

**Universidade do Minho**  
Departamento de Informática

# **Representação Binária de Números**

***Notas de estudo***

*Alberto José Proença*  
*01-Mar-04*



## Parte A: Sistemas de numeração e representação de inteiros

### A.1 Sistemas de numeração

#### A.2 Conversão entre bases

#### A.3 Base hexadecimal

#### A.4 Números negativos

### A.1 Sistemas de numeração

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**<sub>10</sub>

Dígito 4 - ordem -2

Dígito 6 - ordem -1

Dígito 2 - **ordem 0**

Dígito 3 - ordem +1

Dígito 5 - ordem +2

Dígito 1 - ordem +3

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.2 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**<sub>6</sub> (base 6)

$$1 \cdot 6^3 + 5 \cdot 6^2 + 3 \cdot 6^1 + 2 \cdot 6^0 = 416_{10}$$

**1532.64**<sub>10</sub> (base 10)

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

**1532**<sub>13</sub> (base 13)

$$1 \cdot 13^3 + 5 \cdot 13^2 + 3 \cdot 13^1 + 2 \cdot 13^0 = 3083_{10}$$

**110110.011**<sub>2</sub> (base 2)

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

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 junto ao ponto decimal), e a parte decimal a usar na sucessão de multiplicações.

#### Exemplo

**235.375**<sub>10</sub>

235/2 = 117	Resto = 1	<i>/* bit menos significativo int*/</i>
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	<i>/* bit mais significativo int*/</i>
0.375*2 = 0.750	P. int. = 0	<i>/* bit mais significativo frac*/</i>
0.75*2 = 1.5	P. int. = 1	
0.5*2 = 1.0	P. int. = 1	<i>/* bit menos significativo frac*/</i>

**235.375**<sub>10</sub> = **11101011.011**<sub>2</sub>

Outro processo de converter de uma base decimal para outra base utiliza **subtrações 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 cor a “**tabuada**” das potências de 2:

<b>2<sup>0</sup></b>	<b>2<sup>1</sup></b>	<b>2<sup>2</sup></b>	<b>2<sup>3</sup></b>	<b>2<sup>4</sup></b>	<b>2<sup>5</sup></b>	<b>2<sup>6</sup></b>	<b>2<sup>7</sup></b>	<b>2<sup>8</sup></b>	<b>2<sup>9</sup></b>
<b>1</b>	<b>2</b>	<b>4</b>	<b>8</b>	<b>16</b>	<b>32</b>	<b>64</b>	<b>128</b>	<b>256</b>	<b>512</b>

Ajuda também saber como se comportam as potências de 2 para expoentes com mais que um dígito. Sabendo que  $2^{10} = 1024$  (= **1K**,  $\sim 10^3$ ), e que  $2^{1n} = 2^n * 2^{10} = 2^n * 1K$ , 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)**<sup>1</sup>:

#### Exemplos

##### Tabela de potências de 2<sup>10</sup> a 2<sup>19</sup>

<b>2<sup>10</sup></b>	<b>2<sup>11</sup></b>	<b>2<sup>12</sup></b>	<b>2<sup>13</sup></b>	<b>2<sup>14</sup></b>	<b>2<sup>15</sup></b>	<b>2<sup>16</sup></b>	<b>2<sup>17</sup></b>	<b>2<sup>18</sup></b>	<b>2<sup>19</sup></b>
<b>1K</b>	<b>2K</b>	<b>4K</b>	<b>8K</b>	<b>16K</b>	<b>32K</b>	<b>64K</b>	<b>128K</b>	<b>256K</b>	<b>512K</b>

