a expressão

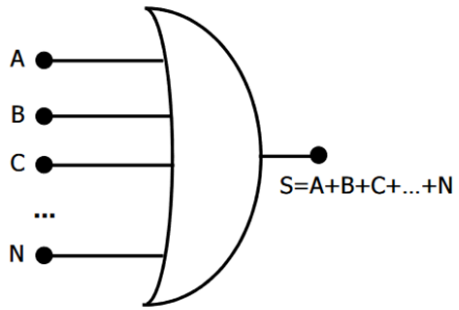
S = A ou B
Adotaremos a representação
S = A+B, onde se lê S = A ou B. Porém, existem notações alternativas
S = A | B

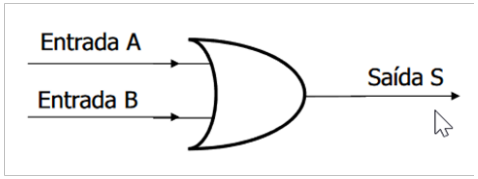
S = A; B
S = A ∨ B
Observe que, no sistema de numeração binário, a soma 1+1=10. Na álgebra booleana,

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

1+1=1, já que somente dois valores são permitidos (0 e 1)

A porta OU é um circuito que executa a função OU, isto é, executa a tabela verdade da função OU. Portanto, a saída será 0 somente se ambas as entradas forem iguais a 0; nos demais casos, a saída será 1.
Também é possível estender o conceito de uma porta OU para um número qualquer de variáveis de entrada.





1.1.8 Porta Não (Not)

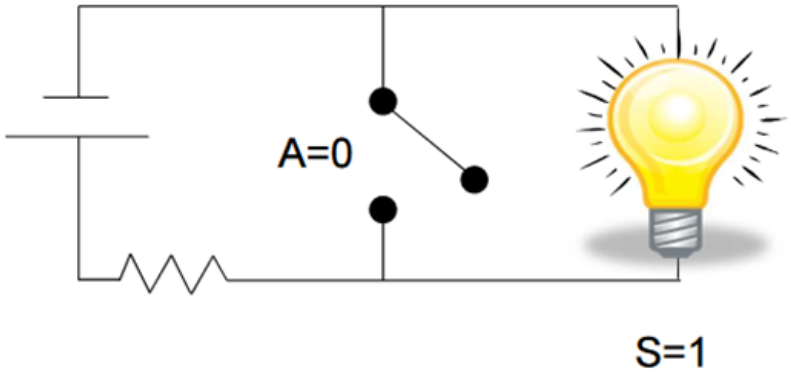
Executa o complemento(negação) de uma variável binária:

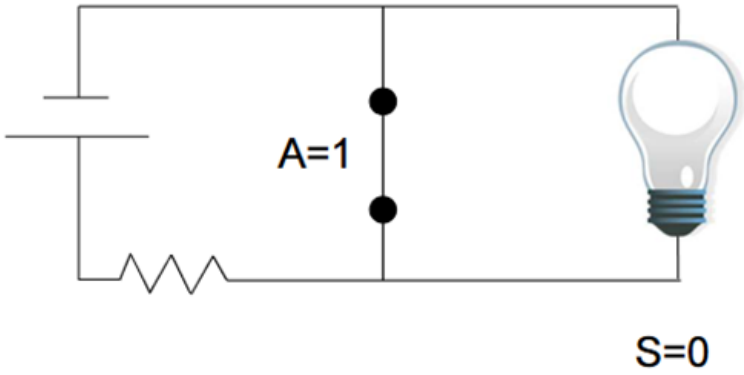
- Se a variável estiver em 0, o resultado da função é 1.
- Se a variável estiver em 1, o resultado da função é 0.

Essa função também é chamada de inversora

Usando as mesmas convenções dos circuitos anteriores, tem-se que:

- Quando a chave A está aberta (A=0), passará corrente pela lâmpada e ela acenderá (S=1)
- Quando a chave A está fechada (A=1), a lâmpada estará em curto-circuito e não passará corrente por ela, ficando apagada (S=0)



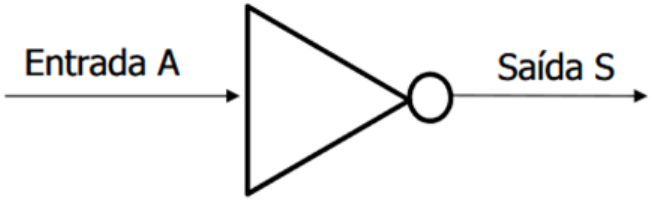


Para representar a expressão
S = não A
Adotaremos a representação
S = Ā, onde se lê S = não A
Notações alternativas:
S = A'
S = ¬A
S = Ã

Tabela verdade da função NÃO (NOT):

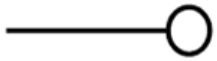
A	$\bar{A}$
0	1
1	0

NÃO
(NOT)



Alternativamente,

Após um bloco lógico



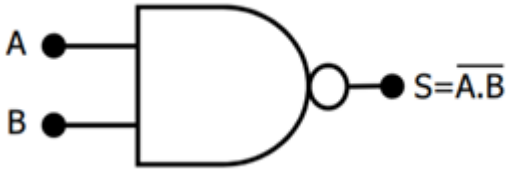
Antes de um bloco lógico

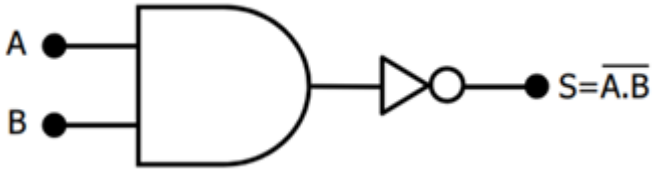
A porta lógica NÃO, ou inversor, é o circuito que executa a função NÃO:
Se a entrada for 0, a saída será 1;
Se a entrada for 1, a saída será 0.

