REPRESENTAÇÃO DE NÚMEROS INTEIROS

Os computadores digitais utilizam principalmente quatro métodos para representar números inteiros:

- módulo de sinal (MS);
- complemento de 1 (C-1);
- complemento de 2 (C-2);
- excesso de 2 elevado a N-1.

Nessas representações de números utiliza-se o sistema binário e considera-se que temos um número limitado de dígitos para cada dado numérico. Esse número de dígitos disponível é representado por N.

MÓDULO E SINAL (MS)

Neste sistema de representação o bit que está situado mais a esquerda representa o sinal, e o seu valor será:

- 0 para o sinal +; e
- 1 para o sinal -.

Os bits restantes representam o módulo do número.

Exemplo: representar 10 e –10
Limitação de 8 bits (N=8)

10	0	0001010		-10	1	0001010
↓	↓	↓		↓	↓	↓
nº	sinal	módulo		nº	sinal	módulo

Denomina-se **AMPLITUDE ou FAIXA** de representação num determinado método o conjunto de números que podem ser nele representados.
Para o sistema módulo e sinal, a faixa de representação para N dígitos é de:

$$(-2^{N-1} + 1) \leq x \leq (+2^{N-1} - 1)$$

Para oito bits a faixa é...........: $-127 \leq x \leq +127$

Para dezesseis bits a faixa é...: $-32767 \leq x \leq +32767$

Para trinta e dois bits a faixa é: $-2147483647 \leq x \leq +2147483647$

Vantagem: possuir faixa simétrica.
Inconveniência: 2 representações para o número 0.
Para 8 bits o 0 tem as seguintes representações:
00000000 (+0)
10000000 (-0)

COMPLEMENTO DE 1 (C – 1)

Este sistema de representação também utiliza o bit mais à esquerda para o sinal, correspondendo o 0 ao sinal + e o 1 ao sinal -. Para os números positivos, os N- 1 bits da direita representam o módulo. O simétrico de um número positivo é obtido pelo complemento de todos os seus dígitos (trocando 0 por 1 e vice-versa) incluindo o bit de sinal.

Exemplo: representar 10 e –10

Limitação de 8 bits (N=8)

10	0	0001010
↓	↓	↓
nº	sinal	módulo

Número – 10 é o complemento do seu simétrico

-10	1	1110101
↓	↓	↓
nº	sinal	módulo

Neste caso a faixa de representação é

$$(-\,2^{N-1} + 1) \leq x \leq (+\,2^{N-1} - 1)$$

Para oito bits a faixa é...........: $-\,127 \leq x \leq +127$

Para dezesseis bits a faixa é...: $-\,32767 \leq x \leq +\,32767$

Para trinta e dois bits a faixa é: $-\,2147483647 \leq x \leq +\,2147483647$

Vantagem: possuir faixa simétrica.
Inconveniência: 2 representações para o número 0.

Para 8 bits o 0 tem as seguintes representações:

00000000 (+0)
10000000 (-0)

COMPLEMENTO DE 2 (C – 2)

Este sistema de representação utiliza o bit mais à esquerda para o sinal, correspondendo o 0 ao sinal + e o 1 ao sinal -. Para os números positivos, os N- 1 bits da direita representam o módulo, igualmente ao MS e C - 1.

O simétrico de um número é obtido em dois passos:

1º passo – obtém-se o complemento de todos os bits do número positivo (trocando 0 por 1 e vice-versa) incluindo o bit de sinal, isto é, executa-se o complemento de 1;

2º passo – ao resultado obtido no primeiro passo, soma-se 1 (em binário), desprezando o último transporte, se houver.

Exemplo: complemento de 2 dos números 10 e –10

Limitação de 8 bits (N=8)

10	0	0001010
↓	↓	↓
nº	sinal	módulo

Número – 10

1º passo: complemento de 1

-10	1	1110101
↓	↓	↓
nº	sinal	módulo

2º passo:

```
 1110101
+      1
--------
 1110110
```

Neste caso a faixa de representação é

$$(- 2^{N-1}) \leq x \leq (+ 2^{N-1} - 1)$$

Para oito bits a faixa é............: $- 128 \leq x \leq +127$

Para dezesseis bits a faixa é....: $- 32768 \leq x \leq + 32767$

Para trinta e dois bits a faixa é: $- 2147483648 \leq x \leq + 2147483647$

Vantagem: uma única representação para o número 0..

