Assembly Language for Intel-Based Computers, 5th Edition

Kip Irvine

Capítulo 1: Conceitos básicos

Slides prepared by the author

Revision date: June 3, 2006

Conteúdo

- **Linguagem Assembly**
- Conceito de máquina virtual
- Representação dos dados
- Operações booleanas

Algumas Questões

- Por que estudar Linguagem Assembly?
- Que base seria necessária?
- O que é um assembler?
- Que hardware/software necessito?
- Que tipo de programas eu posso criar?
- O que vou aprender?

Outras questões *(cont)*

- Como uma linguagem assembly se relaciona a uma linguagem de máquina?
- Como C++ e Java se relaciona com assebly?
- Assembly é portável?
- Por que aprender assembly ?

Aplicações da linguagem Assembly

- Alguns tipos representativos de aplicações:
  - Aplicação comercial para uma plataforma única
  - Driver de um dispositivo de Hardware
  - Aplicações comerciais para plataformas múltiplas
  - Sistemas embarcados & jogos por computador

(ver painel seguinte)

Comparando ASM com linguagens de alto nível

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Seção seguinte

- **Boas vindas à linguagem Assembly**
- Conceito de máquina virtual
- Representação dos dados
- Operações booleanas

Máquinas virtuais

- Tanenbaum: conceito de máquina virtual
- Analogia com linguagem de programação:
  - Cada computador tem uma linguagem de máquina nativa (linguagem L0) que roda diretamente no hardware
  - Uma linguagem mais humanamente amigável é usualmente construida acima da linguagem de máquina, chamada linguagem L1
- Programas escritos em L1 podem ser executados em duas formas :
  - Interpretação – o programa L0 interpreta e executa instruções em L1 um a um
  - Tradução – o programa L1 é completamente traduzido para um programa na linguagem L0, que então é executado no hardware

Traduzindo linguagens

English: mostra a soma de A vezes B mais C.

C++: cout << (A * B + C);

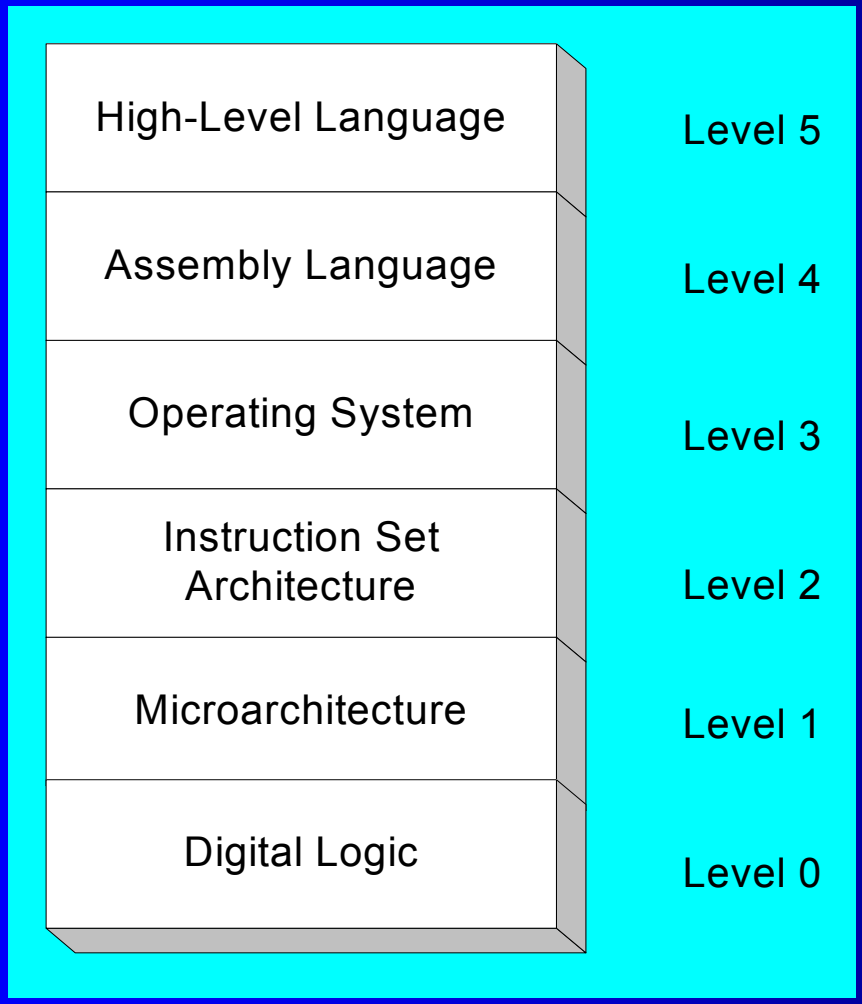
Linguagem Assembly :

