Princípios da Computação

Representação de Informação

Instituto Superior de Engenharia do Porto

Setembro 2012

Departamento de Engenharia Informática

REPRESENTAÇÃO DE INFORMAÇÃO

Coube ao matemático italiano Leonardo de Pisa a glória de ter trazido para a Europa a numeração indo-arábica que veio substituir o complicado sistema inventado pelos romanos. No entanto, a introdução dos numerais indo-árabes encontrou oposição do público, visto que estes símbolos dificultavam a leitura dos livros dos mercadores.

A introdução dos dez símbolos na Europa Ocidental foi lenta. O primeiro manuscrito francês onde são encontrados data de 1275.

O sistema de numeração Árabe é o sistema de numeração da civilização Europeia. Também é denominado por sistema hindu, indo-árabe ou decimal. Teve a sua raiz nas línguas que estiveram na origem do latim e do grego e dos povos primitivos que o habitaram. Foi introduzido na Europa no final da Idade Média, contudo, o seu uso só foi generalizado no séc. XIV.

O sistema de numeração árabe ou decimal, (ou de base 10), é o mais utilizado nos dias de hoje.

Para representar todos os números, emprega apenas 10 símbolos diferentes, os chamados algarismos árabes. Estes símbolos são: 1, 2, 3, 4, 5, 6, 7, 8, 9 e zero (ou cifra - 0).

1 2 3 4 5 6 7 8 9 0

O símbolo correspondente a um número qualquer compõe-se de vários algarismos dispostos, uns a seguir aos outros, correspondendo, os seus lugares, às diferentes ordens, a começar pela direita. Estes lugares denominam-se por casas: casa das unidades, casa das dezenas, ...

Cada algarismo é, também, valorizado segundo a posição que ocupa, indicando a ordem dessas unidades, segundo a posição em que está situado. Não havendo unidades de certa ordem, a posição é ocupada por um zero. Deste modo, um algarismo colocado à esquerda de outro indica unidades da ordem imediatamente superior; colocado à direita, indica unidades de ordem imediatamente inferior.

No quadro seguinte estão indicadas as várias posições, as classes e os grupos, segundo a nomenclatura correspondente ao **Sistema de Numeração Árabe** (ou decimal):

Grupos	Classes	Ordens
Triliões	Triliões	Dezenas de triliões
		Triliões
Biliões	Milhares de biliões	Centenas de milhares de biliões
		Dezenas de milhares de biliões
		Milhares de biliões
	Biliões	Centenas de biliões
		Dezenas de biliões
		Biliões
Milhões	Milhares de milhões	Centenas de milhares de milhões
		Dezenas de milhares de milhões
		Milhares de milhão
	Milhões	Centenas de milhões
		Dezenas de milhões
		Milhões
Unidades	Milhares	Centenas de milhares
		Dezenas de milhares
		Milhares
	Unidades	Centenas
		Dezenas
		Unidades

Vejamos a seguinte representação num ábaco, a título de curiosidade:

Nos números de mais quatro algarismos, separam-se as classes por pequenos intervalos (em grupos de três a três), para facilidade de leitura. As unidades das diferentes ordens estão relacionadas com a unidade simples segundo as potências de 10, o que dá o nome de decimal ao sistema e às unidades das diversas ordens: a

dezena vale 10 unidades, a centena 10^2 unidades, o milhar 10^3, o milhão 10^6, o bilião 10^{12}, o trilião 10^{18}, …

Deste modo um número cujos algarismos sejam a_n, a_{n-1}, a_{n-2},… a_2, a_1, a_0 (onde os índices exprimem a ordem correspondente à posição que ocupam) equivale à seguinte soma: $a_n \times 10^n + a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2} + \ldots + a_2 \times 10^2 + a_1 \times 10^1 + a_0 \times 10^0$ (chamada representação polinomial ou representação em base pesada).

Vejamos os seguintes exemplos:

$5 = 5 \times 10^0$;

$99 = 9 \times 10^1 + 9 \times 10^0$;

$709 = 7 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$;

$543827 = 5 \times 10^5 + 4 \times 10^4 + 3 \times 10^3 + 8 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$;

Podemos desta forma traduzir graficamente qualquer número representado foneticamente, escrevendo os algarismos que exprimem as unidades das diferentes ordens, da esquerda para a direita; e reciprocamente, para ler um número, divide-se este em classes, a partir da direita, e a seguir faz-se a leitura de cada grupo de classes, em separado, a começar pela esquerda.

Como por exemplo:

O número 5 lê-se da seguinte forma:

- Cinco unidades;

O número 99 lê-se:

- Noventa e nove unidades;

O número 709 lê-se:

- Setecentos e nove unidades;

O número 543827 lê-se da seguinte maneira:

- Quinhentos e quarenta e três mil oitocentos e vinte sete unidades;

O número 43758953426287365205 lê-se:

- Quarenta e três triliões, setecentos e cinquenta e oito mil e novecentos e cinquenta e três biliões, quatrocentos e vinte seis mil duzentos e oitenta e sete milhões, trezentos e sessenta e cinco mil duzentos e cinco unidades;

O sistema estende-se aos números decimais, com a criação das casas das décimas, centésimas, milésimas,..., ordenadas segundo a mesma convenção das casa dos números inteiros, a que servem de continuação para a direita, e o emprego de uma virgula para assinalar a casa das unidades.

