

# Lista 1

$$111 = Y \quad 000000 = X (2)$$

1) Decimal  $\rightarrow$  Binário:

a)  $25_{10} = 25/2 \ 1 \ 12/2 \ 0 \ 6/2 \ 0 \ 3/2 \ 1 \ 1/2 \ 1$  inverte =  $11001_2$

b)  $26_{10} = 26/2 \ 0 \ 13/2 \ 1 \ 6/2 \ 0 \ 3/2 \ 1 \ 1/2 \ 1$  inverte =  $11010_2$

c)  $239_{10} = 239/2 \ 0 \ 119/2 \ 1 \ 58/2 \ 0 \ 29/2 \ 1 \ 14/2 \ 0 \ 7/2 \ 1 \ 3/2 \ 1 \ 1/2 \ 1 = 111001010_2$

d)  $622_{10} = 622/2 \ 0 \ 311/2 \ 1 \ 155/2 \ 1 \ 77/2 \ 1 \ 38/2 \ 0 \ 19/2 \ 1 \ 9/2 \ 1 \ 5/2 \ 0 \ 2/2 \ 0 \ 1/2 \ 1 = 1001101110_2$

2) Binário  $\rightarrow$  Decimal:

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

b)  $1001_2 = (1 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0) = 8 + 1 = 9_{10}$

c)  $10010_2 = (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (0 \cdot 2^0) = 16 + 2 = 18_{10}$

d)  $110111_2 = (1 \cdot 2^4) + (1 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) = 16 + 8 + 2 + 1 = 27_{10}$

4) Octal  $\rightarrow$  Binário:

a)  $45_8 = 100 \ 101 = 100101_2$

b)  $712_8 = 111 \ 001 \ 010 = 111001010_2$

c)  $10_8 = 001 \ 000 = 001000_2$

d)  $7402_8 = 111 \ 100 \ 000 \ 010 = 111100000010_2$

5) Octal  $\rightarrow$  Decimal:

a)  $32_8 = 011010_2 = 2^4 + 2^3 + 2^4 = 2 + 8 + 16 = 26_{10}$

b)  $238_8 = 010 \ 011 \ 001_2 = 2^6 + 2^3 + 2^4 + 2^1 = 1 + 8 + 16 + 128 = 153_{10}$

c)  $17_8 = 001 \ 111_2 = 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15_{10}$

d)  $275_8 = 010 \ 111 \ 101_2 = 2^6 + 2^3 + 2^4 + 2^5 + 2^7 = 1 + 4 + 8 + 16 + 32 + 128 = 189_{10}$

7) Hexadecimal  $\rightarrow$  Binário:

a)  $1F_{16} = 0001 \ 1111 = 00011111_2$

b)  $23_{16} = 0010 \ 0000 \ 00FF = 0010000000111111_2$

c)  $1AB_{16} = 0001 \ 1010 \ 1011 = 000110101011_2$

d)  $94D_{16} = 1001 \ 0100 \ 1101 = 100101001101_2$

0-0000 6-0110 A-1010 E-

1-0001 7-0111 B-1011 F110

2-0010 8-1000 C-1100 F-

3-0011 9-1001 D-1101 F111

4-0100 Paula Drunca

5-0101

### 3) Conversão Dec-Bin

Como os sistemas decimal e binário não têm bases de mesma potência, não podemos fazer a conversão baseada em uma tabela, como acontece nos sistemas hexadecimal e octal. Aqui será adotada a Fórmula dos Sistemas de Numeração Posicionais. Esta fórmula indica que um valor é sempre obtido através do somatório de multiplicações dos valores dos dígitos pela sua base elevada à posição do dígito.

Fórmula dos Sistemas Posicionais: A fórmula geral para a conversão de um número de um sistema posicional é:

$$\text{Número} = \sum (d_i \times b^i)$$

onde 'di' é o dígito na posição i e  $b^i$  é a base do sistema numérico.

Conversão de Binário para Decimal:

Para converter um número binário para decimal, são usadas multiplicações sucessivas. Cada valor de dígito é multiplicado por 2 elevado à potência de sua posição, e os resultados de todas as multiplicações são somados. O dígito menos significativo tem índice zero.

