

Bases de Numeração
e
Representação de Números em Computador

Ana Paula Tomás
Nelma Moreira

Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: {apt,nam}@ncc.up.pt

1 Bases de numeração

1.1 Sistema de representação posicional

Num sistema de numeração de base (ou raiz) b usa-se b símbolos diferentes para b dígitos (de 0 a $b - 1$). Os números são representados por uma sequência de dígitos.

Exemplo: Base 10

$$1358 = 1 \times 10^3 + 3 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$

Os dígitos na base 10 são 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Os dígitos na base 2 são 0 e 1, e normalmente são designados por *bits*. Por exemplo,

$$101110_{(2)} = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

O seguinte teorema justifica esta representação:

Teorema 1 Algoritmo de divisão de inteiros (divisão euclidiana de inteiros):

$$\forall a \in \mathbb{Z} \forall b \in \mathbb{Z}^+ \exists! q \in \mathbb{Z} \exists! r \in \mathbb{Z} : b = aq + r \wedge 0 \leq r < b$$

Obs: q é o quociente e r o resto da divisão inteira de a por b

Corolário 1

$$\forall a \in \mathbb{Z}^+ \forall b \in \mathbb{Z}^+ \setminus \{1\} \exists! r_0, r_1, \dots, r_m \in \mathbb{Z} : 0 \leq r_0, r_1, \dots, r_m < b \wedge r_m \neq 0 \wedge a = r_m b^m + r_{m-1} b^{m-1} + \dots + r_2 b^2 + r_1 b + r_0$$

Dem. Sejam a e b inteiros com $a > 0$ e $b > 1$. Tem-se ou $a < b$ ou $a \geq b$. Se $a < b$ então $a = 0b + a$. Portanto, $0 < r_0 = a$. Se $a \geq b$ então, pelo algoritmo de divisão inteira, existem q_0 e r_0 únicos tais que $a = bq_0 + r_0$, com $0 \leq r_0 < b$. Se $q_0 < b$, toma-se $r_1 = q_0$ e obtém-se $a = r_1 b + r_0$. Se $q_0 \geq b$, então o processo repete-se agora para q_0 . Ou seja, $q_0 = bq_1 + r_1$, com $0 \leq r_1 < b$. Logo,

$$a = bq_0 + r_0 = b(bq_1 + r_1) + r_0 = b^2 q_1 + br_1 + r_0$$

Se $q_1 < b$, toma-se $r_2 = q_1$ e obtém-se $a = b^2 r_2 + br_1 + r_0$. Se $q_1 \geq b$, então o processo repete-se agora para q_1 . O processo termina porque $a > q_0 > q_1 > \dots$ e qualquer q_i é não negativo. \square

$r_0, r_1, r_2, \dots, r_m$ dizem-se *dígitos de representação* de \underline{a} na base \underline{b} , e escreve-se

$$a = r_m \dots r_2 r_1 r_0^{(b)}$$

r_0 diz-se o dígito *menos significativo* e r_m o dígito *mais significativo*.

Exemplos:

(i)

$$125_{(10)} = 5^3 = 1000_{(5)} = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 1111101_{(2)} = 175_{(8)} = 7D_{(16)}$$

$\begin{array}{r} 125 \overline{) 5} \\ \underline{0} \\ r_0 \\ \underline{0} \\ r_1 \\ \underline{0} \\ r_2 \\ \underline{1} \\ r_3 \end{array}$	$\begin{array}{r} 125 \overline{) 2} \\ \underline{1} \\ r_0 \\ \underline{0} \\ r_1 \\ \underline{1} \\ r_2 \\ \underline{1} \\ r_3 \\ \underline{1} \\ r_4 \\ \underline{1} \\ r_5 \\ \underline{1} \\ r_6 \end{array}$	$\begin{array}{r} 125 \overline{) 8} \\ \underline{5} \\ r_0 \\ \underline{7} \\ r_1 \\ \underline{1} \\ r_2 \end{array}$	$\begin{array}{r} 125 \overline{) 16} \\ \underline{13} \\ r_0 \\ \underline{7} \\ r_1 \end{array}$
---	--	---	---