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Linguagem de máquina Intel :

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

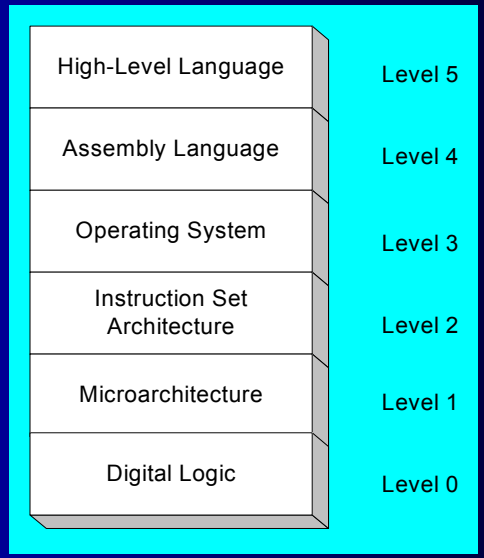
Níveis específicos de máquina



(descriptions of individual levels follow . . .)

Linguagem de alto nível

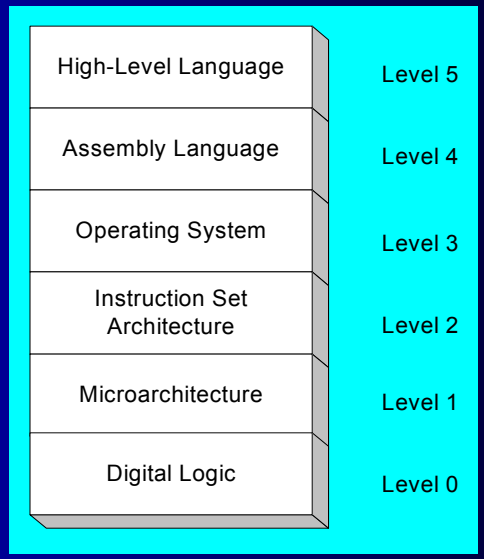
- Nível 5
  - Linguagens orientadas a aplicações
    - C++, Java, Pascal, Visual Basic . . .
  - Os programas são compilados para a linguagem assembly (nível 4)



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Linguagem Assembly

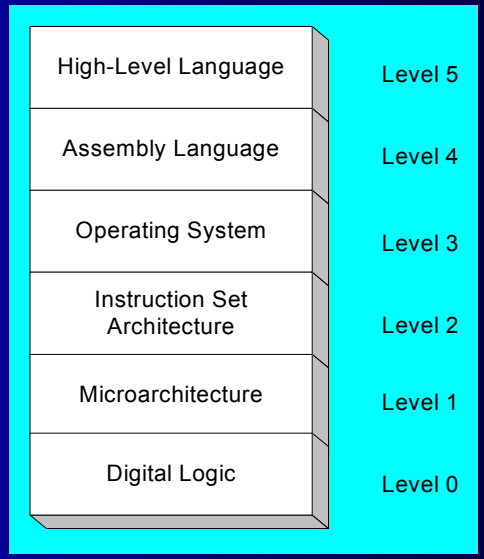
- Nível 4
  - Mnemônicos de instruções em correspondência um-a-um com as instruções em linguagem de máquina
  - Chama funções escritas no nível de sistema operacional (nível 3)
  - Os programas são traduzidos para a linguagem de máquina (nível 2)