Por exemplo, para converter o número binário 1101 para decimal:

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ | \quad | \quad | \quad | \\ | \quad | \quad \swarrow \quad 1 * 2^0 = 1 \\ | \quad | \quad \swarrow \quad 0 * 2^1 = 0 \\ \swarrow \quad \quad \quad 1 * 2^2 = 4 \\ \swarrow \quad \quad \quad 1 * 2^3 = 8 \end{array}$$

Somando os resultados das multiplicações:

$$8 + 4 + 0 + 1 = 13$$

Portanto, 1101 em binário é igual a 13 em decimal.

Conversão de Decimal para Binário:

Para converter um número decimal para binário, são usadas divisões sucessivas por 2. O resultado de cada divisão é novamente dividido por 2, e a divisão é inteira. Os restos serão sempre 0 ou 1. O resultado final será a união dos restos, em ordem inversa (a primeira divisão gera como resto o bit menos significativo da resposta).

Por exemplo, para converter o número decimal 13 para binário:

$$13 \div 2 = 6, \text{ resto } 1$$

$$6 \div 2 = 3, \text{ resto } 0$$

$$3 \div 2 = 1, \text{ resto } 1$$

$$1 \div 2 = 0, \text{ resto } 1$$

Os restos em ordem inversa formam o número binário:

110111011101

Portanto, 13 em decimal é igual a 1101 em binário.

Fórmula dos Sistemas Posicionais:

$$\text{Número} = \sum (d_i \times b^i)$$

Exemplo de Conversão Bin2Dec

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \\ \times 2^3 \ 2^2 \ 2^1 \ 2^0 \\ = 8 + 4 + 0 + 1 \\ = 13 \ (\text{decimal}) \end{array}$$

Exemplo de Conversão Dec2Bin:

$$13 \div 2 = 6, \text{ resto } 1$$

$$6 \div 2 = 3, \text{ resto } 0$$

$$3 \div 2 = 1, \text{ resto } 1$$

$$1 \div 2 = 0, \text{ resto } 1$$

Binário: 1101

## 6) Conversão Octal-Bin

Como os sistemas Octal (de base 8) e Binário (de base 2) têm bases de mesma potência, pode-se fazer os processos de conversão de forma simplificada. Para isso, basta montar uma tabela de conversão bem simples. Essa tabela é construída atribuindo a cada valor octal (valores de 0 a 7) um código binário equivalente. Como são necessários 8 códigos, precisaremos de 3 bits binários neste código (pois com 3 bits pode-se gerar  $2^3=8$  códigos). Os códigos são gerados conforme uma contagem simples binária.

Tabela de Conversão entre Octal e Binário

Octal	Binário
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

### Conversão de Octal para Binário

Para converter um número octal para binário, basta substituir cada dígito octal pelo seu código binário equivalente conforme a tabela.

Por exemplo, para converter o número octal 725 para binário:

7	2	5
111	010	101

Assim, 725 em octal é igual a 111010101 em binário.

### Conversão de Binário para Octal

Para converter um número binário para octal, agrupa-se os dígitos binários em grupos de 3, começando pela parte menos significativa. Se necessário, pode-se inserir zeros à esquerda (na parte mais significativa) para completar o último grupo de 3 bits. Em seguida, procura-se cada grupo de 3 bits na tabela de conversão e substitui-se pelo equivalente octal.

Por exemplo, para converter o número binário 111010101 para octal:

111	010	101
7	2	5

Assim, 111010101 em binário é igual a 725 em octal.