(ii)

$$101110_{(2)} = 46_{(10)}$$

$$101110_{(2)} = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 32 + 8 + 4 + 2 = 46_{(10)}$$

$$\begin{array}{r} 46 \overline{) 2} \\ \underline{0} \\ r_0 \\ \underline{1} \\ r_1 \\ \underline{1} \\ r_2 \\ \underline{1} \\ r_3 \\ \underline{0} \\ r_4 \\ \underline{1} \\ r_5 \end{array}$$

$$\begin{aligned} 46_{(10)} &= 23 \times 2 + 0 = (11 \times 2 + 1) \times 2 + 0 = ((5 \times 2 + 1) \times 2 + 1) \times 2 + 0 = \\ &(((2 \times 2 + 1) \times 2 + 1) \times 2 + 1) \times 2 + 0 = \\ &((((2 \times 1 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 1) \times 2 + 0 = \\ &1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 101110_{(2)} \end{aligned}$$

Para além da base 2 são habitualmente usadas em computação as seguintes bases de numeração:

Octal (base 8) e Hexadecimal (base 16): Os dígitos em octal são 0, 1, 2, 3, 4, 5, 6 e 7. Embora os restos da divisão por 16 sejam 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,...

15, por convenção, os dígitos em hexadecimal a partir de 10 são representados pelas letras maiúsculas de A a F. Os números representados na base octal são habitualmente indexados por (o) , por exemplo $235_{(o)}$, e os na base hexadecimal por (h) , por exemplo $F15A_{(h)}$.

Em resumo:

Designação	Base	Dígitos
Binário	2	0 a 1
Octal	$8(=2^3)$	0 a 7
Hexadecimal	$16(=2^4)$	0 a 9, A, B, C, D, E, F
Decimal	10	0 a 9

Exercício 1 Das sequências seguintes indique as que podem ser representações de inteiros na base 4, e determine as representações base 9 correspondentes: 78412, 12323, 0012323, 1232300.

Exercício 2 Converter a binário: $153_{(10)}$, $153_{(6)}$, $153_{(8)}$, $153_{(16)}$.

Exercício 3 Converter a hexadecimal: $153_{(10)}$, $151323_{(10)}$, $153_{(8)}$, $1010101111_{(2)}$.

Exercício 4 Converter a octal: $1123_{(4)}$, $151323_{(10)}$, $153_{(8)}$, $1010101111_{(2)}$.

Exercício 5 Converter para as bases 251 e 666 os seguintes números em decimal:

$$1383, 1498, 1500, 1580, 1640$$

Exercício 6 Represente:

- (i) x na base x ($x \in N$)
- (ii) x^n e $x^n - 1$ em base x ($x, n \in N$)

1.1.1 Conversões de Binário a Octal e Hexadecimal e vice-versa

Seja por exemplo, $101011110_{(2)}$ um binário que se pretende converter a octal. Como foi visto,

$$1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 0$$

é o inteiro que ele representa. A representação octal correspondente pode ser obtida agrupando os dígitos binário 3 a 3: uma vez que $8 = 2^3$ e $8^2 = 2^6$, tem-se

$$2^6 \times (1 \times 2^2 + 0 \times 2 + 1 \times 2) + 2^3 \times (0 \times 2^2 + 1 \times 2 + 1) + (1 \times 2^2 + 1 \times 2 + 0) = 8^2 \times 5 + 8 \times 3 + 6 = 536_{(2)}$$

Em geral se $a_k \dots a_4 a_3 a_2 a_1 a_0_{(2)}$, sendo $k \geq 0$, representa o inteiro

$$a_k 2^k + \dots + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

então a representação em octal do mesmo inteiro pode ser obtida da forma descrita:

$$\begin{aligned} a_k 2^k + \dots + 2^3(a_5 2^2 + a_4 2^1 + a_3 2^0) + (a_2 2^2 + a_1 2^1 + a_0 2^0) = \\ a_k 2^k + \dots + 8(a_5 2^2 + a_4 2^1 + a_3 2^0) + (a_2 2^2 + a_1 2^1 + a_0 2^0) \end{aligned}$$