A.1 SISTEMAS DE NUMERAÇÃO

Assim, os números podem ser representados em qualquer sistema de numeração. Os seres humanos usam normalmente um sistema de numeração baseado na base 10 (com 10 dígitos diferentes). Os computadores, pelo facto de só representarem dois valores (0, 1), os dígitos binários – também conhecidos por bits, da contracção do inglês *binary digit* – são máquinas binárias, e por isso trabalham em base 2.

Para compreender o que significa a base em que os números são representados num dado sistema de numeração, é necessário relembrar o significado da ordem dos dígitos.

A ordem de um dígito dentro de um número é dada pela posição que esse dígito ocupa no número: 0 é a ordem do dígito imediatamente à esquerda do ponto (vírgula) decimal, crescendo no sentido da esquerda, e decrescendo no sentido da direita.

Exemplo:

$1532.64_{(10)}$

1	**5**	**3**	**2**	**.**	**6**	**4**	**(10)**
Dígito 1	**Dígito 5**	**Dígito 3**	**Dígito 2**		**Dígito 6**	**Dígito 4**	**Base de representação**
Posição +3	**Posição +2**	**Posição +1**	**Posição 0**		**Posição -1**	**Posição -2**	

A base utilizada determina o número de dígitos que podem ser utilizados; por exemplo, base 10 utiliza 10 dígitos (**0** a **9**), base 2 utiliza 2 dígitos (**0** e **1**), base 5 utiliza 5 dígitos (**0** a **4**), base 16 utiliza 16 dígitos (**0** a **9**, e, **A** a **F**).

A tabela seguinte apresenta alguns dos sistemas de numeração mais utilizados. Por convenção, os símbolos dos dígitos nos sistemas de numeração cuja base é inferior a 10, são os primeiros dígitos correspondentes do sistema de numeração

decimal, enquanto que os sistemas de numeração cuja base é superior a 10 usam os símbolos do sistema de numeração decimal mais as primeiras letras do alfabeto.

Base	Sistema de Numeração	Dígitos
2	**Binário**	**{0,1}**
8	**Octal**	**{0,1,2,3,4,5,6,7}**
10	**Decimal**	**{0,1,2,3,4,5,6,7,8,9}**
16	**Hexadecimal**	**{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}**

No estudo dos computadores, os sistemas de numeração binário, octal e hexadecimal têm um interesse especial. Apresenta-se a seguir um exemplo de um número escrito em cada um destes sistemas de numeração.

$1011.101_{(2)}$	**Sistema de numeração Binário**
$372.46_{(8)}$	**Sistema de numeração Octal**
$C65F.83_{(16)}$	**Sistema de numeração Hexadecimal**

É costume acrescentar à frente do número um índice, em decimal, indicativo da base do número. Este índice é omitido quando a base se enquadra no contexto.

Numeração Binária

O sistema binário de computação já era conhecido na China cerca de 3000 a.C., de acordo com os manuscritos da época. Quarenta e seis séculos depois, Leibniz redescobre o sistema binário.

Este sistema de numeração binário é muito importante, na medida em que, modernamente, é de largo alcance por ser utilizado nas calculadoras electrónicas, computadores e nas estruturas que envolvem relações binárias. Este sistema pode ser chamado sistema de base dois, binário ou dual, o qual utiliza apenas dois algarismos, o **0** e o **1**, os quais nas estruturas dessas máquinas se fazem corresponder às situações de sim/não, ligado/desligado, aberto/fechado, contacto/interrupção, passagem/vedação, etc., uma vez que os circuitos digitais são constituídos por elementos dotados de dois estados distintos.

A cada um dos símbolos do sistema binário chama-se um «bit», Dígito binário = *binary digit.* Sistema de numeração que utiliza 2 dígitos: **0** (zero) e **1** (um).

O maior inconveniente da base dois é que a representação de cada número envolve muitos algarismos. Por exemplo, cem mil, que na base dez se representa por 5 algarismos, na base dois representam por 17 algarismos! Porém, este inconveniente é superado nas máquinas electrónicas pela velocidade.

Na base dois, um número imediatamente à esquerda de outro, representa, em relação a este, um número de unidades duas vezes maior. (..., 2^3, 2^2, 2^1, 2^0), em que 4 bits = 1 Nibble e 8 bits = 1 *Byte*.

Pode ser feito o agrupamento de bits em conjuntos de 8, 16, 32, 64, etc.

2 bits = 4 combinações possíveis: **00**; **01**; **10**; **11**

3 bits = 8 combinações possíveis: **000**; **001**; **010**; **011**; **100**; **101**; **110**; **111**

8 bits = 256 (2^8) combinações possíveis.

Em geral, o número de combinações possíveis é $\mathbf{2^n}$, sendo ***n*** o número de dígitos binários.