Tabela de Conversão entre Octal e Binário

### Octal Binário

0 000
1 001
2 010
3 011
4 100
5 101
6 110
7 111

Octal: 7 2 5

Binário: 111 010 101

Exemplo de Conversão Bin2Oct:

Binário: 111 010 101

Octal: 7 2 5

## 8) Conversão Hexad-Bin

Como os sistemas Hexadecimal (de base 16) e Binário (de base 2) têm bases de mesma potência, pode-se fazer os processos de conversão de forma simplificada. Para isso, basta montar uma tabela de conversão. Essa tabela é construída atribuindo a cada valor hexadecimal (valores de 0 a F) um código binário equivalente. Como são necessários 16 códigos, são necessários 4 bits binários neste código (pois com 4 bits pode-se gerar  $2^4=16$  códigos). Os códigos são gerados conforme uma contagem simples binária.

Tabela de Conversão entre Hexadecimal e Binário

Hexadecimal Binário

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Para converter um número hexadecimal para binário, basta substituir cada dígito hexadecimal pelo seu código binário equivalente de 4 bits conforme a tabela.

Por exemplo, para converter o número hexadecimal 2F3 para binário:

2            F            3  
└─┘   └─┘   └─┘  
0010        1111        0011

Assim, 2F3 em hexadecimal é igual a 001011110011 em binário.

Conversão de Binário para Hexadecimal:

Para converter um número binário para hexadecimal, agrupa-se os dígitos binários em grupos de 4, começando pela parte menos significativa. Se necessário, pode-se inserir zeros à esquerda (na parte mais significativa) para completar o último grupo de 4 bits. Em seguida, procura-se cada grupo de 4 bits na tabela de conversão e substitui-se pelo equivalente hexadecimal.

Por exemplo, para converter o número binário 1011110011 para hexadecimal:

1011        1100        0011  
└─┘   └─┘   └─┘  
B            C            3

Assim, 1011110011 em binário é igual a BC3 em hexadecimal.

Exemplo de Conversão Hex2Bin

Hexadecimal: 2 F 3

Binário: 0010 1111 0011

Exemplo de Conversão Bin2Hex

Binário: 1011 1100 0011

Hexadecimal: B C 3

$$9) X = 100100 \quad Y = 101011$$

$$a) Soma \quad X + Y: \quad 100100$$

$$+ 101011$$

$$\hline 1001111$$

$$b) Subtração \quad Y - X: \quad 101011$$

$$- 100100$$

$$\hline 0010011$$

$$000111$$

$$c) Multiplicação \quad X \cdot Y: \quad 101011$$

$$\times 100100$$

$$\hline 000000$$

$$000000\#$$

$$\downarrow 010111$$

$$000000\#$$

$$\downarrow 000000\#$$

$$\downarrow 000000\#$$

$$\downarrow 1010111$$

$$\downarrow 1100001100$$

$$10) X = 111 \quad Y = 1001111$$

Soma

$$a) X + Y: \quad 1001111$$

$$+ 111$$

$$\hline 1010110$$

$$b) Subtração \quad Y - X: \quad 1001111$$

$$- 111$$

$$\hline 000000$$

Multiplicação

$$c) X \cdot Y: \quad 111$$

$$\times 111$$

$$\hline 1001111$$

$$1001111\#$$

$$1001111\#$$

$$1000101001$$

$$11) X = 1010 \quad Y = 1001011$$

11  
S. otei

a) Soma  $X + Y$ :  $1001011$

$$\begin{array}{r} 1010 \\ + 1001011 \\ \hline 1010101 \end{array}$$

b) Subtração  $Y - X$ :  $1001011$

$$\begin{array}{r} 1001011 \\ - 1010 \\ \hline 1000001 \end{array}$$

c) Multiplicação  $X \cdot Y$ :  $1001011$

$$\begin{array}{r} 1010 \\ \times 1001011 \\ \hline 0000000 \end{array}$$

$$\begin{array}{r} 1001011 \\ \times 1010 \\ \hline 0000000 \end{array}$$

$$1001011\#*$$

mas, b=0 no resultado da multiplicação

$(1-0)(1-0) + (1-0)(1-0) = 0+0 = 0$

$(1-0)(1-0) + (1-0)(1-0) = 0+0 = 0$

## Lista 2)

1) Decimal  $\rightarrow$  Binário com Sinal, C-1 e C-2:

	C-1	+	C-2
a) $+24_{10}$	$24_{10} = 0\ 11000$		$011000$
b) $-24_{10}$	$inv(011000) + 1 = 100111 + 1$		$100111$
c) $+31_{10}$	$31_{10} = 011111$		$011111$
d) $-31_{10}$	$inv(011111) + 1 = 100001$		$100001$
e) $-57_{10}$	$57_{10} = 0111001$		$0111001$
	$\rightarrow inv(0111001) + 1 = 101111$		

2) Quais os intervalos de valores que podem ser representados em C-2, com:

a) 5 bits:  $-(2^{5-1})_a + (2^{5-1}-1) = -16_a + 15$

b) 7 bits:  $-(2^{7-1})_a + (2^{7-1}-1) = -64_a + 63$

c) 9 bits:  $-(2^{9-1})_a + (2^{9-1}-1) = -256_a + 255$

d) 11 bits:  $-(2^{11-1})_a + (2^{11-1}-1) = -1024_a + 1023$

e) 13 bits:  $-(2^{13-1})_a + (2^{13-1}-1) = -4096_a + 4095$

3) Decimal  $\rightarrow$  Binário com C-2 com 16 bits:

a)  $5_{10} = 5/2 \ 1 \ 2/2 \ 0 \ 4/2 \ 1 = 0000000000001001$

$-5_{10} = inv(0000000000001001) + 1 = 1111111111111011$

b)  $11_{10} = 11/2 \ 1 \ 5/2 \ 1 \ 2/2 \ 0 \ 4/2 \ 1 = 00000000000010011$

$-11_{10} = inv(00000000000010011) + 1 = 11111111111110101$

c)  $23_{10} = 23/2 \ 1 \ 11/2 \ 1 \ 5/2 \ 1 \ 2/2 \ 0 \ 4/2 \ 1 = 000000000000100111$

$-23_{10} = inv(000000000000100111) + 1 = 111111111111101001$

d)  $38_{10} = 38/2 \ 0 \ 19/2 \ 1 \ 9/2 \ 1 \ 4/2 \ 0 \ 2/2 \ 0 \ 4/2 \ 1 = 0000000000100110$

$-38_{10} = inv(0000000000100110) + 1 = 111111111111011010$

e)  $523_{10} = 123/2 \ 1 \ 63/2 \ 1 \ 30/2 \ 0 \ 15/2 \ 1 \ 7/2 \ 1 \ 3/2 \ 1 \ 1/2 \ 1 = 0000000001111011$

$-523_{10} = inv(0000000001111011) + 1 = 1111111110001011$

## 4) Soma em C-2

A representação de valores em complemento-2 é a estratégia utilizada hoje para representar os números inteiros com sinal nos computadores. Aqui será apresentado como são realizadas somas de valores em complemento-2.

Para realizar uma soma de valores em complemento-2, os números devem ser alinhados pelo dígito menos significativo, e a soma é realizada bit a bit, observando os carrys gerados durante o processo. Primeiramente, alinhe os números a serem somados pelo dígito menos significativo. Em seguida, realize a soma bit a bit da direita para a esquerda, propagando os carrys gerados durante a soma. O último carry gerado (seja 1 ou 0) não é considerado na parte mais significativa do resultado final.

Vamos considerar um exemplo de soma sem overflow, somando os números 5 e -3 em complemento-2 utilizando 4 bits. A representação binária de 5 em complemento-2 é 0101, e de -3 em complemento-2 é 1101. Somando esses valores temos:

$$\begin{array}{r} 0101 \\ + 1101 \\ \hline 0010 \end{array}$$

O resultado é 0010, que corresponde ao valor decimal 2. Neste caso, o último carry gerado (1) não é considerado no resultado final.

Somas de valores com sinais opostos nunca dão erro porque o resultado sempre estará no intervalo entre os dois números somados, e, portanto, sempre estará no intervalo de representação. No entanto, ao somar valores com o mesmo sinal, pode ocorrer um erro se os valores somados resultarem em um valor maior que o maior valor que pode ser representado no intervalo.