Notar que na expressão resultante, qualquer potência de 8 tem por coeficiente

$$a_{i+2} 2^2 + a_{i+1} 2 + a_i$$

para algum i , o qual é sempre não negativo e inferior a 8, podendo assim ser dígito da representação octal. A cada dígito octal correspondem 3 dígitos em binário. Do mesmo modo, a cada dígito hexadecimal correspondem 4 dígitos em binário. Assim, para converter um binário a hexadecimal, agrupa-se os seus dígitos em grupos de 4 correspondendo a cada um desses grupos um dígito hexadecimal. Por exemplo,

$$101011110_{(2)} \Rightarrow 1 \mid 0101 \mid 1110 \Rightarrow 1 \mid 5 \mid E \Rightarrow 15E_{(h)}$$

Para converter um binário a octal procede-se de modo idêntico mas formando grupos de 3. Por exemplo,

$$101011110_{(2)} \Rightarrow 101 \mid 011 \mid 110 \Rightarrow 5 \mid 3 \mid 6 \Rightarrow 536_{(o)}$$

Inversamente, se se pretender converter de hexadecimal a binário basta associar a cada um dos dígitos hexadecimal o grupo de 4 dígitos binários correspondente. Por exemplo,

$$\begin{aligned} 3AC3A_{(h)} \Rightarrow 0011 \mid 1010 \mid 1100 \mid 0011 \mid 1010 \Rightarrow \\ 11 \mid 1010 \mid 1100 \mid 0011 \mid 1010 \Rightarrow 111010110000111010_{(2)} \end{aligned}$$

Notar a eliminação dos zeros não significativos. O que se acaba de ilustrar é um caso particular da seguinte proposição:

Proposição 1 *Se z é um inteiro positivo, a cada dígito (com possível exceção do mais significativo) da representação de z na base b^n corresponde um grupo de n dígitos na representação de z na base b , qualquer que seja n inteiro positivo. Mais concretamente, se*

$$a_k a_{k-1} \dots a_{tn} a_{tn-1} \dots a_{2n} a_{2n-1} \dots a_n a_{n-1} \dots a_1 a_0$$

com $k \geq 0$ é a representação na base b de um inteiro positivo z então, a representação do mesmo inteiro na base b^n é

$$w_t w_{t-1} \dots w_1 w_0$$

sendo $t+1$ o número de blocos de n dígitos da representação de z na base b (podendo o último ser completado por zeros não significativos), e w_i ($i \leq t$) o dígito da base b^n que representa o inteiro $a_{2in-1} \dots a_{in+1} a_{in(b)}$ (representação base b a menos de zeros não significativos).

Exercício 7 Repita os exercícios 2 a 4.

Exercício 8 Mostre a proposição anterior. **Sugestão:**

- (i) comece por mostrar que $w_t w_{t-1} \dots w_1 w_0$ conforme descrito pode ser representação base b^n de z ; use em seguida o facto da representação numa base ser única para concluir que $w_t w_{t-1} \dots w_1 w_0$ é a representação de z .
- (ii) mostre depois que se $w_t w_{t-1} \dots w_1 w_0$ representa z na base b^n , a representação de z na base b é

$$a_k a_{k-1} \dots a_{tn} a_{tn-1} \dots a_{2n} a_{2n-1} \dots a_n a_{n-1} \dots a_1 a_0$$

1.1.2 Adição e Multiplicação na base b

Recorde como se adicionam dois inteiros representados na base 10, calculando por exemplo $987654 + 73561$.

Exercício 9 Justifique que se x e y são inteiros positivos representados na base 10 por $x_k x_{k-1} \dots x_1 x_0$ e $y_m y_{m-1} \dots y_1 y_0$ respectivamente então $s_p s_{p-1} \dots s_1 s_0$, a representação na base 10 de $x + y$, pode ser obtida da forma seguinte: No caso de $x_0 + y_0 < 10$, toma-se $s_0 = x_0 + y_0$ e repete-se o processo para os dígitos seguintes. Senão, s_0 é tal que $x_0 + y_0 = 1s_0$, adicionando-se 1 a $x_1 + y_1$ repetindo-se o processo (global) para os dígitos seguintes. Quando $k < m$ (respectivamente $m < k$) pode-se considerar $x_i = 0$, $i \geq k$ (respectivamente $y_i = 0$, $i \geq m$). Notar que $p = \max(k, m)$ ou $p = \max(k, m) + 1$ sendo neste último caso $s_p = 1$.