Para 8 bits teremos:

```
Nº  0                 00000000 (+0)
Nº -0   passo 1       11111111 (-0)
        Passo 2     __________1____
                     100000000
                    ↙
               estouro desprezado
```

Logo 0 e –0 tem a mesma representação.

SOMA EM COMPLEMENTO

SOMA EM COMPLEMENTO DE 1 (C – 1)

Na aritmética de complemento de 1, dois números são somados da mesma forma que na representação binária. Com a diferença que, na ocorrência de transporte (carry) na soma parcial dos bits mais à esquerda, este transporte será somado ao resultado.

Exemplo: somar os valores 10 e – 3 em complemento de 1, para 8 bits

10 em complemento de 1 é	00001010	(10)
- 3 em complemento de 1 é	11111100	(-3)
somando	1 00000110	

carry

Observe que houve carry. Este carry deve ser somado ao resultado obtido. Vejamos:

```
  00000110
+        1
----------
  00000111
```

00000111 (7) que é o resultado da operação.

SOMA EM COMPLEMENTO DE 2 (C – 2)

Na aritmética em complemento de 2, o processo é idêntico ao de complemento de 1, mas, desprezando-se o carry, se houver.

Exemplo: somar os valores 10 e – 3 em complemento de 2, para 8 bits

10 em complemento de 2 é	00001010	(10)
- 3 em complemento de 2 é	11111101	(-3)
somando	1 00000111	(7)

carry

Observe que houve carry. Este carry deve ser desprezado.

ARITMÉTICA EM COMPLEMENTO - SOMA EM COMPLEMENTO A UM

O algoritmo da soma em complemento a um é:
a) Somar os dois números, bit a bit, inclusive o bit de sinal.
b) Avaliação dos casos de "vai-um":
b.1) Se não ocorreu vai-um para o bit de sinal nem para fora do número:
--- este é o resultado **correto**;
b.2) Se ocorrer "vai-um" só para o bit de sinal (e não para fora do número):
--- incorreto - ocorreu ***overflow***
------- (isto significa que o resultado excede a faixa de representação para o número de bits adotado)**.**
b.3.1) Se ocorrer "vai-um" para fora do número:
--- para obter o resultado final, **soma-se** o "vai-um" externo (para fora do número) ao resultado da soma; o bit para fora do número (que excede o número de bits adotado na representação) é desprezado.
b.3.2) Nesta soma final também pode ocorrer "vai-um" no último bit;
--- se o número de "vai-um" ocorridos (para o bit de sinal, para fora do número ou na soma final) for par (o que equivale a inverter duas vezes o sinal),
------ o resultado está **correto**;
--- se o número for ímpar (1 ou 3 "vai-um"):
------ o resultado está **incorreto** - ocorreu ***overflow*.**

Exemplos (referidos aos casos acima), considerando representação com 6 bits:

caso b.1: $15 + 10 = 25_{10}$	caso b.2: $15 + 22 = 37_{10}$	caso b.3: $-15 - 10 = -25_{10}$	caso b.3: $-15 - 22 = -37_{10}$
_111	1111	1 1	1
001111 (+)	001111 (+)	110000 (-)	110000 (-)
+ 001010 (+)	+ 010110 (+)	+ 110101 (-)	+ 101001 (-)
011001 (+)	100101 (-)	100101	011001
		+ 1	+ 1
		100110 (-)	011010 (+)
CORRETO	*OVERFLOW*	CORRETO	*OVERFLOW*
(não ocorreu "vai-um")	(só ocorreu "vai-um" para o bit de sinal)	(ocorreu "vai-um" p/ bit de sinal e p/ fora do nº mas não na soma final - nº de "vai-um" é par)	(ocorreu "vai-um" só p/ fora do nº mas não na soma final - nº de "vai-um" é ímpar)
	obs.: soma de dois nº positivos não poderia dar negativo		obs.: soma de dois nº negativos não poderia dar positivo

A faixa de repreentação com 6 bits em C1 vai de -31 a +31. Conferindo as contas pela representação decimal, vemos que os resultados fora desta faixa excedem a faixa da representação e não podem ser representados com 6 bits (portanto, com este número de bits, ocorre *overflow*).

Existe um teste prático que, quando ambos os números tem o mesmo sinal, pode mostrar se ocorreu *overflow* e o resultado obtido está incorreto: basta ver que na soma de dois números negativos, o resultado só pode ser negativo, e a soma de dois números positivos só poderia dar resultado positivo!