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sistema operacional

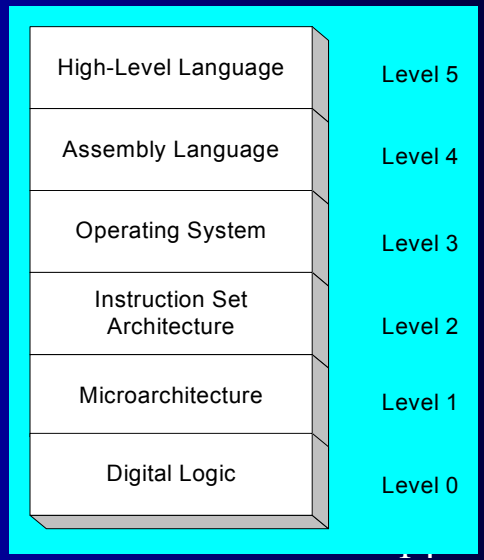
- Nível 3
  - Fornece serviços para os programas de nível 4
  - São traduzidos e executados no nível de arquitetura do conjunto de instruções (instruction set architecture) (nível 2)



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

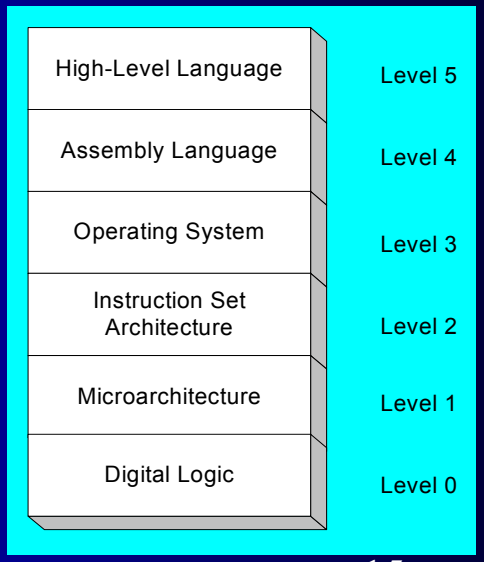
Arquitetura do conjunto de instruções (Instruction Set Architecture)

- Nível 2
  - Também conhecido como linguagem de máquina
  - Executado pelo programa no nível 1 (microarquitetura)



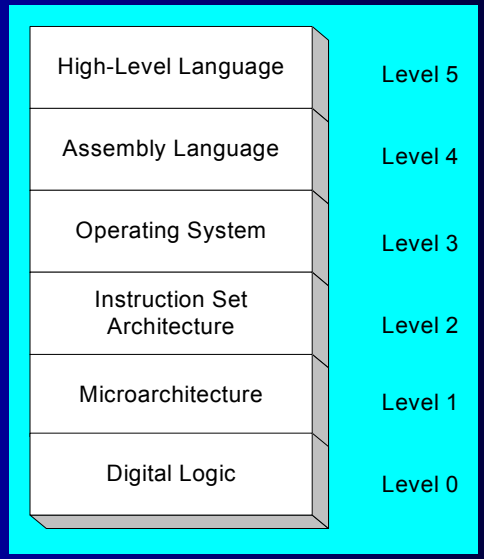
Microarquitetura

- Nível 1
  - Interpreta instruções de máquina convencional (nível 2)
  - Executado pelo hardware digital (nível 0)



Lógica Digital

- Nível 0
  - CPU, construído de portas lógicas digitais
  - Barramento do sistema
  - Memória
  - Implementada usando transistores



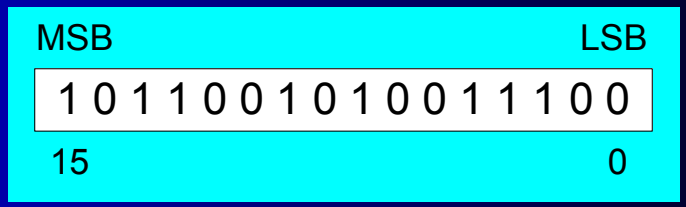
Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Próxima seção

- **Boas vindas à linguagem Assembly**
- Conceito de máquina virtual
- Representação dos dados
- Operações booleanas

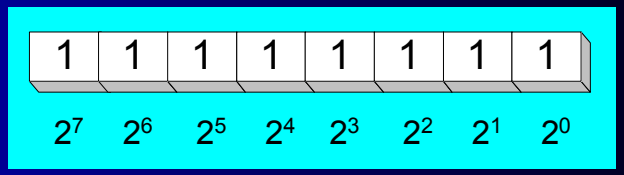
Números binários

- Dígitos são 1 e 0
  - 1 = verdadeiro
  - 0 = falso
- MSB – bit mais significativo
- LSB – bit menos significativo
- Numeração de Bits:



Números binários

- Cada dígito (bit) é 1 ou 0
- cada bit representa uma potência de 2:



Cada número binário é uma soma de potências de 2

Table 1-3 Binary Bit Position Values.