1.1.9 Porta Não E (NAND)

Composição da função E com a função NÃO, ou seja, a saída da função E é invertida

$$S = \overline{(A.B)} = \overline{A.B} = (A.B)' = \neg(A.B)$$





- Como a porta E, a porta NÃO E pode ter duas ou mais entradas.
- Nesse caso, temos uma porta NÃO E com N entradas e somente uma saída.
- A saída será 0 se e somente se as N entradas forem iguais a 1; nos demais casos, a saída será 1.

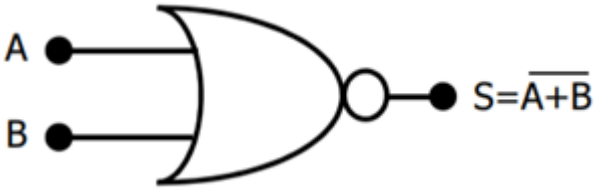
A	B	S=$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

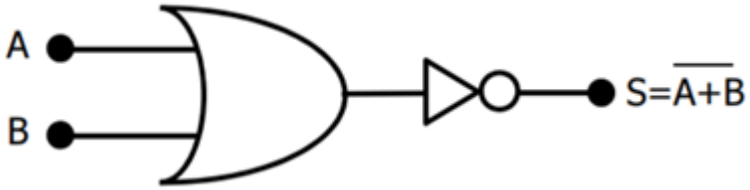
1.1.10 Porta Não OU (NOR)

A	B	S=$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Composição da função OU com a função NÃO, ou seja, a saída da função OU é invertida

$$S = \overline{(A+B)} = \overline{A+B} = (A+B)' = \neg(A+B)$$





- Como a porta OU, a porta NÃO OU pode ter duas ou mais entradas.
- Nesse caso, temos uma porta NÃO OU com N entradas e somente uma saída.
- A saída será 1 se e somente se as N entradas forem iguais a 0; nos demais casos, a saída será 0.

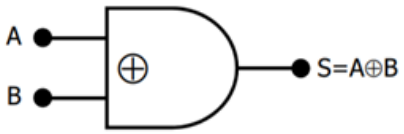
1.1.11 Porta OU Exclusivo (XOR)

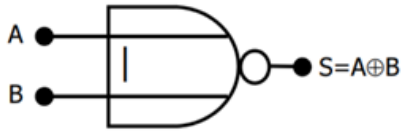
A função OU Exclusivo fornece

- 1 na saída quando as entradas forem diferentes entre si e
- 0 caso contrário

A	B	S=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Outros símbolos utilizados





$$S = A \oplus B = \bar{A}.B + A.\bar{B}$$

1.2 Linguagens de Programação

Linguagens formais para exprimir computação

- sintaxe: regras de formação de frases (gramática)
- semântica: significado associado a cada frase

	sintaxe	semântica
$3 \times (1 + 2)$	ok	9
$3 \times 1 + 2$	ok	5
$\times)1 + 2 + (3$	erro	—
H_2O	ok	água
$_2zZ$	erro	—

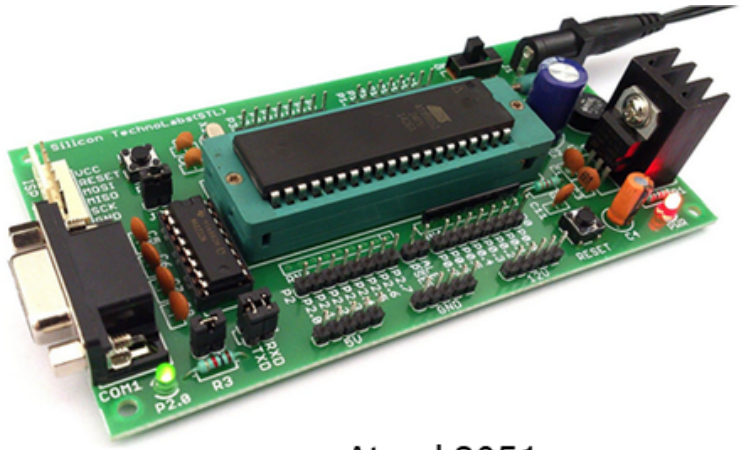
Código máquina

55 89 e5 83 ec 20 83 7d 0c 00 75 0f ...

- Códigos numéricos associados a operações básicas;
- Linguagem específica de cada microprocessadores;
- Única linguagem diretamente executável pelo computador;
- Difícil escrever programas diretamente em código máquina;
- Concebida para facilitar a implementação usando circuitos eletrônicos

1.2.1 ASSEMBLY

- Representação do código máquina usando **mnemónicas**
- Mais legível do que a linguagem máquina
- Pode ser traduzida automaticamente para código máquina
- Continua a ser específica para cada micro-processador
- Exige programação lenta e mais suscetível a erros àUsada apenas em contextos muito específicos



Atmel 8051

```
55                  push   %ebp
89 e5               mov    %esp,%ebp
83 ec 20            sub    $0x20,%esp
83 7d 0c 00         cmpl   $0x0,0xc(%ebp)
75 0f               jne    1b
...                 ...
```

1.2.2 Interpretadores Versus Compiladores

As linguagens de alto-nível são traduzidas para código máquina por programas especiais:
Interpretador: a tradução é efetuada de cada vez que o programa executa
Compilador: tradução é efetuada uma única vez

Vantagens dos interpretadores: permitem uso interativo rápido, facilitam testar fragmentos de programas, são mais simples de implementar
Vantagens dos compiladores: permitem gerar código máquina mais eficiente; geram programas independentes (o compilador não tem de estar presente durante execução)