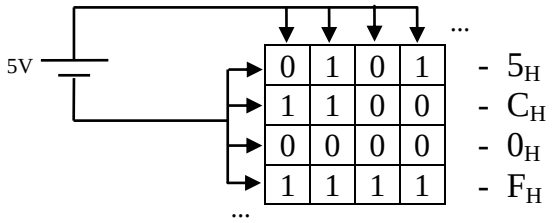
Sistemas Numéricos e o Computador

1 - Computador e Sistemas Numéricos

O computador trabalha basicamente com sistemas numéricos na base 2 (binário). Na realidade todas as informações manipuladas e armazenadas pelo computador são formados por zeros e uns. Fazendo uma analogia podemos considerar que cada caracter (letra) armazenado é formado por inúmeras chaves de liga desliga, que dependendo se estiverem ligadas ou desligadas definem uma informação armazenada.



2 - Bit, Byte e Palavra

O alfabeto do computador é constituído de dois símbolos (dígitos) zeros (0) e uns (1). Combinando um certo número desses símbolos podemos construir mensagens que o computador pode entender. Uma instrução qualquer de um programa pode ser representado internamente como um conjunto de zeros e uns. Exemplo: "LER X" ==> 1001.0110_B ==> 96_H

BIT = dígito binário (zero ou um) sendo a mínima unidade de informação do computador

BYTE = conjunto de 8 bits

$\underbrace{1001}_{9_H}\ \underbrace{1111}_{F_H} = 9F_H$

Um BYTE ocupa uma posição de memória (endereço) equivalendo a um caracter.

1 Byte	= 8 bits
1 kilobyte (KB ou Kbytes)	= 1024 bytes
1 megabyte (MB ou Mbytes)	= 1024 kilobytes
1 gigabyte (GB ou Gbytes)	= 1024 megabytes
1 terabyte (TB ou Tbytes)	= 1024 gigabytes
1 petabyte (PB ou Pbytes)	= 1024 terabytes
1 exabyte (EB ou Ebytes)	= 1024 petabytes
1 zettabyte (ZB ou Zbytes)	= 1024 exabytes
1 yottabyte (YB ou Ybytes)	= 1024 zettabytes

PALAVRA = é um conjunto de bits cujo tamanho (número de bits) depende do computador. Os primeiros computadores desenvolvidos tinham tamanho de palavra de 8 bits ou 1 byte. Com o desenvolvimento do IBM PC-XT (Personal Computer - 1981) a palavra foi definida com 16 bits ou 2 bytes. A linha PENTIUM da Intel foi desenvolvida com uma palavra de 32 bits ou 4 bytes. Atualmente os computadores possuem palavras de 64 bits ou 8 bytes. O tamanho da palavra está diretamente relacionado com a capacidade de endereçamento de memória que o computador possui. Por exemplo se o sistema operacional for de 32 bits o máximo que se consegue endereçar é 4294967295 bytes ou 4 gigabytes. Se o sistema operacional for de 64bits o máximo de endereçamento é 18.446.744.073.709.551.616 bytes ou 17.179.869.184 gigabytes ou 16.384 EB - Exabyte, mas costuma ser limitado pela versão do sistema operacional.

Exemplo, Windows Seven 64 bits:

Home Basic: 8GB
Home Premium: 16GB
Professional / Enterprise / Ultimate: 192GB

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	j
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ ASCII Conversion Chart.doc Copyright © 2008 Donald Weiman 12 August 2008

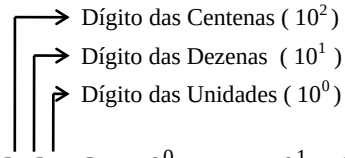
3 - O Sistema Decimal

- Base: 10

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

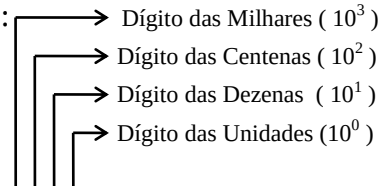
- A contribuição de um dígito num número decimal, depende da posição que ele ocupa.

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.

4 - 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

→ Dígito de 16 (ou 2^4)
→ Dígito de 8 (ou 2^3)
→ Dígito de 4 (ou 2^2)
→ Dígito de 2 (ou 2^1)
→ Dígito de 1 (ou 2^0)

$$10010_2 = 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4$$
$$= 0 + 2 + 0 + 0 + 16 = 18_{10}$$

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

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

$$110.11001_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1x2^{-1} + 1x2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$$
$$= 0 + 2 + 4 + 0{,}5 + 0{,}25 + 0 + 0 + 0{,}03125$$
$$= 6 + 0{,}78125$$
$$= 6{,}78125_{10}$$

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 $2^{\mathbf{n}} - 1$.

4.1 - 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.

```
25 | 2
 1  12 | 2
     0  6 | 2
          0  3 | 2
               1  1
                  ←
```

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

4.2 - Convertendo um número fracionário do sistema decimal para binário fracionário.

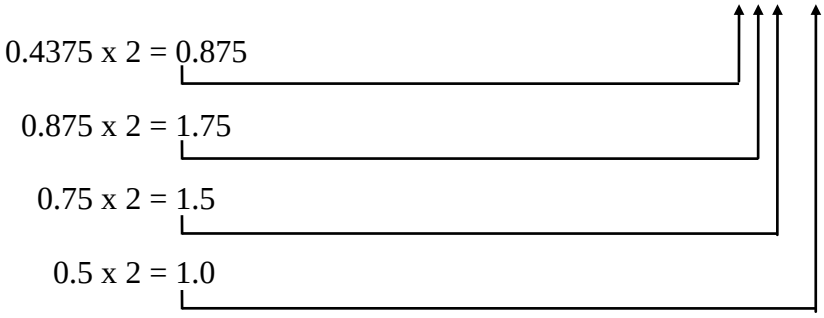
É necessário decompor o número em sua parte inteira e sua parte fracionária. Assim, 102.247 seria decomposto em 102 e 0.247 e a representação de cada parte achada. Agora as duas partes são adicionadas. Já sabemos como transformar um número decimal inteiro em binário inteiro.

