Unidade III

Sistemas Numéricos e o Computador

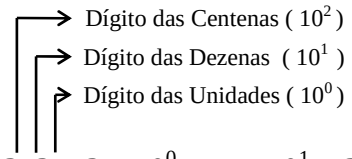
III .1 - O Sistema Decimal

- Base: 10

- Dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

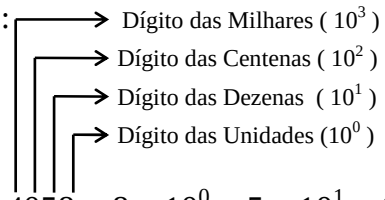
- A contribuição de um dígito num número decimal, depende da posição que ele ocupa.

Exemplo 1:



$373 = 3 \times 10^0 + 7 \times 10^1 + 3 \times 10^2$

Exemplo 2:



$4058 = 8 \times 10^0 + 5 \times 10^1 + 0 \times 10^2 + 4 \times 10^3$

Exemplo 3:

$0{,}325 = 3 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$

$10^{-1}\ 10^{-2}\ 10^{-3}$

Propriedades de um número decimal:

1. São usados os dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

2. Os dígitos em um número inteiro do sistema decimal são coeficientes de potências de base 10 cujos expoentes, começando por 0, crescem de 1 em 1, da direita para a esquerda.

3. Os dígitos da parte fracionária (direita da vírgula) de um número do sistema decimal são coeficientes de potências de base 10 cujos expoentes começando de -1, decrescem de -1 em -1, da esquerda para a direita.

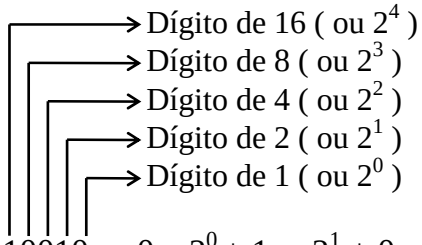
III.2 - O Sistema Binário

- Base: 2

- Dígitos: 0 e 1

- A contribuição de um dígito num número binário depende da posição que ele ocupa.

Exemplo 1: Considere o número binário 10010



$10010_2 = 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4$

$= 0 + 2 + 0 + 0 + 16 = 18_{10}$

Exemplo 2: $1\ 1\ 0\ .\ 1\ 1\ 0\ 0\ 1_2$

$\downarrow\ \downarrow\ \downarrow\quad \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow$

$2^2\ 2^1\ 2^0\quad 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-5}$

$110.11001_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1x2^{-1} + 1x2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$

$= 0 + 2 + 4 + 0{,}5 + 0{,}25 + 0 + 0 + 0{,}03125$

$= 6 + 0{,}78125$

$= 6{,}78125_{10}$

Qual o maior número binário inteiro que pode ser escrito com quatro dígitos?
Observe a tabela a seguir:

Decimal	Binário	Decimal	Binário
0	0	9	1001
1	1	10	1010
2	10	11	1011
3	11	12	1100
4	100	13	1101
5	101	14	1110
6	110	15	1111
7	111	16	10000
8	1000	17	10001

Podemos deduzir:

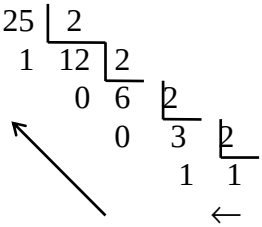
número de bits		maior decimal
1	$\rightarrow$	$1 = 2^1 - 1$
2	$\rightarrow$	$3 = 2^2 - 1$
3	$\rightarrow$	$7 = 2^3 - 1$

Ou seja, o maior número decimal (inteiro) que pode ser representado com **n** bits (dígitos binários) é $2^{\mathbf{n}} - 1$.

III.2.1 - Convertendo inteiro do sistema decimal para binário:

- Divide-se o número decimal dado e os quocientes sucessivos por 2 até que o quociente dê 1. O binário equivalente é a combinação do último quociente (1) com todos os restos, tomados de baixo para cima.