A detecção de overflow é realizada em todas as contas para identificar erros. O computador detecta a ocorrência de erro através de uma porta XOR realizada entre o Carry IN e o Carry OUT do bit mais significativo (bit de sinal, o último bit somado). Se os carrys forem diferentes, ocorreu um erro (overflow).

Considerando um exemplo de soma com overflow, vamos somar os números 7 e 5 em complemento-2 utilizando 4 bits. A representação binária de 7 em complemento-2 é 0111, e de 5 em complemento-2 é 0101. Somando esses valores temos:

$$\begin{array}{r} 0111 \\ + 0101 \\ \hline 1100 \end{array}$$

O Carry IN do bit mais significativo é 1 e o Carry OUT é 0. Como os carrys são diferentes, houve overflow. O resultado binário 1100 corresponde ao valor decimal -4, indicando claramente que ocorreu um erro, pois estamos somando dois números positivos.

Em outro exemplo, vamos somar os números -3 e -2 em complemento-2 utilizando 4 bits. A representação binária de -3 em complemento-2 é 1101, e de -2 em complemento-2 é 1110. Somando esses valores temos:

$$\begin{array}{r} 1101 \\ + 1110 \\ \hline 1011 \end{array}$$

O Carry IN do bit mais significativo é 1 e o Carry OUT também é 1. Como os carrys são iguais, não houve overflow. O resultado binário 1011 corresponde ao valor decimal -5, que está correto.

Em resumo, o overflow é detectado através da comparação dos carrys utilizando uma porta XOR. Quando ocorre overflow, geralmente o resultado terá o sinal oposto dos valores somados. No entanto, a detecção de overflow pelo computador é feita apenas através da porta XOR por ser um método simples e eficiente.

Soma de Valores com Sinais Opostos:

0101 (5)

+ 1101 (-3)

-----

0010 (2)

Soma sem Overflow:

1101 (-3)

+ 1110 (-2)

-----

1011 (-5)

Soma com Overflow:

0111 (7)

+ 0101 (5)

-----

1100 (-4)

## 5) Subtração em C-2

A representação de valores em complemento-2 é a estratégia utilizada hoje para representar os números inteiros com sinal nos computadores. Aqui será apresentado como são realizadas subtrações de valores em complemento-2.

Para realizar uma subtração de valores em complemento-2, deve-se inicialmente realizar o complemento-2 do segundo valor antes de colocar os números alinhados pelo dígito menos significativo e realizar uma soma normal bit a bit. Ou seja, ao invés de fazermos  $A-B$ , faremos  $A+(-B)$ . Essa é a vantagem do complemento-2, poder fazer somas e subtrações apenas usando um somador, sem se preocupar com os sinais.

Vamos considerar um exemplo de subtração sem overflow, subtraindo os números 7 e 3 em complemento-2 utilizando 4 bits. Primeiramente, a representação binária de 7 em complemento-2 é 0111, e de 3 em complemento-2 é 0011. Para subtrair 3 de 7, devemos primeiro encontrar o complemento-2 de 3, que é 1101. Agora, somamos 7 e -3:

0111

+ 1101

-----

0100

O resultado é 0100, que corresponde ao valor decimal 4. Neste caso, o último carry gerado (1) não é considerado no resultado final, mas será usado para a detecção de erros.

Os possíveis casos para subtração são os opostos da soma, pois o segundo valor terá seu sinal invertido. Subtrações de valores com sinais iguais, como a explicada acima, nunca darão erro porque esse caso levará a uma soma de valores com sinais opostos, e, portanto, sempre o resultado estará no intervalo de representação. No entanto, ao subtrair valores com sinais opostos, pode ocorrer um erro se os valores subtraídos resultarem em um valor maior que o maior valor que pode ser representado no intervalo.

A detecção de overflow é realizada em todas as contas para identificar erros. O computador detecta a ocorrência de erro através de uma porta XOR realizada entre o Carry IN e o Carry OUT do bit de sinal (o último bit somado). Se os carrys forem diferentes, ocorreu um erro (overflow).