Múltiplos de byte

KByte (kilo): 1 KB = 2^{10} Bytes = 1 024 Bytes

MByte (mega): 1 MB = 2^{10} KB = 1024 KB = 2^{20} Bytes = 1 048 576 Bytes

GByte (giga): 1 GB = 2^{10} MB = 1024 MB = 2^{20} KB = 2^{30} Bytes = 1 073 741 824 Bytes

TByte (tera): 1 TB = 2^{10} GB = 1024 GB =2^{20} MB = 2^{30} KB = 2^{40} Bytes = 1 099 511 627 776 Bytes

PByte (peta):1 PB = 2^{10} TB = 1024 TB = 2^{20} GB = 2^{30} MB = 2^{40} KB = 2^{50} Bytes = 1 125 899 906 842 624 Bytes

EByte (eta): 1 EB = 2^{10} PB = 1024 PB = 2^{20} TB = 2^{30} GB = 2^{40} MB = 2^{50} KB = 2^{60} Bytes = 1 152 921 504 606 846 976 Bytes

ZByte (zetta): 1 ZB = 2^{10} EB = 1024 EB = 2^{20} PB =2^{30} TB = 2^{40} GB = 2^{50} MB = 2^{60} KB = 2^{70} Bytes = 1 180 591 620 717 411 303 424 Bytes

YByte (yotta): 1 YB = 2^{10} ZB = 1024 ZB = 2^{20} EB =2^{30} PB =2^{40} TB = 2^{50} GB = 2^{60} MB = 2^{70} KB = 2^{80} Bytes = 1 208 925 819 614 629 174 706 176 Bytes

Vejamos os seguintes exemplos, permitem ver como é que se representa um número decimal (numeração árabe) na base dois.

Notação decimal	Notação binária
0	$\mathbf{0 = 0 \times 2^0}$
1	$\mathbf{1 = 1 \times 2^0}$
2	$\mathbf{10 = 1 \times 2^1 + 0 \times 2^0}$
3	$\mathbf{11 = 1 \times 2^1 + 1 \times 2^0}$
4	$\mathbf{100 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0}$
5	$\mathbf{101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0}$
6	$\mathbf{110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0}$
7	$\mathbf{111 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0}$

Temos então que, para passar da notação binária para a notação decimal, o processo é o seguinte, por exemplo:

$$10011010_{(2)} = 1\times2^7 + 0\times2^6 + 0\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0 =$$

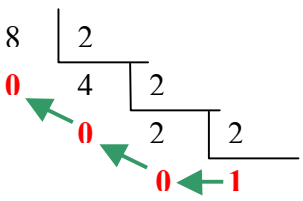
$$= 128 + 0 + 0 + 16 + 8 + 0 + 2 + 0 =$$

$$= 154_{(10)}$$

Agora para passar de decimal para binário o processo é um pouco mais complexo, mas não deixa de ser interessante, vejamos os seguintes exemplos:

$$8_{(10)} = ?_{(2)}$$

Façamos o seguinte raciocínio:



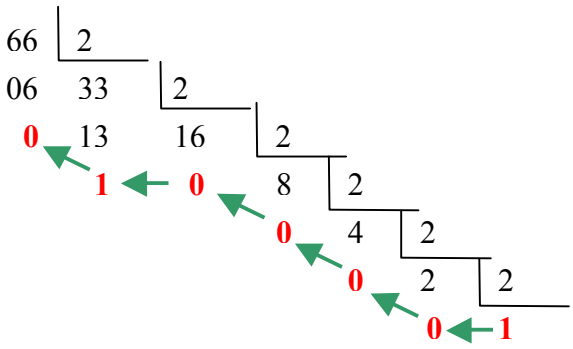
Podemos então concluir que:

$8_{(10)} = 1000_{(2)}$

$8 = 1\times2^3 + 0\times2^2 + 0\times2^1 + 0\times2^0$

Vejamos este outro exemplo:

$66_{(10)} = ?_{(2)}$



Podemos então concluir que:

$66_{(10)} = 1000010_{(2)}$

$66 = 1\times2^6 + 0\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

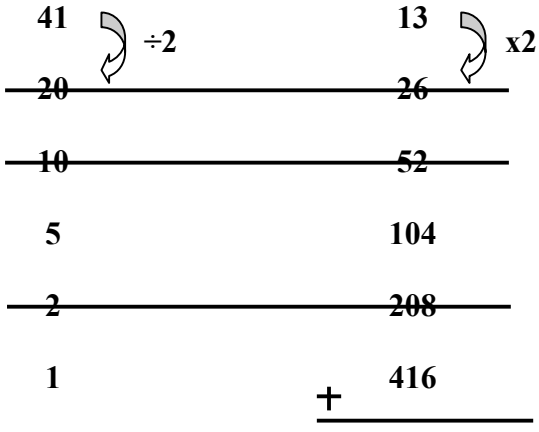
Um pequeno truque...

No sistema binário está o segredo de um pequeno truque com o qual podemos intrigar os nossos amigos mais ingénuos. Ele consiste num método para multiplicar dois números, através de uma técnica não mais difícil do que somar, multiplicar e dividir por dois.