Exercício 10 Justifique que se x e y são inteiros positivos representados na base b por $x_k x_{k-1} \dots x_1 x_0$ e $y_m y_{m-1} \dots y_1 y_0$ respectivamente então $s_p s_{p-1} \dots s_1 s_0$, a representação na base b de $x + y$, pode ser obtida por um processo análogo ao descrito em 9 (isto é, seguindo a regra/algoritmo habitual).

As regras/algoritmos usuais para adição e multiplicação base 10 são válidas quando se utilizam representações em qualquer outra base, embora sejam obviamente diferentes as tabuadas dessas operações. O mesmo se pode dizer para subtracção (aditivo maior do que subtractivo) e divisão inteira. Assim, por exemplo:

Tabuadas de adição e multiplicação binário:

$+_2$	0	1	\times_2	0	1
0	0	1	0	0	0
1	1	10	1	0	1

Tabuadas de adição e multiplicação em octal:

$+_8$	0	1	2	3	4	5	6	7	\times_8	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7	0	0	0	0	0	0	0	0	0
1	1	2	3	4	5	6	7	10	1	0	1	2	3	4	5	6	7
2	2	3	4	5	6	7	10	11	2	0	2	4	6	10	12	14	16
3	3	4	5	6	7	10	11	12	3	0	3	6	11	14	17	22	25
4	4	5	6	7	10	11	12	13	4	0	4	10	14	20	24	30	34
5	5	6	7	10	11	12	13	14	5	0	5	12	17
6	6	7	10	11	12	13	14	15	6	0	6
7	7	10	11	12	13	14	15	16	7	0	7	61

Exercício 11 Complete a tabuada de multiplicação base 8, e determine as tabelas para a base hexadecimal.

Exercício 12 Calcule:

- (i) $1110010_{(2)} + 111001111_{(2)}$ em binário.
- (ii) $1F5_{(h)} + 111001111_{(2)}$ em hexadecimal.
- (iii) $1330_{(4)} + 123_{(4)}$ em octal.
- (iv) $1F5_{(h)} + ABCD_{(h)} + 1FB_{(h)}$ em hexadecimal.
- (v) $ABCD_{(h)} - 1FB_{(h)}$ em hexadecimal.
- (vi) $73542_{(o)} \times 27_{(10)}$ em octal.
- (vii) $73542_{(o)} / 27_{(10)}$ em octal.
- (viii) $111000010_{(2)} / 111_{(2)}$ em binário.

Exercício 13 (i) Qual a condição para que um inteiro representado na base 10 seja divisível por 100? E se a base for 2, para que seja divisível por 4?

- (ii) Dado um inteiro representado numa certa base b qual é a condição necessária e suficiente que deve ser satisfeita por essa representação para que o inteiro seja divisível por b^k , sendo k inteiro não negativo. Justifique.

Exercício 14 Justifique o seguinte critério de divisibilidade por 9: Um inteiro é divisível por 9 se a soma dos dígitos da sua representação na base 10 for divisível por 9.

Exercício 15 Justifique que um critério suficiente de divisibilidade por 11 é o seguinte: Um inteiro é divisível por 11 se a soma dos dígitos de ordem par da sua representação na base 10 for igual à soma dos dígitos de ordem ímpar.

1.2 Representação de números com um número fixo de dígitos

Num computador cada inteiro é representado por um número fixo de bits. Em 8 bits, 13 seria representado por 00001101. Isto é, introduzem-se 0 à esquerda sempre que o número de bits da representação do inteiro seja menor que o número fixo de bits. Se tal número for n , os bits são designados da direita para a esquerda por $bit_0, bit_1, \dots, bit_{n-1}$. O bit_{n-1} diz-se o bit *mais significativo*.