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, $25.4375_{10}=11001.0111_2$

5 - 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

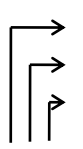
6 - Sistema Hexadecimal

- 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)



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

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

6.1 - 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: $282_{10} = ?_{16}$

$$\begin{array}{rrr} 282 & \vert & 16 \\ \cline{2-3} 10 & 17 & \vert 16 \\ \cline{3-3} (A) & 1 & 1 \end{array}$$

ou seja, $282_{10}=11A_{16}$

6.2 - 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

6.3 - Transformando Hexadecimal Inteiro em Binário

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

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

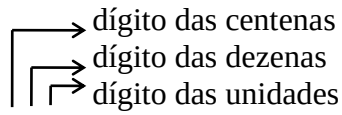
$$\underline{7}\quad\underline{4}\quad\underline{A} \quad\Rightarrow\quad 74A_{16} = 11101001010_2$$
0111 0100 1010

7 - Aritmética Binária

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

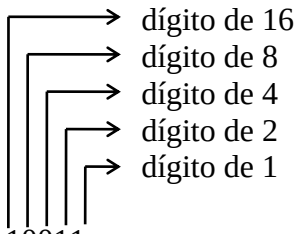
7.1 - Adição Binária

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



$$\begin{array}{r} 235 \\ +\ 46 \\ \hline 281 \end{array}$$

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



$$\begin{array}{r} 10011 \\ +\ 1001 \\ \hline 11100 \end{array}$$

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:

$$\begin{array}{cccc} & & & {\scriptstyle 1\leftarrow\text{vai-um}} \\ 0 & 0 & 1 & 1 \\ +\ 0 & +\ 1 & +\ 0 & +\ 1 \\ \hline 0 & 1 & 1 & 0 \end{array}$$

Ainda se define:

$$\begin{array}{r} {\scriptstyle 1\ \leftarrow\ \text{vai-um}} \\ 1 \\ 1 \\ +1 \\ \hline 1 \end{array}$$

Então :

$$\begin{array}{r} 10011 \\ +\ 1001 \\ \hline 11100 \end{array}$$

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

$$\begin{array}{r} 10 \\ +10 \\ \hline 100 \end{array}$$

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

$$\begin{array}{r} 111 \\ +101 \\ \hline 1100 \end{array}$$

7.2 - Multiplicação de Binários

Exemplo: 10011 x 1011

x	0	1
0	0	0
1	0	1

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

1111001
```

7.3 - 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
```

7.4 - 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)
```

7.5 - 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

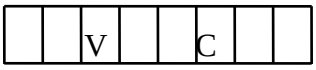
8 - Aritmética Binária em "Complemento-de-dois"

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 → Overflow
C → Carry

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

+5 →	00000011	+3 →	00000011	→	00000001
Comp.-de-um →	11111011	+(-5) →	11111011	+	1
	+1	-2 →	11111110		00000010 → (+2)
-5 →	11111011				

Exemplo: Processar 64+ 65 em "complemento-de-dois" em um byte.
Solução:

+64 →	01000000	→	01111110
+65 →	01000001	+	1 (6)
-127 →	10000001		$01111111 \rightarrow \sum_{i=0} 2^i = 2^7 = 127$

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	OBS. O simétrico de -128,
+1 →	01000001	V=1	+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	

9 - 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.

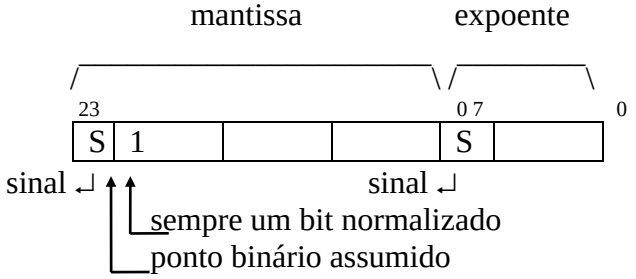
Assim, $0.000123 = 0.00123 \times 10^{-1} = 0.123 \times 10^{-3}$ ← forma normalizada

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

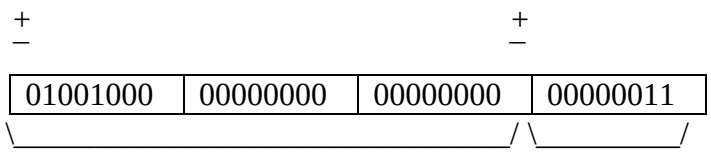
Temos :
4,5 | 2
0 2,25 | 2
0 1,125 | 2
0 0,5625

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.

10 - 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*	-

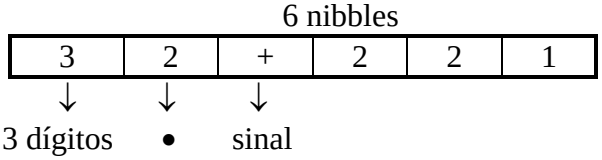
* Codificação não usada em BCD

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:



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

11 - 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 $\rightarrow$ 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 : nº de bits 1 é sempre par

Paridade ímpar : nº de bits 0 é sempre ímpar

Exemplo: Paridade ímpar

A $\rightarrow$ 1100 0001
B $\rightarrow$ 1100 0010
C $\rightarrow$ 0100 0011

12 - 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) $38_{10} + 45_{10}$