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Traduzindo binário para decimal

A notação posicional mostra como calcular o valor decimal de número binário:

$$dec = (D_{n-1} \times 2^{n-1}) + (D_{n-2} \times 2^{n-2}) + \ldots + (D_1 \times 2^1) + (D_0 \times 2^0)$$

D = dígito binário

binário 00001001 = decimal 9:

$$(1 \times 2^3) + (1 \times 2^0) = 9$$

Tradução de decimal sem sinal para binário

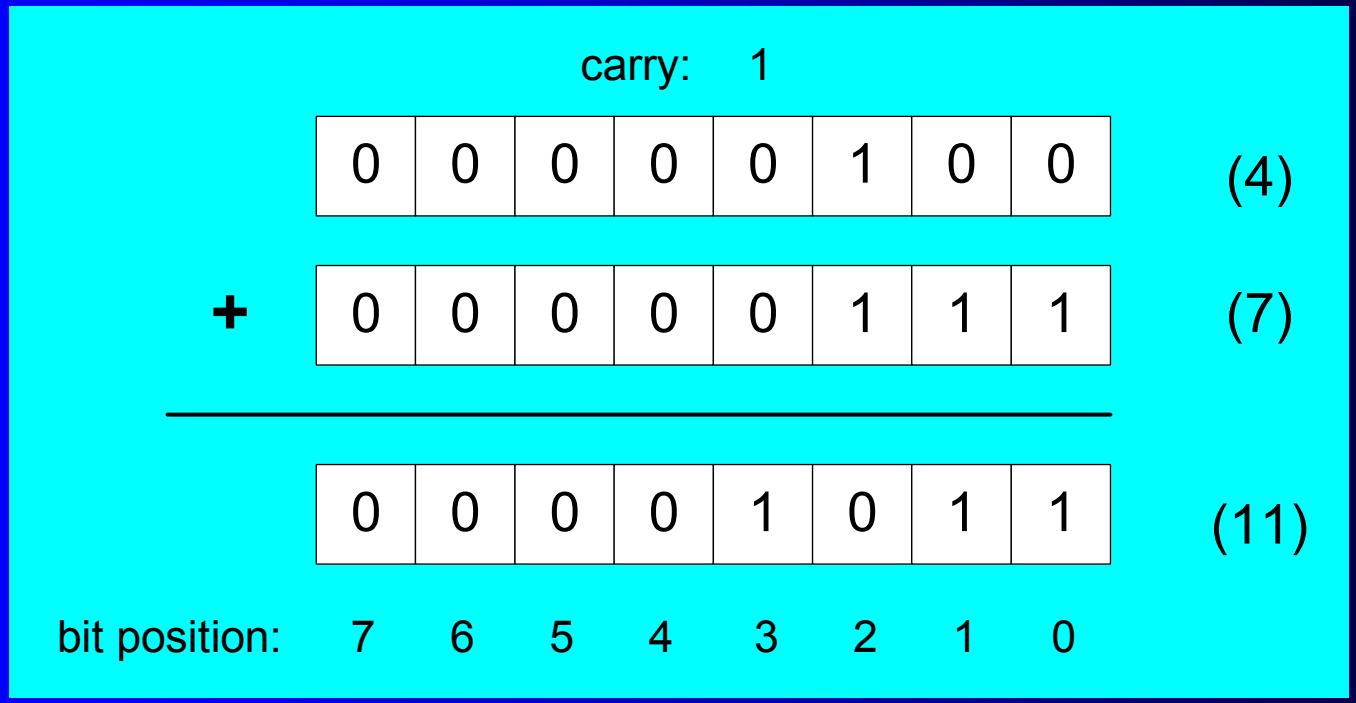
- Dividir repetidamente o inteiro decimal por 2. Cada resto é um dígito binário no valor traduzido:

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

37 = 100101

Adição binária

- Começando com o LSB, somar cada par de dígitos, incluindo o vai-um (carry) se tiver.