Por outro lado, ao fixar-se o número de bits da representação limita-se os valores que podem ser representados.

Se o número de bits for 8 o maior inteiro positivo que se pode representar é:

1	1	1	1	1	1	1	1
$bit\ 7$	$bit\ 6$	$bit\ 5$	$bit\ 4$	$bit\ 3$	$bit\ 2$	$bit\ 1$	$bit\ 0$

cujo valor é

$$2^7 \times 1 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = 255 (= 2^8 - 1)$$

Em geral, com n bits podemos representar números inteiros positivos de 0 a $2^n - 1$.

Teorema 2 *O maior inteiro positivo que se pode representar na base b com n dígitos é $b^n - 1$.*

Dem. Suponhamos que A é representado com n dígitos na base b por $a_{n-1}a_{n-2} \dots a_1a_0$ com $a_i \in \{0, \dots, b-1\}$. Então,

$$A = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} \dots a_1b^1 + a_0b^0 = \sum_{i=0}^{n-1} a_i b^i$$

Ora,

$$\begin{aligned} \sum_{i=0}^{n-1} a_i b^i &\leq \sum_{i=0}^{n-1} (b-1)b^i \\ &= (b-1) \sum_{i=0}^{n-1} b^i \\ &= (b-1) \frac{b^0 - b \times b^{n-1}}{1-b} \\ &= (b-1) \frac{1-b^n}{1-b} \\ &= b^n - 1 \end{aligned}$$

□

Exercício 16 (i) Qual é o número mínimo de bits necessário para representar 1125?

(ii) Qual o valor máximo que pode ser representado com esse número de bits?

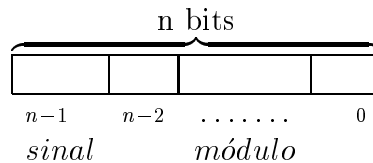
Normalmente o número de bits usados são 8, 16, 32 ou 64. Com 8 bits podemos representar inteiros entre 0 e 255, com 16 bits entre 0 e 65535, com 32 bits entre 0 e 4294967295, etc.

1.2.1 Representação de números negativos

Para representar números inteiros positivos e negativos numa base b e com um número fixo n de dígitos é necessário codificar o sinal e encontrar um processo eficiente de determinar se um número é positivo ou negativo. Normalmente é reservado um dígito para indicar o sinal. Por exemplo, sendo $A = a_{n-1}a_{n-2} \dots a_1a_0_{(b)}$, se $a_{n-1} = 0$ então A é positivo, se $a_{n-1} = b - 1$ o número é negativo.

Vamos considerar duas maneiras habituais de representar números inteiros positivos e negativos e que obedecem à condição de sinal apresentada. Vamos supor que a base é a 2 e que o número de dígitos é n , mas os resultados podem ser generalizados para qualquer base b .

Representação com sinal Reserva-se o bit mais à esquerda para o sinal e os restantes para o módulo do número.



O bit de sinal é 0 para os positivos e é 1 para os negativos. Assim, um número positivo é da forma $A = 0a_{n-2} \dots a_1a_0$ e um negativo $A = 1a_{n-2} \dots a_1a_0$. Com n bits podemos representar números positivos de 0 a $2^{n-1} - 1$ e negativos de $-(2^{n-1} - 1)$ a 0.

As representações de dois números com o mesmo módulo diferem apenas no bit de sinal. Se $n = 3$ temos o seguinte quadro:

<i>Valor</i>	<u>Representação com sinal</u>
0	000
1	001
2	010
3	011
-0	100
-1	101
-2	110
-3	111

Um problema deste método é que o zero tem duas representações: $+0$ e -0 . Mais importante, ainda é que para fazer adições de números de sinal diferente é necessário primeiro determinar qual é o maior e qual o sinal do resultado. O mesmo problema surge para a subtração, que pode ser tratada como a adição associando o sinal negativo ao subtraendo, $x - y = x + (-y)$.