Escrevem-se, um ao lado do outro, dois números, (na notação decimal). Em linhas consecutivas, multiplica-se o número da direita por dois e divide-se o da esquerda por dois, ignorando-se as fracções (metade de 11 deve ser considerado 5 e não 5.5). Então, riscam-se as linhas em que o número da esquerda é par e soma-se tudo o que sobrou na coluna da direita. O total será o produto procurado!

Vejamos o exemplo:

41 × 13 = ?



41 × 13 = 533

Somente quem estiver habituado ao sistema binário perceberá como funciona o truque...

Lembre-se que qualquer número se pode escrever no sistema binário, onde, qualquer número deverá ser igual à soma dos números escolhidos na série 1, 2, 4, 8, 16, 32,... (exemplo: 14 = 2 + 4 + 8).

Seja qual for a base numérica, cada dígito representa um valor absoluto. Por exemplo, os mesmos dígitos têm valores diferentes consoante a base em que o número é representado:

Decimal	Binário	Octal	Hexadecimal
$101_{(10)}$	$101_{(2)}$	$101_{(8)}$	$101_{(16)}$
$= 1\times10^2+0\times10^1+1\times10^0$	$= 1\times2^2+0\times2^1+1\times2^0$	$= 1\times8^2+0\times8^1+1\times8^0$	$= 1\times16^2+0\times16^1+1\times16^0$
$= 100 + 0 + 1$	$= 4 + 0 + 1$	$= 64 + 0 + 1$	$= 256 + 0 + 1$
$= 101_{(10)}$	$= 5_{(10)}$	$= 65_{(10)}$	$= 257_{(10)}$

Ou seja, cada dígito representa em absoluto, o seu valor multiplicado pelo peso da posição que ocupa no número. Na base decimal, o primeiro dígito é o das unidades (10^0), o segundo representa as dezenas (10^1), o terceiro representa as

centenas (10^2), etc. Na base binária, o princípio é o mesmo, excepto que os valores representados por cada dígito são calculados através de potências de 2^n, pois o número de valores possíveis num dígito é 2. O valor máximo representado no primeiro dígito é 2^0, no segundo 2^1, no terceiro 2^2, etc..

A.3 CONVERSÃO ENTRE BASES

A **conversão** de um número escrito na **base *b* para a base decimal** obtém-se multiplicando cada dígito pela base *b* elevada à ordem do dígito, e somando todos estes valores.

Exemplos:

$1532_{(6)}$ (base 6)

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

$1532.64_{(10)}$ (base 10)

$1\times10^3 + 5\times10^2 + 3\times10^1 + 2\times10^0 + 6\times10^{-1} + 4\times10^{-2} = 1532.64_{(10)}$

$1532_{(13)}$ (base 13)

$1\times13^3 + 5\times13^2 + 3\times13^1 + 2\times13^0 = 3083_{(10)}$

$110110.011_{(2)}$ (base 2)

$1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} = 54.375_{(10)}$

Na **conversão** de um número na **base decimal para uma base *b***, o processo mais directo é composto por 2 partes:

- **Divisão sucessiva da parte inteira** desse número pela respectiva base, sendo os restos obtidos com cada uma dessas divisões, os dígitos da base *b* (a começar com o menos significativo, i.e., mais junto ao ponto decimal) e os quocientes a usar na sucessão de divisões;
- **Multiplicação sucessiva da parte fraccionária** desse número pela respectiva base, sendo a parte inteira de cada um dos produtos obtidos, os dígitos da base *b* (a começar com o mais significativo, i.e., mais longe do ponto decimal), e a parte decimal a usar na sucessão de multiplicações.

Exemplo

$235.375_{(10)}$

235÷2 = 117 Resto = **1** /* *bit* menos significativo int*/

117÷2 = 58 Resto = **1**

58÷2 = 29 Resto = **0**

29÷2 = 14 Resto = **1**

14÷2 = 7 Resto = **0**

7÷2 = 3 Resto = **1**

3÷2 = 1 Resto = **1**

1÷2 = 0 Resto = **1** /* *bit* mais significativo int*/

0.375×2 = **0**.750 Parte int. = **0** /* *bit* mais significativo fraccionário*/

0.5×2 = **1**.0 Parte int. = **1** /* *bit* menos significativo fraccionário*/

$235.375_{(10)} = 11101011.011_{(2)}$

Outro processo de converter de uma base decimal para outra base utiliza **subtracções sucessivas**, mas apenas é utilizado na **conversão para a base binária**, e mesmo nesta para valores que não ultrapassam a ordem de grandeza dos milhares e normalmente apenas para inteiros.

A grande vantagem deste método é a sua rapidez de cálculo mental, sem ajuda de qualquer máquina de calcular, desde que se saiba de memória a **"tabuada" das potências de 2**:

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Ajuda também saber como se comportam as potências de 2 para expoentes com mais que um dígito.