Tamanhos de armazenamento de inteiros

Tamanhos padrões:

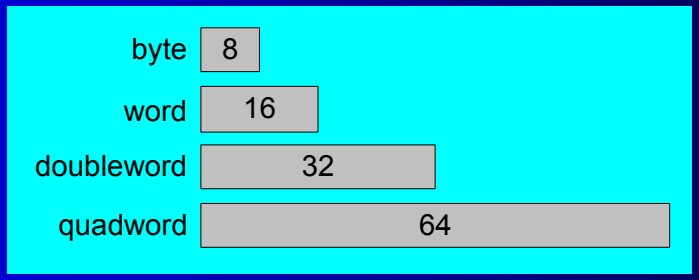


Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

Qual é o maior inteiro sem sinal que pode ser armazenado em 20 bits?

Inteiros hexadecimais

Valores binários são representados em hexadecimal.

Table 1-5 Binary, Decimal, and Hexadecimal Equivalents.

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

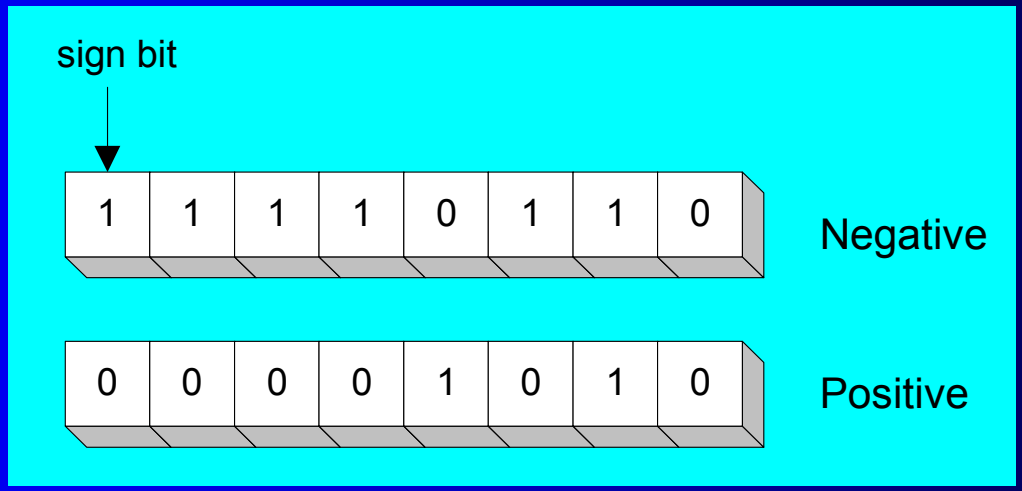
Traduzindo binário para hexadecimal

- Cada dígito hexadecimal corresponde a 4 bits.
- Exemplo: Traduzir o inteiro binário 000101101010011110010100 para hexadecimal:

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

Inteiros com sinal

O bit mais à esquerda indica o sinal. 1 = negativo, 0 = positivo



Se o dígito mais à esquerda de um inteiro hexadecimal é > 7, o valor é negativo. Exemplos: 8A, C5, A2, 9D

Notação em Complemento de 2

- Números negativos são armazenados em notação complemento de 2

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

Note que 00000001 + 11111111 = 00000000

Subtração Binária

- Quando se subtrai A – B, converte B ao seu complemento de 2
- Add A a (–B)

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

Prática: Subtrair 0101 de 1001.

Intervalos de inteiros com sinal

O bit mais à esquerda é reservado para o sinal. Isso limita o intervalo:

Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^7 to $(2^7 - 1)$
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Prática: Qual é o maior valor positivo que pode ser armazenado em 20 bits?