Complemento para 2 Neste método pretende-se que metade dos números representados sejam não negativos e a outra metade corresponda a números negativos. De 0 a

$2^{n-1} - 1$ tem-se números não negativos e de 2^{n-1} a $2^n - 1$ números negativos. Para os positivos a representação coincide com a representação anterior. Os números negativos são complementos para a base. Por definição, o *complemento para 2* de x numa representação de n bits é o inteiro $2^n - x$.

Se $n = 3$ então -1 é representado por 111 pois $2^3 - 1 = 7$.

Em resumo, temos o seguinte quadro para $n = 3$:

<i>Valor</i>	<i>Complemento para 2</i>
0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111

Usando a definição de complemento para 2, pode-se concluir que 2^{n-1} representa -2^{n-1} e $2^n - 1$ representa -1 .

No sistema de representação com n bits e complemento para 2 os valores possíveis dos inteiros representáveis variam de -2^{n-1} a $2^{n-1} - 1$. Note que os números negativos estão “ordenados” por ordem decrescente do seu valor absoluto, isto é, $\underbrace{100\dots 0}_{n-1}$ representa o negativo com maior valor absoluto (-2^{n-1}) e $\underbrace{11\dots 1}_n$ representa o número negativo com menor valor absoluto (-1).

Na tabela seguinte compara-se o significado de seqüências de n -bits para as três representações de inteiros (com n -bits) apresentadas.

	00 ... 0	01 ... 1	10 ... 0	10 ... 01	11 ... 1
Sem sinal	0	$2^{n-1} - 1$	2^{n-1}	$2^{n-1} + 1$	$2^n - 1$
Com sinal	0	$2^{n-1} - 1$	0	-1	$-(2^{n-1} - 1)$
Complemento para 2	0	$2^{n-1} - 1$	-2^{n-1}	$-(2^{n-1} - 1)$	-1

Para além de só haver um zero, neste caso todos os negativos tem 1 como bit mais significativo, que também funciona como bit de sinal. Outra vantagem deste método é que as adições podem ser feitas sem analisar o sinal dos operandos e nas subtrações basta calcular o complemento para 2 do subtraendo e adicionar o valor resultante ao subtrativo.

Exercício 17 Quais os inteiros que podem ser representados num sistema de complemento para 2 em 8, 16, e 32 bits?

Como $2^n - x = (2^n - 1 - x) + 1$ e $2^n - 1 - x$ pode ser obtido complementando todos os dígitos de x (isto é trocando os uns com os zeros e os zeros com os uns, uma vez que $2^n - 1 = (\underbrace{11\dots 1}_n)$), pode-se usar a seguinte regra prática para calcular o complemento para 2 de um inteiro positivo x :

toma-se a representação em binário de x com n bits; troca-se nessa representação cada 1 em 0 e cada 0 em 1 e adiciona-se ao resultado 1

Por exemplo, para representar o inteiro -3 em 8 bits e complemento para 2, toma-se 0000011 (binário 3); efectua-se a troca indicada obtendo-se 1111100 e soma-se 1 resultando 1111101.

Exercício 18

1. Qual a representação em 8-bits e complemento para 2 dos números -120 e -1 ?
2. Determine a representação 8-bits e complemento para 2 dos números 01100110, 10001000 e 11111111.

Para obter o valor decimal basta considerar que a parcela $a_{n-1}2^{n-1}$ é negativa. Por exemplo,

$$110_{(2)} = 1 \times (-2^2) + 1 \times 2^1 + 0 \times 2^0 = -4 + 2 + 0 = -2_{(10)}$$

Exercício 19 Quais os inteiros representados em 8-bits e complemento para 2 por $F2_{(h)}$, $4F_{(h)}$, $A3_{(h)}$, 10010000, 00000000, 10000000, $7F_{(h)}$?

Exercício 20 Represente em complemento para 2 e com 8 bits os números inteiros seguintes 25, -25, 41, 56, 19, -31, e -87.

Exercício 21 Suponha que a representação em complemento para dois com 4 dígitos de um inteiro é 1101. Como é a representação com:

- (i) 6 dígitos
- (ii) 8 dígitos
- (iii) n dígitos, com $n \geq 4$.