Sabendo que $2^{10} = 1024$ (= **1K**, $\approx 10^3$), e que $2^{1x} = 2^{10+x} = 2^{10} \times 2^x = 2^x \times 1K$, é possível a partir daqui extrapolar não apenas todos os restantes valores entre 2^{10} e 2^{19},

como ainda ter uma noção da **ordem de grandeza de um valor binário com qualquer número de dígitos** (bits):

Exemplos

Tabela de potências de 2^{10} a 2^{19}

2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}
512K	256K	128K	64K	32K	16K	8K	4K	2K	1K

Tabela de potências de 2 com expoentes variando de 10 em 10

2^{80}	2^{70}	2^{60}	2^{50}	2^{40}	2^{30}	2^{20}	2^{10}
1Y	**1Z**	**1E**	**1P**	**1T**	**1G**	**1M**	**1K**
$\approx 10^{24}$	$\approx 10^{21}$	$\approx 10^{18}$	$\approx 10^{15}$	$\approx 10^{12}$	$\approx 10^{9}$	$\approx 10^{6}$	$\approx 10^{3}$

Com base nesta informação, é agora possível pôr em prática o método das subtracções sucessivas para converter um nº decimal num binário: 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 subtracção repete-se o processo até chegar ao resultado 0.

Exemplo

$1081.625_{(10)}$

$1081.625 - 2^{10} = 57.625$

$57.625 - 2^{5} = 24.625$

$25.625 - 2^{4} = 9.625$

$9.625 - 2^{3} = 1.625$

$1.625 - 2^{0} = 0.625$

$0.625 - 2^{-1} = 0.125$

$0.125 - 2^{-3} = 0$

1	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**.**	**1**	**0**	**1**	**(2)**
10	9	8	7	6	5	4	3	2	1	0		-1	-2	-3	**← posição**

Os processadores utilizam um determinado número de bits para representar um número. A quantidade de bits utilizados determina a gama de valores representáveis. Tal como qualquer outro sistema de numeração – onde a gama de valores representáveis com ***n*** dígitos é b^n – a mesma lógica aplica-se à representação de valores binários.

Sendo ***n*** o número de bits utilizados, a **gama de valores representáveis em binário, usando *n* bits é 2^n**.

Base hexadecimal

O sistema de numeração de **hexadecimal** (base 16) é frequentemente utilizado como forma alternativa de representação de valores binários, não apenas pela facilidade de conversão entre estas 2 bases, como ainda pela menor probabilidade de erro humano na leitura/escrita de números. Tal como referido anteriormente, são utilizados 16 dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Exemplos:

Converter $4312_{(10)}$ para hexadecimal

4312 ÷ 16 = 269 Resto = 8

269 ÷ 16 = 16 Resto = 13 (dígito D)

16 ÷ 16 = 1 Resto = 0

1 ÷ 16 = 0 Resto = 1

Logo, $4312_{(10)} = 10D8_{(16)}$

Converter $2AF3_{(16)}$ para decimal

$2 \times 16^3 + 10 \times 16^2 + 15 \times 16^1 + 3 \times 16^0 = 10995_{(10)}$

Logo, $2AF3_{(16)} = 10995_{(10)}$

$101_{(2)} = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 1 \times 1 + 0 \times 2 + 1 \times 4 = 6_{(10)}$

$34F7_{(16)} = 7 \times 16^0 + F \times 16^1 + 4 \times 16^2 + 3 \times 16^3 =$

$$= 7 \times 1 + 15 \times 16 + 4 \times 256 + 3 \times 4096 = 13559_{(10)}$$

Não é prático converter um número duma base que não a decimal, para outra base. No entanto observa-se que:

Existem determinadas bases numéricas, que tem a particularidade do seu dígito mais elevado usar um número fixo de dígitos binários completamente ocupados, ou seja, todos a 1. Por exemplo, $3_{(4)} = 11_{(2)}$, $7_{(8)} = 111_{(2)}$, ou $F_{(16)} = 1111_{(2)}$.

Assim, uma cadeia de dígitos binários pode ser subdividida em grupos de dígitos. A partir daí bastará fazer corresponder a cada grupo de dígitos binários o dígito na base específica. Assim, conhecendo apenas um conjunto limitado (e simples) de relações entre números em diferentes bases, é fácil e imediato fazer conversões.

Exemplos:

Binário – Octal	Binário – Hexadecimal
111 110 100 100 110 $_{(2)}$ = 7 6 4 4 6 $_{(8)}$	**111 1101 0010 0110 $_{(2)}$ = 7 D 2 6 $_{(16)}$**
111 $_{(2)}$→ 7 $_{(10)}$	0111 $_{(2)}$ → 7 $_{(10)}$ → 7 $_{(16)}$
110 $_{(2)}$ → 6 $_{(10)}$	1101 $_{(2)}$ → 14$_{(10)}$ → D $_{(16)}$
100 $_{(2)}$ → 4 $_{(10)}$	0010 $_{(2)}$ → 2 $_{(10)}$ → 2 $_{(16)}$
100 $_{(2)}$ → 4 $_{(10)}$	0110 $_{(2)}$ → 6 $_{(10)}$ → 6 $_{(16)}$
110 $_{(2)}$ → 6 $_{(10)}$	**Conversão entre base binária e hexadecimal por agrupamento 4 a 4 de dígitos binários.**
Conversão entre base binária e octal por agrupamento 3 a 3 de dígitos binários.	