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

<b>2<sup>10</sup></b>	<b>2<sup>20</sup></b>	<b>2<sup>30</sup></b>	<b>2<sup>40</sup></b>	<b>2<sup>50</sup></b>	<b>2<sup>60</sup></b>	<b>2<sup>70</sup></b>	<b>2<sup>80</sup></b>	<b>2<sup>90</sup></b>	<b>2<sup>x0</sup></b>
<b>1Kilo</b>	<b>1Mega</b>	<b>1Giga</b>	<b>1Tera</b>	<b>1Peta</b>	<b>1Exa</b>	<b>1Zeta</b>	<b>1Yota</b>	<b>?</b>	
<b>~10<sup>3</sup></b>	<b>~10<sup>6</sup></b>	<b>~10<sup>9</sup></b>	<b>~10<sup>12</sup></b>	<b>~10<sup>15</sup></b>	<b>~10<sup>18</sup></b>	<b>~10<sup>21</sup></b>	<b>~10<sup>24</sup></b>	<b>~10<sup>27</sup></b>	<b>~10<sup>3*x</sup></b>

<sup>1</sup> Refira-se a título de curiosidade, que a nomenclatura dos grandes números, conforme proposta em 1948 pelo *Sistema Internacional de Unidades* e transporta para a Norma Portuguesa NP-18, segue a regra N dada pela expressão "10 elevado a 6N = (N) ilião". Daqui os seguintes termos (correspondência): milhão (10<sup>6</sup>), **bilhão** (10<sup>12</sup>), **trilião** (10<sup>18</sup>), **quadrilião** (10<sup>24</sup>), **quintilião** (10<sup>30</sup>), **sextilião** (10<sup>36</sup>), **septilião** (10<sup>42</sup>), **octilião** (10<sup>48</sup>), **nonilião** (10<sup>54</sup>), **decilião** (10<sup>60</sup>), ...

Com base nesta informação, é agora possível pôr em prática o método das subtrações 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 subtração repete-se o processo até chegar ao resultado 0.

**Exemplo****1081.625<sub>10</sub>**

$$\begin{aligned}
 1081.625 - 2^{10} &= 57.625 \\
 57.625 - 2^5 &= 24.625 \\
 25.625 - 2^4 &= 9.625 \\
 9.625 - 2^3 &= 1.625 \\
 1.625 - 2^0 &= 0.625 \\
 0.625 - 2^{-1} &= 0.125 \\
 0.125 - 2^{-3} &= 0
 \end{aligned}$$

**1 0 0 0 0 1 1 1 0 0 1 . 1 0 1<sub>2</sub>**

10 9 8 7 6 5 4 3 2 1 0 -1 -2 -3 &lt; ordem

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  $2^n$** .

**A.3 Base hexadecimal**

O sistema de numeração de **base hexadecimal** (16) é frequentemente utilizada 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, ..., 9, A, B, C, D, E, F.

**Exemplos****4312<sub>10</sub> em 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<sub>10</sub> = 10D8<sub>16</sub>**2AF3<sub>16</sub> em decimal**

$$2 * 16^3 + 10 * 16^2 + 15 * 16^1 + 3 * 16^0 = 10995_{10}$$

A motivação para usar hexadecimal é a facilidade com que se converte entre esta base e binário. 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.

### Exemplos

2    A    F    (hexadecimal)  
0010 1010 1111 (binário)

$2AF_{16} = 001010101111_2$

1101 0101 1011 (binário)  
D    5    B    (hexadecimal)

$110101011011_2 = D5B_{16}$  (também comum representar como  $0xD5B$ , ou ainda  $0xd5b$ )

## A.4 Números negativos

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 amplitude/magnitude** (S+M)
- **complemento para 1**
- **complemento para 2**
- **notação em excesso** (ou *biased*)

Como o próprio nome indica, a representação **sinal e amplitude** 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

$100_{10} = 01100100_2$  (com 8 bits)

Invertendo todos os bits:

$10011011_2 = -100_{10}$

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

$100_{10} = 01100100_2$  (com 8 bits)

Invertendo todos os bits:

$10011011_2$

Somando uma unidade :

$10011011_2 + 1 = 10011100_2 = -100_{10}$

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

**Exemplo**

Qual o número representado por  $11100100_2$  (com 8 bits)?