Exemplo 1: Converter o decimal 25 em binário.



ou seja, $25_{10} = 11001_2$

III.2.2 - Convertendo um número fracionário do sistema decimal para binário fracionário.

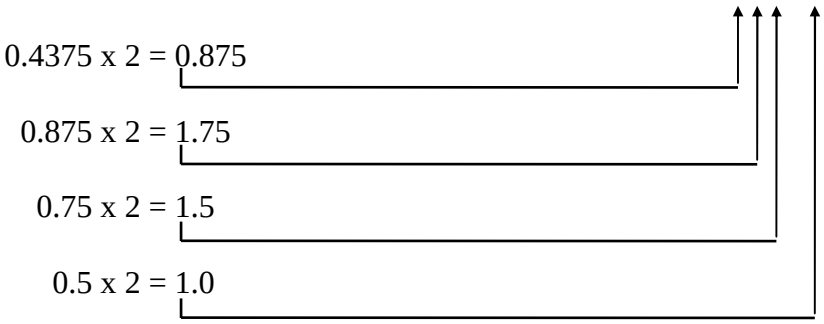
É necessário decompor o número em sua parte inteira e sua parte fracionária. Assim, 102.247 seria decomposto em 102 e 0.247 e a representação de cada parte achada. Agora as duas partes são adicionadas. Já sabemos como transformar um número decimal inteiro em binário inteiro.

Por outro lado, para se transformar um número decimal fracionário (menor que 1) em número binário, usamos o método que consiste em "dobrar" repetidamente a fração decimal. Se aparecer um "1" na parte inteira, esse "1"é acrescentado à direita da fração binária que está sendo formada e é eliminado da parte inteira. Se depois de uma multiplicação por 2 permanecer um "0" na parte inteira, esse "0" é acrescentado à fração binária que está sendo formada.

Exemplo: $25.4375_{10} = ?_2$

25.4375 = 25 + 0.4375

$25_{10} = 11001_2$ 0.4375 = 0. 0 1 1 1



ou seja, $25.4375_{10}=11001.0111_2$

III.3 - Dígitos Significativos

Em qualquer número inteiro, o dígito mais à direita é dito "dígito menos significativo" (Least Significant Bit) ou dígito de mais baixa ordem.

O dígito não nulo mais à direita é dito "dígito de mais alta ordem" ou dígito mais significativo" (Most Significant Bit).

MSB↓ ↓LSB
Exemplo: número binário: →1101011 ←
número decimal: → 2374 ←
↓ ↓
Dígito mais Significativo
Dígito menos Significativo

Com relação a bytes: [00000000] 01011010 [00000011]
↓ ↓
MSBy LSBy

III. 4 - Sistema Hexadecimal

- Base: 16

- Digítos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E e F

- A contribuição de um digíto num número hexadecimal depende da posição que ele ocupa.

Exemplo: Considere $1A3_{16}$ (ou 1A3H como notação optativa)

16^2
16^1
16^0

$1A3 = 3 \times 16^0 + 10 \times 16^1 + 1 \times 16^2$
$= 3 + 160 + 256 = 419$

Obs.: Cada dígito hexadecimal pode ser representado por 4 dígitos binários.

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

III .4.1 - Convertendo Decimal em Hexadecimal

- Divide-se o número decimal dado e os quocientes sucessivos por 16 até obter-se um quociente menor do que 16. Combina-se o último quociente com todos os restos obtidos nas sucessivas divisões de baixo para cima formando o número hexadecimal desejado.

Exemplo: $282_{10} = ?_{16}$

```
282 | 16
 10  17 | 16        ou seja, 282₁₀=11A₁₆
(A)   1 |  1
```

III .4.2 - Transformando Binário Inteiro em Hexadecimal

- Basta grupar de 4 em 4 bits, da direita para a esquerda.

Exemplo: $\underline{110}\ \underline{1001}\ \underline{1101}_{\,2} = ?_{16} = 69D_{16}$
6 9 D

III.4.3 - Transformando Hexadecimal Inteiro em Binário

- A cada dígito hexadecimal é associado um grupo de 4 bits.