Usando os resultados das conversões de binário para octal ou hexadecimal, também o procedimento de conversão de binário para decimal se torna menos exaustivo, uma vez que se pode usar o valor em octal ou hexadecimal para a conversão.

Exemplos completos:

- Converter o número $101111011_{(2)}$ para base hexadecimal, octal e decimal.

Binário → Octal	Binário → Hexadecimal	Binário → Decimal

$101\ 111\ 011_{(2)}$ = **$5\ 7\ 3_{(8)}$**	**$1\ 0111\ 1011_{(2)}$ =** **$1\ 7\ B_{(16)}$**	**$101111011_{(2)}$ =** **$573_{(8)}$=** **$3\times8^0+7\times8^1+5\times8^2 = 379_{(10)}$**

- Converter o número $761_{(8)}$ para base binária, hexadecimal e decimal.

Octal → Binário	Octal → Hexadecimal	Octal → Decimal
$7\ 6\ 1_{(8)}$ = **$= 111\ 110\ 001_{(2)}$**	**$761_{(8)}$ =** **$= 1\ 1111\ 0001_{(2)}$=** **$= 1\ F\ 1_{(16)}$**	**$761_{(8)}$ =** **$= 1 \times 8^0 + 6 \times 8^1 + 7 \times 8^2 =$** **$497_{(10)}$**

A motivação para usar hexadecimal é a facilidade com que se converte entre esta base e binário, ou para qualquer base que seja do tipo 2^n. Cada dígito hexadecimal representa um valor entre 0 e 15; cada conjunto de 4 bits representa também um valor no mesmo intervalo. Pode-se então aproveitar esta característica nos 2 tipos de conversão: de binário, agrupando os bits de 4 em 4 a partir do ponto decimal, e convertendo-os; para binário, convertendo cada dígito hexadecimal em 4 bits.

O mesmo é verdade para qualquer base do tipo 2^n.

Exemplos:

2 A F (hexadecimal)
0010 1010 1111 (binário)
$2AF_{(16)}$ = $001010101111_{(2)}$

1101 0101 1011 (binário)
D 5 B (hexadecimal)
$110101011011_{(2)}$ = $D5B_{(16)}$ (é comum representar como 0xD5B, ou 0xd5b)

A.4 ARITMÉTICA MATEMÁTICA

As operações aritméticas, seguem os mesmos procedimentos seja qual for a base numérica em questão. No entanto analisa-se de seguida os procedimentos aplicados nas operações:

- Adição;

- Subtracção;
- Multiplicação e Divisão.

Adição

Nota: Os valores a vermelho por cima dos operadores, representa o "que vai" ou em terminologia informática o *"carry"*.

Decimal	Binária	Octal	Hexadecimal
1 9 + 1 1 0	1 1 + 1 1 0	1 7 + 1 1 0	1 F + 1 1 0
1 1 1 2 9 2 + 8 7 3 2 1 6 5	1 1 1 1 0 1 + 1 0 1 1 1 1 1 1 0 0	1 1 6 7 + 3 2 1 2 1	1 4 F 1 2 + 9 F 4FB1

Subtracção

Decimal	Binária	Octal	Hexadecimal
1 1 2 – 5 0 7	1 1 0 – 1 0 1	1 1 2 – 5 0 5	1 1 B – C 0 F
1 5 – 1 2 – 0 7	1 1 1 1 0 – 1 0 0 1 – 0 0 1 1	1 1 5 6 – 3 0 2 – 2 2 4	1 4 F 2 – D A 9 F – D 5 A D

Multiplicação e Divisão

Os procedimentos relativos à operação de multiplicação são os mesmos, qualquer que seja a base numérica, mas como já havia sido referido, a tabuada que se

conhece é a tabuada decimal, pelo que não é simples realizar as operações de multiplicação e divisão.

Decimal	Binária	Octal	Hexadecimal
1		1	2
1 2	1 0	1 2	1 B
× 5	× 1	× 5	× 3
6 0	1 0	6 2	5 1

O mesmo princípio se aplica para a operação divisão, uma vez que o algoritmo se baseia na tabuada.

Decimal	Binária	Octal	Hexadecimal
12 | 2	100 | 10	12 | 4	A12 | 6 ____
0 6	00 10	20 2.4	41 1AD
	0	0	52
			4

A.5 NÚMEROS NEGATIVOS

A representação de números negativos coloca problemas em sistemas informáticos, pois os valores possíveis de representar são binários, o que implica que o sinal "–" não esteja incluído. Daí ser necessário definir regras de representação.

A solução está em atribuir a um dos dígitos do número, informação acerca do sinal. Ou seja, definir um "bit de sinal". Se tal dígito estiver a 0 (zero) o valor será positivo, se estiver a 1 o valor será negativo. Por norma, o bit de sinal é o bit mais significativo (mais à esquerda) do número.