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

Invertendo todos os bits:

$$00011011_2$$

Somando uma unidade :

$$00011011_2 + 1 = 00011100_2 = 28_{10}$$

Logo:

$$11100100_2 = -28_{10}$$

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<sub>2</sub>**

Positivo, logo:

$$00000000\ 01101010_2$$

**11011110<sub>2</sub>**

Negativo, logo:

$$11111111\ 11011110_2$$

A multiplicação e a divisão têm algoritmos algo complexos que ultrapassam o âmbito destas notas de estudo. No entanto a **multiplicação e a divisão por potências de 2** realizam-se efectuando deslocamentos de bits à direita ou à esquerda, 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**  $11001010_2$  ( $= 20_{10}$ ) por 4:

Deslocar à direita 2 vezes:

$$000110010.1_2 = 50.5_{10}$$

**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 + Ampl	Compl p/ 1	Compl p/ 2	Excesso (128)
0000 0001 <sub>2</sub>	+1	+1	+1	-127
...				
1000 0001 <sub>2</sub>	-1	-126	-127	+1
...				
1111 1110 <sub>2</sub>	-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 + Ampl	Compl p/ 1	Compl p/ 2	Excesso (7)	Excesso (8)
0000	0	0	0	-7	-8
0001	1	1	1	-6	-7
0010	2	2	2	-5	-6
0011	3	3	3	-4	-5
0100	4	4	4	-3	-4
0101	5	5	5	-2	-3
0110	6	6	6	-1	-2
0111	7	7	7	0	-1
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



## Parte B: Representação de reais em vírgula flutuante

- B.1 Notação científica
- B.2 Normalização na representação
- B.3 Intervalo e precisão de valores representáveis
- B.4 Formato binário dum valor em fp
- B.5 O bit escondido
- B.6 A norma IEEE 754 para valores em fp

### B.1 Notação científica

A representação de um valor infinito de valores usando uma máquina finita vai obrigar a assumir um conjunto de compromissos, os quais, no caso dos reais, irão afectar não só a gama de valores representáveis, como ainda a sua precisão. A utilização da notação científica, do tipo:

$$\text{Valor} = (-1)^S * \text{Mantissa} * \text{Radix}^{\text{Exp}}$$

é ainda aquela que permite obter a melhor representação de um n.º real em vírgula flutuante (ou *fp* na terminologia inglesa) com um n.º limitado de dígitos. O valor do radix é de 10 na representação decimal, e pode ser 2 ou uma potência de 2 na representação interna num computador. A IBM usava nos seus *mainframes* um radix de 16, pois permitia-lhe aumentar o intervalo de representação de valores; contudo os problemas que tiveram com esta representação deram mais força à utilização do valor 2 como radix.

### B.2 Normalização na representação

A notação científica permite que um mesmo n.º possa ser representado de várias maneiras com os mesmos dígitos (por ex., 43.789E+12, .43789E14, 43789E+09). Para facilitar a sua representação - omitindo a necessidade de representar o ponto/vírgula decimal - impõe-se a adopção de uma norma de representação, e diz-se que um dado n.º fp está normalizado quando cumpre essa norma. Alguns autores consideram que um n.º está **normalizado** quando a mantissa (ou parte fraccionária, **F**) se encontra no intervalo **[Radix, 1]**. Por outras palavras, existe sempre um dígito diferente de 0 à esquerda do ponto decimal.

Num exemplo em decimal com 7 algarismos na representação de fp (5 para a mantissa e 2 para o expoente), o intervalo de representação dum fp normalizado, seria em valor absoluto [1.0000E-99, 9.9999E+99]. Existe aqui um certo desperdício na representação de fp usando 7 algarismos, pois fica excluído todo o intervalo [0.0001E-99, 1.0000E-99]. Para se poder otimizar a utilização dos dígitos na representação de fp, aceitando a representação de valores menores que o menor valor normalizado, mas com o menor valor possível do expoente, se designa esta representação de **desnormalizada**.