Armazenamento de caracteres

- Conjunto de caracteres
  - Standard ASCII (0 – 127)
  - Extended ASCII (0 – 255)
  - ANSI (0 – 255)
  - Unicode (0 – 65,535)
- Cadeia terminada por zero
  - Arranjo de caracteres seguido por um byte nulo

Representação de dados numéricos

- Binário puro
  - Pode ser calculado diretamente
- Binário em ASCII
  - Cadeia de dígitos: "01010101"
- Decimal em ASCII
  - Cadeia de dígitos: "65"
- Hexadecimal em ASCII
  - Cadeia de dígitos: "9C"

next: Boolean Operations

Seção seguinte

- **Boas vindas à linguagem Assembly**
- Conceito de máquina virtual
- Representação dos dados
- Operações booleanas

Operações Booleanas

- NOT
- AND
- OR
- Precedência de operadores
- Tabela verdade

Álgebra Booleana

- Baseada na lógica simbólica, projetada por George Boole
- Expressões booleanas criadas de:
  - NOT, AND, OR

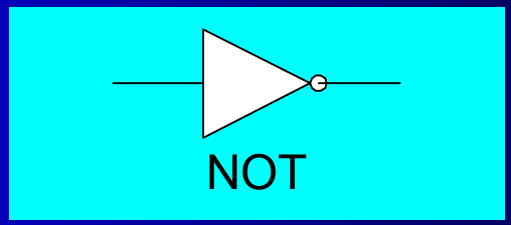
Expression	Description
¬X	NOT X
X ∧ Y	X AND Y
X ∨ Y	X OR Y
¬X ∨ Y	(NOT X) OR Y
¬(X ∧ Y)	NOT (X AND Y)
X ∧ ¬Y	X AND (NOT Y)

NOT

- Inverte um valor booleano
- Tabela verdade para o operador booleano NOT :

X	¬X
F	T
T	F

Digital gate diagram for NOT:

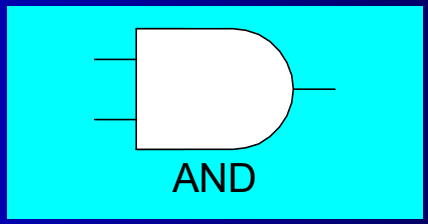


AND

- Tabela verdade para o operador booleano AND:

X	Y	X ∧ Y
F	F	F
F	T	F
T	F	F
T	T	T

Digital gate diagram for AND:

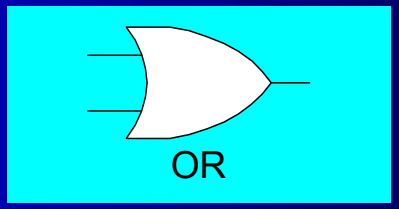


OR

- Tabela verdade para o operador booleano OR:

X	Y	X ∨ Y
F	F	F
F	T	T
T	F	T
T	T	T

Digital gate diagram for OR:



Precedência de operadores

- Exemplos mostrando a ordem das operações:

Expression	Order of Operations
$\neg X \vee Y$	NOT, then OR
$\neg (X \vee Y)$	OR, then NOT
$X \vee (Y \wedge Z)$	AND, then OR

Tabela verdade (1 de 3)

- Uma função booleana tem uma ou mais entradas booleanas e retorna uma única saída booleana.
- Uma tabela verdade mostra todas as entradas e saídas de uma função booleana

Exemplo: $\neg X \vee Y$

X	¬X	Y	¬X ∨ Y
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

Tabela verdade (2 de 3)

- Exemplo: $X \wedge \neg Y$

X	Y	¬Y	X ∧ ¬Y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

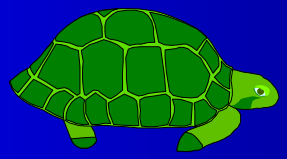
Tabela verdade (3 de 3)

- Exemplo: $(Y \wedge S) \vee (X \wedge \neg S)$

X	Y	S	$Y \wedge S$	$\neg S$	$X \wedge \neg S$	$(Y \wedge S) \vee (X \wedge \neg S)$
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T

Resumo

- Linguagem Assembly ajuda a entender como o software é construído nos níveis mais baixos
- Linguagem Assembly tem relação um-a-um com a linguagem de máquina
- Cada camada na arquitetura de computadores é uma abstração de uma máquina
  - Camada pode ser hardware ou software
- Expressões booleanas são essenciais para o projeto do hardware e software de computadores



54 68 65 20 45 6E 64

O que esses números representam?