Exemplo: $74A_{16} = ?_2$

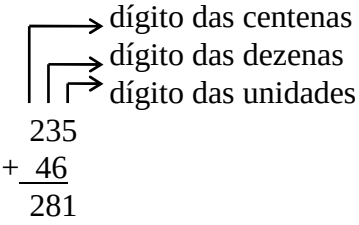
$\underline{7}\quad\underline{4}\quad\underline{A} \quad\Rightarrow\quad 74A_{16} = 11101001010_2$
0111 0100 1010

III.5 - Aritmética Binária

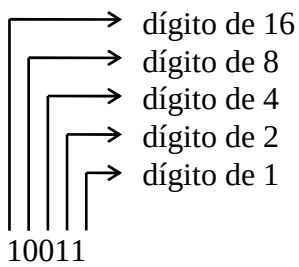
Obs.: Estaremos inicialmente operando binários puros. Numa segunda etapa, trataremos dos
binários sinalizados.

III. 5.1 - Adição Binária

Para efetuar uma adição decimal, alinhamos os dígitos dos dois números que devem ser adicionados. Por exemplo, 235 + 46 = 281:



Fazemos essencialmente a mesma coisa para a adição entre binários. Por exemplo, somemos 10011_2 a 1001_2



```
  + 1001
  11100
```

Adicionamos então, dois dígitos de mesma posição de cada vez, começando pelos dígitos de mais baixa ordem. Ao adicionarmos dois dígitos binários, existem quatro possibilidades:

```
                                    1←vai-um
  0        0        1        1
+ 0      + 1      + 0      + 1
  0        1        1        0
```

Ainda se define:

```
  1 ← vai-um
    1
    1
  +1
  1
```

Então :

```
   10011
+  1001
   11100
```

Exemplo 1: $2_{10} + 2_{10} = 4_{10}$

```
  10
+10
100
```

Exemplo 2: $7_{10} + 5_{10} = 12_{10}$

```
  111
+101
1100
```

III. 5.2 - Multiplicação de Binários

Exemplo: 10011 x 1011

x	0	1
0	0	0
1	0	1

```
   1011
x  1011
   1011
  1011
  0000
1011
-------
1111001
```

III . 5.3 - Subtração Binária

4 ← Minuendo → 205

-2	← Subtraendo →	- 121
2	← Diferença →	84

Possibilidades:

1 ← emprestado ao subtraendo

0	1	1	0
- 0	- 1	- 0	- 1
0	0	1	1

Exemplo 1: 4 - 2 = 2

```
100
- 10
  10
```

Exemplo 2 : 132 - 47 = 85

```
10000100
-  101111
 1010101
```

III .5.4 - Divisão Binária

A divisão binária pode ser realizada (calculada) usando-se os mesmos passos da divisão decimal.

Exemplo: 110111 / 101 = ?

```
110111 | 101
101    |-----
-----    1011
00111
  101
-----
 0101
  101
-----
 (000)
```

III .6 - Armazenamento de Números Binários Negativos

A grande maioria das CPU's não implementa subtração binária em seus circuitos lógicos. A subtração, nesse caso, é tratada como adição :

7 - 5 = (+7) + (-5) = +2

O problema, pois, é escolher uma forma padronizada para armazenar números negativos. A forma canônica "sinal-magnitude" (valor absoluto) não funciona.

Por exemplo, se queremos subtrair 5 de 7 , escrevemos:

+7	00000111	
-5	10000101	7 - 5 = +7 + (-5) = +2
+2	10001100	(-12)?

Uma alternativa consiste em usar a representação "complemento-de-dois". Nessa forma, a representação dos números positivos é equivalente à "sinal-magnitude". Já um número negativo é representado da seguinte maneira:

1. Configura-se em binário o seu simétrico (positivo)
2. Invertem-se seus bits (complemento-de-um)
3. Adiciona-se o número binário "1"

É importante salientar que o bit mais à esquerda ainda representa o sinal do número

Exemplo 1: Representar $+3_{10}$ em “complemento-de-dois” usando um byte de armazenamento.

