SISTEMAS DE COMPUTAÇÃO

REPRESENTAÇÃO DE NÚMEROS INTEIROS

↳ Os seres humanos usam um sistema de numeração baseado na base 10 → base decimal
10 dígitos ≠ (0 a 9)

↳ Os computadores usam um sistema de numeração baseado na base 2
↓ ou dígitos binários também conhecidos por bits.
usa só tem 2 algarismos:
o 0 e o 1 !!

ORDEM → é dada pela posição que um dígito ocupa num nº : 0 é a ordem do dígito imediatamente à esquerda do ponto decimal, e vai crescendo no sentido da esquerda e vai diminuir no sentido da direita.

BASE → determina o nº de dígitos que vão utilizados:
- base 10 : 10 dígitos (0 a 9)
- base 2 : 2 dígitos (0 e 1)
- base 16 : 16 dígitos (0 a 9 e A a F)
  ou em minúsculas: a a f

CONVERSÃO DE UMA BASE B QUALQUER PARA DECIMAL

↓

obtém-se multiplicando cada dígito pela base b elevada à ordem do dígito e somando todos os valores.

EXEMPLO:

$1532_6 = 1\times6^3 + 5\times6^2 + 3\times6^1 + 2\times6^0 = 416_{10}$

este índice significa que o nº está em base 6

este índice significa que o nº está em base 10 (muita base, em é necessário colocar).

$110\,110,011_2 = 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} =$
$= 54.375_{10}$

↓
o nº está na base 2.

PI ↓ parte inteira ÷ ↓ divisão

PD ↓ parte decimal * ↓ multiplicação

CONVERSÃO DE UMA BASE DECIMAL PARA UMA BASE B QUALQUER

• PARTE INTEIRA → ÷ sucessiva da PI desse nº pela respetiva base b, usando os restos obtidos com cada uma das ÷, os dígitos da base b (a começar com o + junto ao ponto decimal) e os quocientes a usar na sucessão de ÷.

• PARTE DECIMAL / FRACIONÁRIA → * sucessiva da PD desse nº pela respetiva base b, usando a PI de cada um dos produtos obtidos, os dígitos da base b (a começar pelo + junto ao PD) e a PD do produto obtido a usar na sucessão de *.

EXEMPLO:

$416_{10} = 1532_6$

416	6
56	69 | 6
②	09 11 | 6
	③ ⑤ ①

a ordem é nesta direção

$54.375_{10} = 110110.011_2$

54	2
14	27 | 2
⓪	07 13 | 2
	① ① 6 | 2
	⓪ 3 | 2
	① ①

0.375	
	× 2
⓪.750	
	× 2
①.500	
	× 2
①.000	

CASO EXCEÇÃO: BASE DECIMAL ⇒ BASE BINÁRIA

↳ existe outro processo que utiliza subtrações sucessivas, mas apenas é usado para base binária para valores menores à ordem de grandeza dos milhares e para nºs inteiros.

↓

VANTAGEM: rapidez de cálculo mental desde que se saiba a "tabuada" das potências de 2.

→

TABUADA DAS POTÊNCIAS DE 2 (de 2^0 a 2^{10})

$2^0 = 1$	$2^6 = 64$
$2^1 = 2$	$2^7 = 128$
$2^2 = 4$	$2^8 = 256$
$2^3 = 8$	$2^9 = 512$
$2^4 = 16$	$2^{10} = 1024$
$2^5 = 32$	

Quando temos acima, vamos para os múltiplos:

$2^{10} = 1024 = 1$ Ki(bi) → dar em atenção que é parecido com $10^3 = 1k$

$2^{20} = 1$ Mi(bi)

$2^{30} = 1$ Gi(bi)

$2^{40} = 1$ Ti(bi)

$2^{50} = 1$ Pi(bi)

⇒ Um exemplo de uso destes valores pode ser:

$2^{12} = 2^2 * 2^{10} = 4 * 1$ Ki(bi) = $= 4$ Ki(bi)

⇓

Método das subtrações sucessivas → procura-se a maior potência de 2 imediatamente inferior ao valor do nº decimal e subtrai-se essa potência do nº decimal. O expoente da potência indica que o nº binário terá um bit 1 nessa ordem. Com o resultado da subtração repete-se o processo até chegar a 0.

NOTA: A quantidade de bits utilizados determina a gama de valores representáveis. Num sistema de numeração qualquer, a gama de valores representáveis com n dígitos é b^n.

Para a representação de valores binários, sendo n o nº de bits utilizados, a gama de valores representáveis em binário, usando n bits é 2^n.

BASE HEXADECIMAL → é usada para ajudar na representação de valores binários.

São utilizados 16 dígitos:

$0, 1, 2, \ldots, 9, \overset{A}{a}, \overset{B}{b}, \overset{C}{c}, \overset{D}{d}, \overset{E}{e}, \overset{F}{f}$

$\downarrow 10 \quad \downarrow 11 \quad \downarrow 12 \quad \downarrow 13 \quad \downarrow 14 \quad \downarrow 15$

- **DECIMAL ⇔ HEXADECIMAL** → processos que foram descritos anteriormente!

- **HEXADECIMAL ⇔ BINÁRIO** → a partir do binário, agrupando os bits de 4 em 4 a partir do ponto decimal, convertendo depois cada um desses conjuntos de 4 bits num só dígito hexadecimal; para binário, convertendo cada dígito hexadecimal em 4 bits.

