COMSM1302 電腦架構概論
IEEE 754 浮點數補充教材

32 位元單精度（binary32）與 64 位元雙精度（binary64）
含 6 道練習題完整詳解

Week 2 補充教材（數的表示法延伸）

2026 年 7 月

為什麼放在第 2 週? 第 2 週我們學了無號數、2 補數與二進位加減法——這些都是**整數**的世界。真實程式還需要 3.14159、6.02×10^{23}、0.001 這類**實數**。本補充教材說明現代電腦（從手機到超級電腦）一致採用的實數表示標準：**IEEE 754**。模擬考的自訂 8 位元浮點格式題（1 位符號、2 位指數、5 位尾數）考的正是同一套原理，只是欄位縮小；把本教材讀通，那類題目就是「換個參數再算一次」。

目錄

1　為什麼需要浮點數?

用 16 位元表示數字時，若採**定點數**（fixed point），例如規定「小數點固定在第 8 位」，我們得到：

$$\underbrace{b_{15}\, b_{14} \cdots b_8}_{\text{整數部分}} \;.\; \underbrace{b_7\, b_6 \cdots b_0}_{\text{小數部分}}$$

最大約 255.996、解析度固定 $2^{-8} \approx 0.0039$。問題是：

- 想表示 10^{20}（天文數字）？整數位不夠。
- 想表示 10^{-20}（原子尺度）？小數位不夠。
- 兩者**同時**要？定點數完全無解——位元預算被小數點的「固定位置」綁死。

解法借自科學記號：6.02×10^{23} 用「**有效數字**」（6.02）搭配「**指數**」（23）——小數點的位置變成一個 **可調的欄位**，讓小數點「浮動」（floating），同一格式就能同時涵蓋巨大與微小的數。代價是：**有效位數固定**，數字越大，相鄰可表示值的間距越大（精度是「相對的」而非「絕對的」）。

IEEE 754 標準

IEEE 754（1985 年首版，最新修訂為 IEEE 754-2019）規定了浮點數的 **二進位格式、捨入規則、特殊值與運算行為**。兩種最重要的基本格式：**binary32**（單精度，C 的 `float`）與 **binary64**（雙精度，C 的 `double`、Python 的 `float`、JavaScript 的 `Number`）。

2　二進位科學記號與正規化

十進位科學記號要求有效數字介於 $1 \sim 9.99\ldots$；二進位的對應要求是有效數字介於 $1 \sim 1.11\ldots_2$（即 $[1, 2)$）：

$$13.625_{10} = 1101.101_2 = \underbrace{1.101101_2}_{\text{有效數字（significand）}} \times 2^{\overbrace{3}^{\text{指數}}}$$

把小數點移到「最高位的 1 之後」稱為**正規化**（normalisation）。

關鍵觀察：隱藏位元（hidden bit）。正規化後，小數點前**永遠是 1**（二進位只有 0 和 1，最高有效位必是 1）。既然一定是 1，就**不必儲存**——IEEE 754 只存小數點後的部分（fraction），解碼時自動補回「1.」。這讓 23 位的欄位實際提供 **24 位**的精度，白賺一個位元。

小數轉二進位的兩把工具（第 2.1 講的延伸）：

- 整數部分：反覆除以 2 取餘數（由下往上讀）；
- 小數部分：反覆**乘以 2 取整數位**（由上往下讀）。例：$0.625 \times 2 = 1.25$（取 1）、$0.25 \times 2 = 0.5$（取 0）、$0.5 \times 2 = 1.0$（取 1）$\Rightarrow 0.625 = 0.101_2$。

3　binary32：32 位元單精度格式

binary32 解碼公式（正規數）

$$x = (-1)^S \times 1.F_2 \times 2^{E-127} \qquad (1 \le E \le 254)$$

- S：符號位，0 為正、1 為負；
- E：8 位**偏移指數**（biased exponent）。實際指數 $e = E - 127$（偏移量 bias $= 127$），可用範圍 $e \in [-126, +127]$；$E = 0$ 與 $E = 255$ 保留給特殊值（見第 5 節）；
- F：23 位小數，解碼時補回隱藏位元成 $1.F$。

為什麼指數要加偏移量? 若指數用 2 補數存，比較兩個浮點數大小就要先拆欄位、分別處理。加上偏移後，指數欄位變成**無號數**且「越大代表指數越大」——整個 32 位元（正數時）可以**直接當無號整數比大小**，硬體比較器可沿用整數電路。這是刻意的工程設計。

3.1　編碼流程（十進位 → binary32 五步驟）

1. **定符號**：負數 $S = 1$，正數 $S = 0$，之後只處理絕對值；
2. **轉二進位**：整數部分除 2 取餘、小數部分乘 2 取整；
3. **正規化**：寫成 $1.f \times 2^e$；
4. **算指數欄位**：$E = e + 127$，轉成 8 位二進位；
5. **填小數欄位**：取 f 的前 23 位（不足補 0；超過則**捨入**，見第 6 節），組合 $S \,|\, E \,|\, F$。

快速示例：-6.25。$6.25 = 110.01_2 = 1.1001_2 \times 2^2$；$S = 1$；$E = 2 + 127 = 129 = 1000\,0001_2$；$F = 1001\,\underbrace{0\cdots0}_{19}$。

$$\underbrace{1}_{S}\ \underbrace{10000001}_{E}\ \underbrace{10010000000000000000000}_{F} = \texttt{0xC0C80000}$$