Número positivo: **0**01100

Número negativo: **1**01100

No entanto isto levanta uma questão interessante: e o zero, também pode ser negativo? Se considerarmos que o número é representado pelo computador usando 4 bits, teríamos:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

Tabela 1: Representação de inteiros em Sinal em Grandeza com 4 bits

Portanto, para além de ser incorrecto representar-se o valor -0, existe uma combinação de dígitos que poderia ser usada para um outro valor válido.

Uma outra representação é o complemento para 1 que consiste em inverter todos os bits (trocar os 0's para 1's e os 1's para 0's) de um número positivo para se obter a respectiva representação do valor em negativo:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

Tabela 2: Representação de inteiros em Complemento para 1 com 4 bits

Mas como é óbvio inverter os bits da representação de **0** obtém-se, também neste tipo de representação o valor **-0** (como se pode verificar na tabela).

A solução está em usar a representação em complemento para 2. O complemento para 2 dum número encontra-se usando o seguinte algoritmo:

1. Complementar o valor (trocar os 0's para 1's e os 1's para 0's);
2. Adicionar 1 ao valor encontrado no ponto anterior.

Exemplo:

$5_{(10)} = 0101_{(2)}$ — Complemento para 1

1010

+1

1011 $\qquad 1011_{(2)} = -5_{(10)}$

Assim a representação em complemento para 2, e que é a representação corrente em informática, será:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Ou seja, existe mais um valor negativo que nas representações em Sinal e Grandeza e em Complemento para 1, e que substitui o **-0**. Aplica-se o mesmo procedimento quer se trate de 4, 8, 16, 32 ou outro qualquer número de bits.

Como já foi exposto, os computadores lidam com números positivos e números negativos, sendo necessário encontrar uma representação para números com sinal negativo.

Existe uma grande variedade de opções, das quais apenas se destacam 4, sendo apenas 3 as actualmente usadas para representar valores negativos:

• **Sinal e grandeza** (S+G)

• **Complemento para 1**

• **Complemento para 2**

• **Notação em excesso** (ou *biased*)

Como o próprio nome indica, a representação **sinal e grandeza** utiliza um bit para representar o sinal, o bit mais à esquerda: **0** para indicar um valor positivo, **1** para indicar um valor negativo.

Na representação em **complemento para 1** invertem-se todos os bits de um número para representar o seu complementar: assim se converte um valor positivo para um negativo, e vice-versa. Quando o bit mais à esquerda é **0**, esse valor é positivo; se for **1**, então é negativo.

Exemplo:

$\mathbf{100_{(10)}} = 01100100_{(2)}$ (com 8 bits)

Invertendo todos os bits:

$10011011_{(2)} = \mathbf{-\ 100_{(10)}}$

O problema desta representação é que existem 2 padrões de bits para o **0**. Nomeadamente $0_{(10)} = 00000000_{(2)} = 11111111_{(2)}$. A solução encontrada consiste em representar os números em **complemento para 2.** Para determinar o negativo de um número negam-se todos os seus bits e soma-se uma unidade.

Exemplo:

$\mathbf{100_{(10)}} = 01100100_{(2)}$ (com 8 bits)

Invertendo todos os bits:

$10011011_{(2)}$

Somando uma unidade:

$10011011_{(2)} + 1_{(2)} = 10011100_{(2)} = \mathbf{-\ 100_{(10)}}$

A representação em complemento para 2 tem as seguintes características:

- O bit da esquerda indica o sinal;
- O processo indicado no parágrafo anterior serve para converter um número de positivo para negativo e de negativo para positivo;
- O 0 tem uma representação única: todos os bits a 0;
- A gama de valores que é possível representar com *n* bits é -2^{n-1} ... $2^{n-1}-1$.

Exemplo

Qual o número representado por **$1110\ 0100_{(2)}$** (com 8 bits)?

Como o bit da esquerda é 1 este número é negativo.

Invertendo todos os bits:

$00011011_{(2)}$

Somando uma unidade :

$0001\ 1011_{(2)} + 1_{(2)} = 0001\ 1100_{(2)} = 28_{(10)}$

Logo:

$1110\ 0100_{(2)} = \mathbf{-28_{(10)}}$

Como é que se converte um número representado em complemento para 2 com *n* bits, para um número representado com mais bits?

Resposta: basta fazer a extensão do sinal! Se o número é positivo acrescenta-se 0's à esquerda, se o número é negativo acrescenta-se 1's à esquerda.

Exemplo:

Representar os seguintes números (de 8 bits) com 16 bits:

$01101010_{(2)}$

Positivo, logo:

$00000000\ 01101010_{(2)}$

$11011110_{(2)}$

Negativo, logo:

$11111111\ 11011110_{(2)}$

A multiplicação e a divisão têm algoritmos algo complexos como pudemos verificar anteriormente. No entanto, no caso da **multiplicação e a divisão por potências de 2** (2, 4 ,8, 16, 32, 64, 128,…) realizam-se efectuando deslocamentos de bits à esquerda ou à direita, respectivamente.