Exercício 22 Suponha agora que, em 4 dígitos, a representação em complemento para dois é 0101. Como responderia às alíneas anteriores? Em geral, dado um inteiro x representado em n dígitos como procederia para obter a sua representação com mais dígitos?

1.3 Adição e Subtracção de inteiros

1.3.1 Inteiros não negativos

Para adicionar dois inteiros não negativos, segue-se o algoritmo habitual de adição binária; se se tiver uma representação de m bits e a soma for superior a $2^m - 1$ (o maior inteiro positivo com m bits) diz-se que há transporte e o valor resultante está errado.

Por exemplo, existe transporte depois de efectuar a soma em 8-bits dos inteiros positivos 10010000 e 1111101 cujo resultado é 10001101.

Na subtracção de dois inteiros não negativos, se o valor da diferença for inferior a zero, diz-se que há transporte e o valor obtido está errado. Por exemplo: a diferença em 8-bits $10010000 - 1111101 = 10010011$ (resultado errado)

1.3.2 Adição e Subtração de inteiros em complemento para dois

Uma das vantagens da representação em complemento para dois é que o algoritmo para a adição para inteiros não negativos pode ser utilizado para quaisquer inteiros, desprezando-se o transporte que eventualmente exista nos bits mais significativos. No entanto, dado que trabalhamos com um número finito de dígitos, pode ocorrer um **overflow**, indicando que o resultado não é representável.

Se os inteiros são *não negativos* e a soma for superior a $2^{m-1} - 1$ diz-se que há **overflow**.

Por exemplo: a soma em 8-bits e complemento para 2 de 01111111 com 00000001 é 10000000 ou seja, a soma de +127 com +1 é -128(!).

Se os inteiros são *ambos negativos* há **overflow** também designado de **underflow**), se a soma for inferior a -2^{m-1} .

Por exemplo: a soma em 8-bits e complemento para 2 de 10000000 com 11111111 seria 01111111 ou seja, $-128 + (-1) = +127$ (!).

Se forem de sinais contrários não pode ocorrer **overflow** na adição.

A correcção de “desprezar o transporte”, se não houver **overflow**, é justificada a seguir. Em complemento para 2 com m bits, a representação dum número não negativo coincide com a que ele teria numa representação sem sinal e para a representação de um número negativo adiciona-se 2^m . Quando se adicionam dois números x e y em complemento para dois temos 3 situações:

1. Tanto x como y são positivos. A soma $x+y$ é um número positivo, cuja representação, se existir, é a mesma que para as representações sem sinal.
2. Um, por exemplo x , é positivo e o outro é negativo. Adicionando, na representação em complemento para dois, temos $x+y+2^m$. Se o valor de x é maior que o valor de y , então $x+y \geq 0$ e na adição em complemento para dois, isto é, $x+y+2^m$, haverá um transporte que podemos desprezar obtendo $x+y$, que é o resultado correcto. Caso contrário, não há transporte, pois $x+y+2^m$ é menor que 2^m . Como $x+y \leq 0$, $x+y+2^m$ é a representação correcta em complemento para 2.
3. Tanto x como y são negativos. Então na representação em complemento para dois a adição corresponde a $(x+2^m) + (y+2^m)$. Se não existir **overflow**, então $x+y \geq -2^{m-1}$, e então existe transporte que desprezando leva a $x+y+2^m$. Como $x+y$ é negativo, essa é uma representação correcta em complemento para dois.

A subtração binária de inteiros representados em complemento para 2 reduz-se à adição, se se negar o subtraendo. Considere a seguinte tabela:

	Exemplos
$(-x)-(-y) = (-x)+y$	10010000-11111101=10010000+00000011=10010011
$(-x)-y=(-x)+(-y)$	10010000-00000011=10010000+11111101=10001101
$y-(-x)=y+x$	00000011-10010000=00000011+ 01110000=01110011
$y-x=y+(-x)$	00000011-00010000=00000011+11110000=11110011

Exercício 23 Converta os inteiros seguintes em representações em 8-bits e complemento para 2 e efectue as adições. Indique em cada caso se ocorreu um **overflow**.