EXEMPLO:

1010	0110	1000	0111	0110	0101	1101	0100
$2^3+2^1=$	$2^2+2=$	$2^3=8$	$2^2+2+2^0=$	$2^2+2^1=$	$2^2+2^0=$	$2^3+2^2+2^0=$	$2^2=4$
$=10=$	$=6$		$=7$	$=6$	$=5$	$=13=d$	
$=a$							

$\therefore 10100110100001110110010111010100_2 = \underbrace{a68765d4}_{16}$

pode-se representar assim:

$0xa68765d4$

PARA REPRESENTAR NÚMEROS NEGATIVOS:

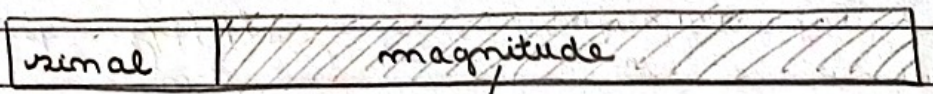
- **SINAL + AMPLITUDE/MAGNITUDE (S+M)**

Utiliza o bit + significativo para representar o sinal, ou seja, o bit + à esquerda (o 1º bit):

→ se for 0 indica um nº positivo

→ se for 1 indica um nº negativo

Os restantes bits representam a amplitude/magnitude:



o valor dos bits usados para representar a magnitude é independente do sinal, isto é, seja o nº positivo ou negativo, a sua representação binária da magnitude é a mesma.

PROBLEMA: existem 2 representações para o zero.

→

EXEMPLO: 6 bits !!

5 } 0 (S) 00101 (A) → 000101_2

-5 } 1 (S) 00101 (A) → 100101_2

- COMPLEMENTO PARA 1: → C1

Os complementos são estratégias para representar valores negativos e apenas negativos. Portanto, se o valor for positivo, fica igual à representação de S+M, mas se o valor for negativo vega-se ao seu valor positivo correspondente e invertem-se todos os seus bits (trocar 0 por 1 e 1 por 0).

EXEMPLO: 6 bits !! complemento para 1 → logo a representação é feita por binário puro.

5 } 000101_2 } $\bar{m}$ contempla estes valores

-5 } 000101_2 → 111010_2 (inverte os bits)

↑ representamos o valor positivo correspondente

PROBLEMA: continuamos a ter 2 representações para o zero.

Se quisermos fazer o processo inverso:

* Sabendo que 000101_2 está em C1, converter o decimal correspondente:

1) Isolar o 1º bit e olhar para o seu valor

$\underline{0}00101_2$

↳ este valor é positivo

↓

logo, ele foi codificado como binário puro e basta fazer a conversão normal:

$000101_2 \Rightarrow 5_{10}$

* Sabendo que 111010_2 está em C1, converter o decimal correspondente:

1) Isolar o 1º bit e olhar para o seu valor:

$\underline{1}11010_2$

↳ nº negativo

2) Esquecemos o sinal por agora e invertemos todos os bits mas nº:

$000101_2 \rightarrow \underbrace{5_{10}}_{\text{este nº é o resultado final}}$

3) Como verificamos no passo 1, o nº é negativo. Portanto, a resposta final é -5_{10}.

• COMPLEMENTO PARA 2 → C2

O método é igual ao C1 mas para determinar os valores negativos iremos ter um passo a mais que é somar 1 depois de inverter os nºs:

→ Para os valores positivos é =

→ Para os valores negativos vamos ter que:

1) Vemos o sinal e anotamos essa informação sem tirar o 1º bit
2) invertemos todos os bits
3) somamos 1
4) apresentamos a resposta final com o sinal visto no passo 1!

PROBLEMA: pode resolver o problema das 2 representações do 0 mas a gama de valores deixa de ser simétrica e passamos a ter + um elemento.

EXEMPLO: 6 bits!!

5 } 000101_2

-5 } 000101_2 → (invertemos os bits) → 111010_2 → (somamos 1 unidade, +1) → 111011_2 ← resultado final

representamos a magnitude

Se quisermos fazer o processo inverso:

★ 111011_2:

1) Vemos o 1º bit e verificamos que é negativo ← guardar esta informação!!
2) Invertemos os bits todos:

000100_2

3) Depois de invertermos os dígitos, somamos uma unidade:

$000100_2 + 1_2 = 000101_2$

4) Convertemos o valor obtido no passo 3 para decimal

$000101_2 \Rightarrow (5_{10}) \rightarrow$ é o resultado final

5) Como sabemos, do passo 1, que o sinal tem de ser o negativo, então temos que:

$111011_2 = -5_{10}$

• EXCESSOS

Usar um excesso significa que pegamos num valor na base decimal e iremos representar na base binária usando +, isto é, vamos representar em binário uma representação inflacionada do valor.

↳ ou seja, fazer uma soma

NOTA: Formalmente, o excesso é dado !!

EXEMPLO: Queremos representar com 4 dígitos o valor 3_{10} com excesso de 5:

⚠ No excesso, são considerados os valores positivos e negativos (≠ C1 e C2, que só consideram os negativos).

Como queremos representar o 3_{10} em excesso, então vai acontecer uma inflação do seu valor, logo iremos fazer uma soma:

1) Pegar no valor e somar com o excesso:

$3 + 5 = 8$

2) Verificar se com 4 dígitos é possível representar o 8:

$8_{10} = (1000_2)$

↳ é o resultado final!!
(notação por excesso)

Se quisermos fazer o processo inverso:

★ 0100_2

Se foi inflacionado para colocar em binário, então vamos ter que retirar a inflamação para podermos passar para decimal.

1) Ver quanto 0100_2 é em decimal:

$0100_2 = 4_{10}$

↳ valor inflacionado

2) Subtrair o excesso ao valor obtido no passo 1:

$4 - 5 = \underbrace{-1}_{10}$

↳ podemos passar para binário (o resultado fica -1) porque não conseguimos representar -1 um binário puro.