Todas as restantes representações designam-se por **não normalizadas**.

### B.3 Intervalo e precisão de valores representáveis

Pretende-se sempre com qualquer codificação obter o maior intervalo de representação possível e simultaneamente a melhor precisão (relacionada com a distância entre 2 valores consecutivos). Existindo um n.º limitado de dígitos para a representação de ambos os valores - **F** e **Exp** - há que ter consciência das consequências de se aumentarem ou diminuírem cada um deles.

O intervalo de valores representáveis depende essencialmente do **Exp**, enquanto a precisão vai depender do número de dígitos que for alocado para a parte fraccionária, **F**. Numa representação em binário, a dimensão mínima a usar para fp (que será sempre um múltiplo da dimensão da célula) deverá ser pelo menos 32. Se fosse 16, 1 bit seria para o sinal, e os restantes 15 seriam insuficientes mesmo para representar apenas a parte fraccionária (daria uma precisão de 1 em cerca de 32 000...).

Usando **32 bits** para representação mínima de fp, torna-se necessário encontrar um valor equilibrado para a parte fraccionária e para o expoente. Esse valor é **8 para o expoente** - permite representar uma gama da ordem de grandeza dos  $10^{39}$  - e pelo menos **23 para a parte fraccionária** - permite uma precisão equivalente a 7 algarismos decimais.

#### B.4 Formato binário dum valor em fp

Existem 3 campos a representar nos 32 bits dum valor em fp: o sinal (1 bit), a parte fraccionária (23 bits) e o expoente (8 bits). Para se efectuar qualquer operação aritmética estes 3 campos terão de ser identificados e separados para terem um tratamento distinto na unidade que processa os valores em fp. A ordem da sua representação (da esquerda para a direita) segue uma lógica:

- **sinal, S:** ficando mais à esquerda, permite usar o mesmo *hardware* (que trabalha com valores inteiros) para testar o sinal de um valor em fp;
- **expoente, E:** ficando logo a seguir vai permitir fazer comparações quanto à grandeza relativa entre valores absolutos em fp, sem necessidade de separar os 3 campos: basta comparar os valores como se de valores meramente binários se tratassem;
- **parte fraccionária, F:** é o campo mais à direita.

#### B.5 O bit escondido

Um valor normalizado tem sempre um dígito diferente de zero à esquerda do ponto decimal. Se o sistema de numeração é decimal, esse dígito pode ser um de entre 9 possíveis; se o sistema de numeração é binário, esse dígito só pode ser um. Assim, **e apenas na representação binária normalizada**, esse dígito à esquerda do ponto decimal toma sempre o mesmo valor, e é um desperdício do espaço de memória estar a representá-lo fisicamente. Ele apenas se torna necessário para efectuar as operações, permanecendo **escondido** durante a sua representação. Ganha-se um bit para melhorar a precisão, permitindo passar para 24 o n.º de bits da parte fraccionária (numa representação com 32 bits).

#### B.6 A norma IEEE 754 para valores em fp

A representação de valores em fp usando 32 bits e com o formato definido anteriormente permite ainda várias combinações para representar o mesmo valor. Por outro lado, não ficou ainda definido como representar os valores desnormalizados, bem como a representação de valores externos ao intervalo permitido com a notação normalizada.