(i) $45 + 60$

(ii) $(-45) + 60$

(iii) $(-45) + (-80)$

(iv) $78 + 60$

(v) $(-78) + (-60)$

2 Representação em vírgula fixa

A representação posicional de inteiros pode ser generalizada para representar números racionais considerando-se potências negativas da base. Por exemplo, na base 10:

$$344.45 = 3 \times 10^2 + 4 \times 10^1 + 4 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$$

Podemos ainda escrever:

$$344.45 = 34445. \times 10^{-2} = (3 \times 10^4 + 4 \times 10^3 + 4 \times 10^2 + 4 \times 10^1 + 5 \times 10^0) \times 10^{-2}$$

Se se fixar a posição da vírgula (neste caso o factor é 10^{-2}), o número pode ser tratado como um inteiro. Assim as operações com números racionais podem ser feitas internamente como operações com inteiros, desde que os factores de ajuste sejam guardados para que o resultado final seja correctamente calculado.

Note-se em particular que as representações de inteiros em computadores estudada anteriormente são representações em vírgula fixa, com a vírgula à direita do bit menos significativo. No caso geral, os ajustes da posição da vírgula e o seu armazenamento é deixado a cargo do programador.

Contudo, actualmente, a representação em computadores dos números racionais é feita, geralmente, em vírgula flutuante.

3 Representação em vírgula flutuante

A representação de um número racional em vírgula flutuante, contrariamente à representação em vírgula fixa, é feita através de um par de inteiros que representam respectivamente a *mantissa* m e o *expoente* e de forma que para uma determinada *base* b , o seu valor é:

$$F = m \times b^e$$

Por exemplo, 673×10^{14} .

A designação *vírgula flutuante* resulta do facto de que a posição da vírgula depender do expoente e portanto não ser fixada previamente.

A notação mais usada para vírgula flutuante é a do **IEEE**¹. A base é a binária. Em *precisão simples* cada número é representado com 32 bits:

S	E (8 bits)	M (23 bits)
-----	--------------	---------------

O sinal da mantissa é representado pelo bit S , que por questões de eficiência é separado da representação do módulo da mantissa o qual é constituído pelos 23 bits mais à direita. O valor do módulo da mantissa é normalmente dado por $1.M_{(2)}$, isto é, 1 mais o valor de M considerado como número racional binário, com a vírgula à esquerda do bit mais significativo. Os oito bits restantes são interpretados como um inteiro positivo E e representam o expoente cujo valor é $E - 127$.

O valor representado é

$$F = (-1)^S (2^{E-127}) (1.M)_{(2)}$$

excepto se E for 00000000 então $F = (-1)^S (2^{-127}) (.M)_{(2)}$ e se M também é zero então $F = 0$.

Considere

$$1\ 10000111\ 101000000000000000000000$$

Neste caso $S = 1$ indica que o número é negativo; $10000111_{(2)} = 135_{(10)}$ logo o expoente é $135 - 127 = 8$; e o módulo da mantissa é

$$1 + 0.101_{(2)} = 1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 1 + (5/8) = 1 + 0.625 = 1.625$$

O valor representado é $-2^8 \times 1.625 = -416$.

Existem outras representações em vírgula flutuante **IEEE**, como a *precisão dupla* em que são usados 64 bits e a *quádrupla* em que são usados 128 bits.

Exercício 24 Determina qual o maior e o menor número que é representável em precisão simples **IEEE**.

Exercício 25 Indica o valor das seguintes representações em precisão simples **IEEE**:

(i) 0 01110101 01010100000000000000000000

(ii) 1 00101010 11100000000000000000000000

Exercício 26 Exprima o mais exactamente possível os seguintes números em precisão simples **IEEE**: 2.5, .0005, 2^{-40} e 256

4 Bibliografia

1. Alfred V. Aho e Jeffrey D. Ullman, *Foundations of Computer Science*, Computer Science Press, W. H. Freeman and Company, 1992. *Capítulo*: 4.11-12.

¹*Institute of Electrical and Electronics Engineers*