Vamos considerar um exemplo de subtração com overflow, subtraindo 3 de -5 em complemento-2 utilizando 4 bits. A representação binária de -5 em complemento-2 é 1011, e de 3 em complemento-2 é 0011. Para subtrair 3 de -5, devemos primeiro encontrar o complemento-2 de 3, que é 1101. Agora, somamos -5 e -3:

$$\begin{array}{r} 1011 \\ + 1101 \\ \hline 1000 \end{array}$$

O Carry IN do bit de sinal é 1 e o Carry OUT é 0. Como os carrys são diferentes, houve overflow. O resultado binário 1000 corresponde ao valor decimal -8, indicando claramente que ocorreu um erro, pois estamos subtraindo um número positivo de um negativo.

Em outro exemplo, vamos subtrair -2 de -4 em complemento-2 utilizando 4 bits. A representação binária de -4 em complemento-2 é 1100, e de -2 em complemento-2 é 1110. Para subtrair -2 de -4, devemos primeiro encontrar o complemento-2 de -2, que é 0010. Agora, somamos -4 e 2:

$$\begin{array}{r} 1100 \\ + 0010 \\ \hline 1110 \end{array}$$

O Carry IN do bit de sinal é 1 e o Carry OUT também é 1. Como os carrys são iguais, não houve overflow. O resultado binário 1110 corresponde ao valor decimal -2, que está correto.

Em resumo, o overflow é detectado através da comparação dos carrys utilizando uma porta XOR. Quando ocorre overflow, geralmente o resultado terá o sinal oposto dos valores subtraídos. No entanto, a detecção de overflow pelo computador é feita apenas através da porta XOR por ser um método simples e eficiente.

Subtração de Valores com Sinais Iguais:

$$\begin{array}{r} 0111 (7) \\ + 1101 (-3) \\ \hline 0100 (4) \end{array}$$

Subtração sem Overflow:

$$\begin{array}{r} 1100 (-4) \\ + 0010 (2) \\ \hline 1110 (-2) \end{array}$$

Subtração com Overflow:

$$\begin{array}{r} 1011 (-5) \\ + 1101 (-3) \\ \hline 1000 (-8) \end{array}$$

6)  $x = 0110$   $y = 0011$ , 4 bits, C-2:

a)  $x + y$  causa overflow?

$$\begin{array}{r} 0110 \\ + 0011 \\ \hline 1001 \end{array}$$

Sim,  $C_{in} \neq C_{out}$

b)  $x - y$  causa overflow?

$$\begin{array}{r} 0110 - 0011 \\ \downarrow \\ \text{inv}(0011) + 1 \\ 1100 + 1 \\ 1101 \end{array}$$
$$\begin{array}{r} 0110 \\ + 1101 \\ \hline 0011 \end{array}$$

Não,  $C_{in} = C_{out}$

7)  $x = 0111$   $y = 1101$ , 4 bits, C-2:

a)  $x + y$  causa overflow?

$$\begin{array}{r} 0111 \\ + 1101 \\ \hline 0100 \end{array}$$

Não,  $C_{in} = C_{out}$

b)  $x - y$  causa overflow?

$$\begin{array}{r} 0111 - 1101 \\ \downarrow \\ \text{inv}(1101) + 1 \\ 0010 + 1 \\ 0011 \end{array}$$
$$\begin{array}{r} 0111 \\ + 0011 \\ \hline 1010 \end{array}$$

Sim,  $C_{in} \neq C_{out}$

8)  $x = 10010111$   $y = 11000001$ , 8 bits, C-2:

a)  $x + y$  causa overflow?

$$\begin{array}{r} 10010111 \\ + 11000001 \\ \hline 01011000 \end{array}$$

Sim,  $C_{in} \neq C_{out}$

b)  $x - y$  causa overflow?

$$\begin{array}{r} 10010111 - 11000001 \\ \downarrow \\ \text{inv}(11000001) + 1 \\ 00111101 + 1 \\ 00111110 \end{array}$$

Não,  $C_{in} = C_{out}$

$$\begin{array}{r} 10010111 \\ + 00111110 \\ \hline 11010001 \end{array}$$