ARITMÉTICA EM COMPLEMENTO - SOMA EM COMPLEMENTO A DOIS

O algoritmo da soma (ou subtração) em complemento a dois é:
a) Somar os dois números, bit a bit, inclusive o bit de sinal.
b) Despreza-se o bit para fora do número, se houver.
c.1) Se não ocorreu vai-um para o bit de sinal nem para fora do número ou
c.2) Se ocorrer "vai-um" tanto para o bit de sinal quanto para fora do número (equivale a inverter duas vezes o sinal):
--- o resultado está **correto**;
d.1) Se ocorrer "vai-um" só para o bit de sinal (e não para fora do número):
d.2) Se não ocorrer "vai-um" para o bit de sinal e somente ocorrer para fora do número:
--- o resultado é **incorreto** - ocorreu ***overflow***
------- (isto significa que o resultado excede a faixa de representação para o número de bits adotado)**.**

Nota: Podemos constatar que o algoritmo de soma em C2 é bem mais simples que o de soma em C1.

Exemplos (referidos aos casos acima), considerando representação com 6 bits:

caso c.1: 15+10 = 25_{10}	caso d.1: 15 + 17 = 32_{10}	caso c.2: -15 -10 = -25_{10}	caso c.2: -15 -17 = -32_{10}	caso d.2: -15 -27 = -42_{10}
_111	11111	1	1 11111	1 ____1
001111 (+)	001111 (+)	110001 (-)	110001 (-)	110001 (-)
+ 001010 (+)	+ 010001 (+)	+ 110110 (-)	+ 101111 (-)	+ 100101 (-)
011001 (+)	100000 (-)	100111 (-)	100000 (-)	010110 (+)
CORRETO	*OVERFLOW*	CORRETO	CORRETO	*OVERFLOW*
(não ocorreu "vai-um")	(só ocorreu "vai-um" para o bit de sinal)	(ocorreu "vai-um" p/ bit de sinal e p/ fora do nº - o nº de "vai-um" é par)	(ocorreu "vai-um" p/ bit de sinal e p/ fora do nº - o nº de "vai-um" é par)	(só ocorreu "vai-um" p/ fora do nº - o nº de "vai-um" é ímpar)
	obs.: soma de nº positivos não poderia dar negativo	(despreza-se o "vai-um" para fora do número)	(despreza-se o "vai-um" para fora do número)	obs.: soma de nº negativos não poderia dar positivo (despreza "vai-um" p/ fora do nº)

A faixa de repreentação com 6 bits em C2 vai de -32 a +31. Conferindo as contas pela representação decimal, vemos que os resultados fora desta faixa estão necessariamente incorretos (ocorreu *overflow*)

Pode-se aplicar o mesmo teste prático que, quando ambos os números tem o mesmo sinal, pode mostrar se ocorreu *overflow* e o resultado obtido está incorreto: basta ver que na soma de dois números negativos, o resultado só pode ser negativo, e a soma de dois números positivos só poderia dar resultado positivo!

ARITMÉTICA EM SINAL E MAGNITUDE

Em sinal e magnitude, o algoritmo da soma é:

a) Se os números forem de mesmo sinal, basta somar os dois números e manter o sinal;

b) Se os números forem de sinais diferentes, subtrai-se o menor número do maior; o sinal será o do maior número.

c) Tem-se *overflow* sempre que ocorrer "vai-um" da magnitude para o bit de sinal (o número de bits da magnitude foi excedido).

	__1			1 1
0010 (positivo)	1001 (negativo)	0110 (positivo e maior)	1110 (negativo e maior)	1110 (negativo)
+ 0101 (positivo)	+ 1101 (negativo)	+ 1010 (negativo e menor)	+ 0100 (positivo e menor)	+ 1101 (negativo)
0111	1110	0100	1010	1011
CORRETO	CORRETO	CORRETO	CORRETO	*OVERFLOW*
ambos positivos -> soma e mantém o sinal positivo	ambos negativos -> soma e mantém o sinal negativo	sinais contrários -> subtrai maior magnitude é positivo -> resultado é positivo	sinais contrários -> subtrai maior magnitude é negativo -> resultado é negativo	ambos negativos -> soma "vai-um" para bit de sinal -> overflow