Solução: 00000011_2

Exemplo 2: Representar -5 em “complemento-de-dois” usando um byte de armazenamento.

Solução:

+5	→	00000101
Complemento-de-um	→	11111010
	+	1
Complemento-de-dois (-5)	→	11111101

OBS. O “complemento-de-dois” de um número negativo dá o seu simétrico positivo.

A seguir apresentamos uma tabela contendo a representação em complemento-de-dois, em um byte, de números inteiros no intervalo [-128,+127]:

BINÁRIO	**DECIMAL**	**BINÁRIO**	**DECIMAL**
10000000	-128	00000000	0
10000001	-127	00000001	+1
10000010	-126	00000010	+2
10000011	-125	00000011	+3
.	.	.	.
.	.	.	.
.	.	.	.
11111110	-2	01111101	+125
11111111	-1	01111110	+126
		01111111	+127

III.7 - Aritmética Binária em “Complemento-de-dois”

Comecemos adicionando +4 a -3:

+3 →	00000011	+4 →	00000100
Comp.-de-dois →	11111100	-3 → +	11111101
	+1		————

──────── (1) 00000001

-3 → 11111101 ↑desprezado (carry externo)

Se ignorarmos o "carry" externo, o resultado $00000001_2 = 1_{10}$ é correto. Em "complemento-de-dois" o "carry" externo é sempre desprezado!

As CPU's normalmente dispõem de um registrador de STATUS que tem um bit (flag C) indicando o valor do "carry" em cada adição bit a bit. Existem algumas instruções que permitem a manipulação do flag "carry".

Registrador de STATUS → | | | V | | | C | | |

V → Overflow
C → Carry

Exemplo: Processar 3-5 em "complemento-de-dois" em 1 byte.
Solução: 3 - 5 = (+3) + (-5)

+5 → 00000011
Comp.-de-um → 11111011
+1
-5 → 11111011

+3 → 00000011
+(-5) → 11111011
-2 → 11111110

00000001
+ 1
00000010 → (+2)

Exemplo: Processar 64+ 65 em "complemento-de-dois" em um byte.
Solução:
+64 → 01000000
+65 → 01000001
-127 → 10000001

01111110
+ 1
01111111 → $\sum_{i=0}^{6} 2^i = 2^7 = 127$

Quando os números adicionados são, em valor absoluto, bastante grandes, há possibilidade de OVERFLOW (estouro de armazenamento). O registrador de STATUS da CPU reserva um bit (FLAG V) para indicar essa situação (V=1).

O indicador de OVERFLOW será posto em 1 nas seguintes situações:

1. Há um "carry" do bit 6 ao bit 7 e não há "carry" externo;
2. Não há "carry" do bit 6 ao bit 7 mas há um "carry" externo.

Isso indica que o bit 7, sinal do resultado, foi "acidentalmente" modificado.

Exemplos:

* positivo-positivo

+6 → 01000000 C=0
+8 → 01000001 V=0
────────
+14 → 10000001

* positivo-positivo (com overflow)

+127 → 01000000 C=0
+1 → 01000001 V=1
────────

OBS. O simétrico de -128, +128, não pode ser armazenado em 1 byte.

-128 → 10000000 (?) Verifique!

* positivo-negativo (resultado positivo)

+4 → 01000000 C=1
-2 → 11111110 V=0
———————
+2 → (1)00000010
↑desprezado

* positivo-negativo (resultado negativo)

+2 → 00000010 V=0
-4 → 11111100 C=0
———————
-2 → 11111110
resultado correto

* negativo-negativo

-2 → 11111110 C=1
-4 → 11111100 V=0
———————
-6 → (1) 11111010
↑desprezado

* negativo-negativo com overflow (underflow)

-127 → 10000000 C=1
-62 → 11000010 V=1
———————
-189 → (1) 01000010 (?) → Errado
↑desprezado

III.8 - Representação de Números de Ponto Flutuante

No Sistema Decimal, considere o número 0.000123

Observe que os três zeros à direita do ponto decimal não são significativos. A normalização de um número decimal consiste em eliminar os zeros não significativos.