A norma IEEE 754 define com clareza estas imprecisões, permitindo uma maior compatibilidade ao nível dos dados no porte de aplicações entre sistemas que adoptem a mesma norma. De momento todos os microprocessadores disponíveis comercialmente com unidades de fp suportam a norma IEEE 754 no que diz respeito aos valores de 32 bits. Aspectos relevantes na norma IEEE 754:

- **representação do sinal e parte fraccionária:** segue o formato definido anteriormente, sendo a parte fraccionária representada sempre em valor absoluto, e considerando o bit escondido na representação normalizada;
- **representação do expoente:** para permitir a comparação de valores em fp sem separação dos campos, a codificação do expoente deveria ser tal que os valores menores de expoente (os negativos) tivessem uma representação binária menor que os valores positivos (e maiores); as codificações usando complemento para 1 ou 2, ou ainda a representação usando sinal+amplitude, não possuem este comportamento, i.e., os valores negativos têm o bit mais significativo (à esquerda) igual a 1, o que os torna, como números binários, maiores que os números positivos; a notação que satisfaz este requisito é uma notação por excesso, na qual se faz um deslocamento na gama de valores decimais correspondentes ao intervalo de representação de n bits, de 0 a  $2^{n-1}$ , de modo a que o 0 decimal passe a ser representado não por uma representação binária com tudo a zero, mas por um valor no meio da tabela; usando 8 bits por exemplo, esta notação permitiria representar o 0 pelo valor 127 ou 128; a norma IEEE adoptou o primeiro destes 2 valores, pelo que a representação do expoente se faz por notação **por excesso 127**; o expoente varia assim entre -127 e +128;
- **valor decimal de um fp em binário (normalizado):**  $V = (-1)^S * (1.F) * 2^{E-127}$ , em que S, F e E

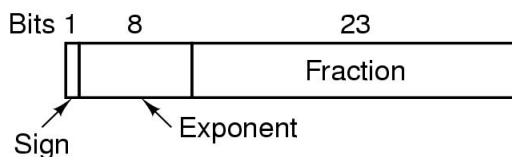
representam respectivamente os valores em binário dos campos no formato em fp;

- **representação de valores desnormalizados:** para poder contemplar este tipo de situação a norma IEEE reserva o valor de  $E = 0000\ 0000_2$  para representar valores desnormalizados, desde que se verifique também que  $F \neq 0$ ; o valor decimal vem dado por  $V = (-1)^S * (0.F) * 2^{-126}$
- **representação do zero:** é o caso particular previsto em cima, onde  $E = 0$  e  $F = 0$  ;
- **representação de  $\pm\infty$  :** a norma IEEE reserva a outra extremidade de representação do expoente; quando  $E = 1111\ 1111_2$  e  $F = 0$  , são esses os "valores" que se pretendem representar;
- **representação de n.º não real:** quando o valor que se pretende representar não é um n.º real (imaginário por exemplo), a norma prevê uma forma de o indicar para posterior tratamento por rotinas de excepção; neste caso  $E = 1111\ 1111_2$  e  $F \neq 0$  .

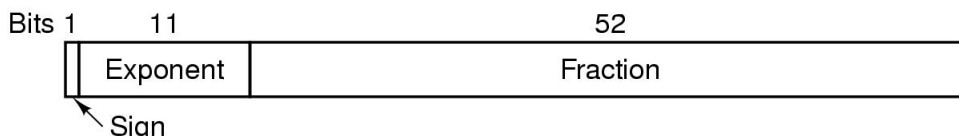
Normalized	±	0 < Exp < Max	Any bit pattern
Denormalized	±	0	Any nonzero bit pattern
Zero	±	0	0
Infinity	±	1 1 1...1	0
Not a number	±	1 1 1...1	Any nonzero bit pattern

↙ Sign bit

A norma IEEE 754 contempla ainda a representação de valores em fp que necessitem de maior intervalo de representação e/ou melhor precisão, por várias maneiras. A mais adoptada pelos fabricantes utiliza o dobro do n.º de bits, 64, pelo que é também conhecida pela representação em **precisão dupla**, enquanto a representação por 32 bits se designa por precisão simples. Para precisão dupla, a norma especifica, entre outros aspectos, que o expoente será representado por 11 bits e a parte fraccionária por 52 bits.



(a)



(b)

**Representação de reais com precisão simples (a) e dupla (b)**