Fazer o deslocamento à esquerda uma vez – num número binário – corresponde a multiplicar por 2, duas vezes corresponde a multiplicar por 4 (= 2^2), 3 vezes corresponde a multiplicar por 8 (= 2^3), e assim sucessivamente. O mesmo se aplica à divisão com o deslocamento à direita.

Exemplo:

Dividir $1100\ 1010_{(2)}$ (= $202_{(10)}$) **por 2**:

Deslocar à direita 1 vez:

$0001100101_{(2)} = 101_{(10)}$

No entanto, se fosse para **Dividir** $1100\ 1010_{(2)}$ (= $202_{(10)}$) **por 4**:

Deslocar à direita 2 vezes:

$000110010.1_{(2)} = 50.5_{(10)}$

Multiplicar $000001010_{(2)}$ (= $10_{(10)}$) por 2.

Deslocar à esquerda 1 vez:

$000010100_{(2)} = 20_{(10)}$

Se fosse para **Multiplicar** $000001010_{(2)}$ (= $10_{(10)}$) por 8.

Deslocar à esquerda 3 vezes:

$001010000_{(2)} = 80_{(10)}$

Esta regra também se aplica aos números em complemento para 2 desde que se mantenha o sinal. A última notação referida no início – **notação em excesso** – tem uma vantagem sobre qualquer outra referida anteriormente: a representação numérica dos valores em binário, quer digam respeito a valores com ou sem sinal, tem o mesmo comportamento na relação entre eles.

Por outras palavras, o valor em binário com todos os bits a 0 representa o menor valor inteiro, quer este tenha sinal ou não, e o mesmo se aplica ao maior valor em binário, i.e., com todos os bits a 1: representa o maior inteiro, com ou sem sinal.

Exemplo

Binário (8 bits)	Sinal + Grandeza	Compl p/ 1	Compl p/ 2	Excesso (128)
$0000\ 0001_{(2)}$	**+1**	**+1**	**+1**	**-127**
…	**…**	**…**	**…**	**…**
$1000\ 0001_{(2)}$	**-1**	**-126**	**-127**	**+1**
…	**…**	**…**	**…**	**…**
$1111\ 1110_{(2)}$	**-126**	**-1**	**-2**	**+126**

Como o próprio nome sugere, esta codificação de um inteiro (negativo ou positivo) em binário com *n* bits é feita sempre em excesso (de 2^{n-1} ou 2^{n-1}-1).

Neste exemplo com 8 bits, o valor $+1_{(10)}$ é representado em binário, em notação por excesso de 2^{n-1}, pelo valor $(+1_{(10)} + \text{excesso}) = (+1_{(10)} + 128_{(10)}) = (0000\ 0001_{(2)} + 1000\ 0000_{(2)}) = 1000\ 0001_{(2)}$.

A tabela que a seguir se apresenta, representando todas as combinações possíveis com 4 bits, ilustra de modo mais completo as diferenças entre estes 4 modos (+1 variante) de representar inteiros com sinal.

Binário (4 bits)	Sinal + Grandeza	Compl p/ 1	Compl p/ 2	Excesso 7	Excesso 8
0111	**7**	**7**	**7**	**0**	**-1**
0110	**6**	**6**	**6**	**-1**	**-2**
0101	**5**	**5**	**5**	**-2**	**-3**
0100	**4**	**4**	**4**	**-3**	**-4**
0011	**3**	**3**	**3**	**-4**	**-5**
0010	**2**	**2**	**2**	**-5**	**-6**
0001	**1**	**1**	**1**	**-6**	**-7**
0000	**0**	**0**	**0**	**-7**	**-8**

1000	**-0**	**-7**	**-8**	**1**	**0**
1001	**-1**	**-6**	**-7**	**2**	**1**
1010	**-2**	**-5**	**-6**	**3**	**2**
1011	**-3**	**-4**	**-5**	**4**	**3**
1100	**-4**	**-3**	**-4**	**5**	**4**
1101	**-5**	**-2**	**-3**	**6**	**5**
1110	**-6**	**-1**	**-2**	**7**	**6**
1111	**-7**	**-0**	**-1**	**8**	**7**

A.6 CODIFICAÇÃO DE DADOS

Código ASCII

ASCII (acrónimo para *American Standard Code for Information Interchange*) é um conjunto de códigos para o computador, que permite representar diversos tipos de caracteres (números, letras, pontuação e outros). ASCII é uma padronização da indústria de computadores, onde cada carácter é manipulado na memória discos etc., sob forma de código binário. O código ASCII é formado por todas as combinações possíveis de 7 bits, sendo que existem várias extensões que abrangem 8 ou mais bits. A tabela abaixo representa a tabela ASCII de 7 bits.

Exemplo:

Letra B = 100 0010 (ou 0100 0010 para formar um byte completo)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Código UNICODE

Também é utilizado para representar texto, mas permite representar todos os caracteres (incluindo os orientais). Normalmente são utilizados 16 bits (2 bytes) para representar cada carácter.