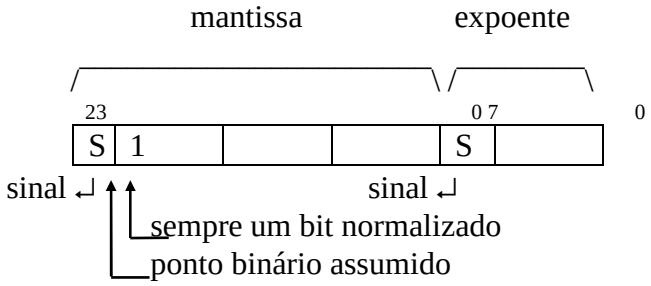
Assim, $0.000123 = 0.00123 \times 10^{-1} = 0.123 \times 10^{-3}$ ← forma normalizada

No Sistema Binário, também podemos normalizar números. Por exemplo:

$111.01 = 1.1101 \times 2^2 = \underline{0.11101 \times 2^3}$
forma normalizada

A regra a ser seguida é conservar o máximo de precisão possível. Conseguimos isto mantendo tantos bits significativos quanto possível. Como temos um número limitado de bits para mantissa, devemos eliminar os bits não significativos, mantendo os significativos. Para isso,

normalizamos a mantissa de modo que o primeiro bit "1" localize-se imediatamente à direita do ponto binário assumido, em frente ao primeiro bit da mantissa. O expoente, por outro lado, será armazenado aqui na forma de complemento-de-dois. Algumas arquiteturas armazenam o expoente na forma de “Excesso de 128”, que consiste em adicionar 128 ao verdadeiro expoente e armazenar o resultado.



Desta maneira, asseguramos precisão máxima, pois não existem zeros não significativos antes do primeiro bit "1" e minimizamos a porção truncada da mantissa no final.

Normalizar um número decimal fracionário y, é escrevê-lo na forma: $y = m \times 2^n$, onde $½ \leq |m| < 1$ e n é um expoente do tipo inteiro.

Por exemplo, considere y = 4.5

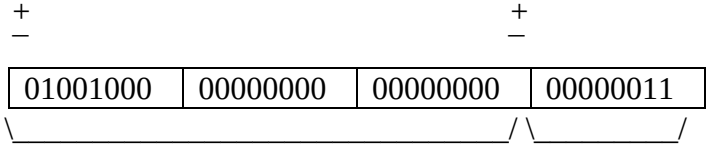
Temos :
4,5 | 2
0 2,25 | 2
0 1,125 | 2
0 0,5625

Ou seja, $4.5 = 0.5625 \times 2^3$, onde m = 0.5625 e n = 3

O valor de m é chamado "mantissa". Resta, agora, representar a mantissa e o expoente em binário:

$n = 3_{10} = 11_2$
$m = 0{,}5625 = 0{,}1001_2$
0,5625 x 2 = 1,125
0,125 x 2 = 0,25
0,25 x 2 = 0,5
0,5 x 2 = 1,0

$4{,}5_{10} = 0{,}1001 \times 2^3$



$0{,}5625_{10} = 0.1001_2$ $\qquad$ $3_{10} = 11_2$

Essa representação é dita "Precisão Simples". Normalmente se oferece uma outra opção chamada "Precisão Dupla" em que a mantissa é armazenada em 7 bytes (afora 1 byte do expoente).

Alguns processadores ainda oferecem a precisão expandida, em que a mantissa é armazenada em mais de 7 bytes.

III.9 - Representação BCD

Algumas arquiteturas de computadores oferecem adicionalmente, a opção de representação de números em BCD (Binary Coded Decimal, ou Decimal Codificado em Binário = DCB) que consiste em codificar cada dígito decimal separadamente em quatro bits:

Código	Símbolo BCD		Código	Símbolo BCD
0000	0		8	1000
0001	1		9	1001
0010	2		10*	-
0011	3		11*	-
0100	4		12*	-
0101	5		13*	-
0110	6		14*	-
0111	7		15*	-

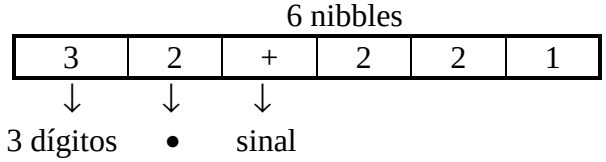
* Codificação não usada em BCD

Em cada byte são codificados dois dígitos BCD. Isso é chamado "BCD Compactado".

Exemplos: 0000 0000 é 00 em BCD
1001 1001 é 99 em BCD

Ainda são usados 1 nibble para indicar o número de dígitos, um nibble para indicar o sinal é um nibble para indicar a posição do ponto binário.

Exemplo: + 2.21 seria representado em BCD assim:



III.10.1 - Aritmética em BCD:

Exemplo 1:

11	0001	0001
+ 22	0010	0010
33	0011	0011
	---	---
	3	3

Exemplo 2:

22	0010 0010	0101 1011	
+ 39	0011 1001	+ 0000 0110	← (6)
61	0101 1011	0110 0001	
	5 ?	6 1	→ Resultado Correto

Vantagem: - Decodificação mais rápida
Desvantagem: - Grande quantidade de memória

- Operações aritmética lentas

III.10 - Representação de Dados Alfa-Numéricos

87 caracteres { 26 letras maiúsculas
26 letras minúsculas
25 caracteres especias
10 dígitos numéricos

$2^7 = 128 \Rightarrow$ 7 bits são suficientes

Código ASCII → American Standard Code for Information Interchange
(usado em microcomputadores)

*1 byte para cada caracter

A - 1000001
B - 1000010
C - 1000011

Bit de paridade: teste de erros em transmissão

Paridade par : n^o de bits 1 é sempre par

Paridade ímpar : n^o de bits 0 é sempre ímpar

Exemplo: Paridade ímpar

A → 1100 0001
B → 1100 0010
C → 0100 0011

III.11 - EXERCÍCIOS

01. Qual é o sistema de numeração mais indicado para ser usado nos computadores? Porque? Qual é a base desse sistema? Quais são os dígitos desse sistema?

02. Indique o valor posicional de cada dígito em negrito nos seguintes números decimais:

a) 3.2**6**4,56 b) 176**3**,34

03. Indique o valor decimal dos seguintes números binários:

a) 111010 b) .1110111 c)1010.0011

04. Expresse o número hexadecimal 84.E como um valor decimal.

05. Transforme os números hexadecimais abaixo em binário:

a) 29 b) 42C c) 63.4F d) 163A7.2E7

06. Processe as seguintes adições em binário:

a) 11001 + 101

b) 100011 + 10010

c) 1010 + 1000 + 1001

07. Processe as seguintes subtrações em binário puro:

a) 1110 - 101

b) 110110 - 1001

08. Processe a seguinte divisão em binário:

100100 : 1100

09. Qual é o complemento-de-um de 1001011010_2?

10. Qual é o complemento-de-dois de 011100101_2?

11. Usando o método de complemento-de-dois, execute A + B em cada caso, usando um byte:

a) A = -10 e B = -15 b) A = +11 e B = -4 c) A = -8 e B = -6

12. Supondo um sistema que reserva uma palavra de 16 bits para armazenamento de números inteiros, faça a configuração para os seguintes inteiros:

a) 345 b) -345 c) 753452 d) -753452

13. Supondo um sistema que reserva 4 bytes para armazenamento de números de ponto flutuante em precisão simples e 8 bytes para armazenamento em precisão dupla, faça a configuração para os seguintes números de ponto flutuante:

a) 34.455 b) -34.455 c) 7.6666 d) -7.6666

14. Explique como os números podem ser representados em BCD num computador.

15. Processe as seguintes adições em BCD:

a) $25_{10} + 13_{10}$ b) $38_{10} + 45_{10}$