4　binary64：64 位元雙精度格式

binary64 解碼公式（正規數）

$$x = (-1)^S \times 1.F_2 \times 2^{E-1023} \qquad (1 \le E \le 2046)$$

偏移量 bias $= 1023$，實際指數範圍 $e \in [-1022, +1023]$，有效精度 $52 + 1 = 53$ 位。

原理與 binary32 完全相同，只是欄位加寬。兩種格式對照（順帶列出 GPU ／機器學習常用的 binary16）：

	binary16（half）	**binary32**（float）	**binary64**（double）
總位元數	16	32	64
符號／指數／小數	1/5/10	1/8/23	1/11/52
偏移量 bias	15	127	1023
實際指數範圍	$-14 \sim 15$	$-126 \sim 127$	$-1022 \sim 1023$
有效精度（位）	11	24	53
約當十進位有效位數	≈ 3.3	≈ 7.2	≈ 15.9
最大正規數	65504	$\approx 3.40 \times 10^{38}$	$\approx 1.80 \times 10^{308}$
最小正規數	$\approx 6.1 \times 10^{-5}$	$\approx 1.18 \times 10^{-38}$	$\approx 2.23 \times 10^{-308}$
最小次正規數	$\approx 6.0 \times 10^{-8}$	$\approx 1.40 \times 10^{-45}$	$\approx 4.94 \times 10^{-324}$
機器 epsilon（2^{-p+1}）	$2^{-10} \approx 9.8 \times 10^{-4}$	$2^{-23} \approx 1.19 \times 10^{-7}$	$2^{-52} \approx 2.22 \times 10^{-16}$

怎麼記偏移量？bias $= 2^{k-1}-1$（k 為指數位數）：binary16 是 $2^4-1=15$、binary32 是 $2^7-1=127$、binary64 是 $2^{10}-1=1023$。模擬考的 2 位指數自訂格式則是 $2^1-1=1$——同一條公式。

5　特殊值：零、次正規數、無限大、NaN

指數欄位「全 0」與「全 1」被保留，用來編碼四類特殊值（以 binary32 為例，binary64 把 255 換成 2047、-126 換成 -1022 即可）：

指數 E	小數 F	意義
0	$=0$	± 0（正零與負零，比較時相等）
0	$\neq 0$	**次正規數**：$(-1)^S \times 0.F_2 \times 2^{-126}$（無隱藏位元）
$1 \sim 254$	任意	正規數：$(-1)^S \times 1.F_2 \times 2^{E-127}$
255	$=0$	$\pm\infty$（如 $1/0=+\infty$、$-1/0=-\infty$）
255	$\neq 0$	**NaN**（Not a Number，如 $0/0$、$\sqrt{-1}$、$\infty-\infty$）

- **為什麼需要次正規數（subnormal）？**最小正規數是 $1.00\ldots0 \times 2^{-126}$。若再小就直接跳到 0，會出現「$x \neq y$ 但 $x-y=0$」的怪事。次正規數把隱藏位元改成 0（$0.F \times 2^{-126}$），讓數值以**漸進**方式滑向 0（gradual underflow），一路撐到 2^{-149}；代價是有效位數逐漸縮水。
- **NaN 的傳染性**：任何含 NaN 的運算結果都是 NaN，且 **NaN 不等於任何值（包括它自己）**——「`x != x`」是判斷 NaN 的慣用寫法。

- **無限大會飽和**：超出最大正規數的運算（如 $10^{38} \times 100$）產生 $\pm\infty$（溢位不繞回，與 2 補數整數溢位的行為完全不同）。

6　捨入：格子塞不下的時候

多數十進位小數（如 0.1）在二進位是**無限循環小數**，23 或 52 位的小數欄位塞不下，必須捨入。IEEE 754 預設採用：

Round to nearest, ties to even（就近捨入、平手取偶）

把真實值捨入到**最接近**的可表示浮點數；若恰好落在兩個可表示值的**正中間**（平手），取小數欄位**最後一位為 0（偶數）**的那個。

操作面：看第 24 位（要被切掉的第一位，稱 round bit）——

- round bit $= 0$：直接截斷（往下捨）；
- round bit $= 1$ 且後面**還有 1**：進位（往上入）；
- round bit $= 1$ 且後面全 0（正中間）：往「尾位為偶」的方向捨入。

「平手取偶」避免了「一律四捨五入」造成的系統性偏大——統計上一半往上、一半往下，長期誤差互相抵消。

7　常見陷阱（寫程式必知）

陷阱 1：$0.1 + 0.2 \neq 0.3$。三個數在 binary64 中都不精確（見練習 3 的原理），儲存值分別約為 $0.1000000000000000055\ldots$、$0.2000000000000000111\ldots$、$0.2999999999999999889\ldots$；相加再捨入後得 $0.3000000000000000444\ldots$，比「0.3 的儲存值」大了一個最低有效位（ULP）。所以 `0.1 + 0.2 == 0.3` 在幾乎所有語言中都是 `false`。**正確做法**：比較用容差 $|x - y| < \varepsilon$（ε 依問題尺度選取），或改用十進位定點／有理數型別處理金額等精確需求。

陷阱 2：大數吃小數（swamping）。浮點加法要先**對齊指數**：小的數往右移位，移出 24（或 53）位視窗的部分直接消失。極端例子：binary32 中 $2^{24} + 1 = 2^{24}$（見練習 6）。連鎖效應是**結合律失效**：$(a + b) + c \neq a + (b + c)$——累加一大串數字時，先加小的再加大的，誤差較小。

陷阱 3：可表示的數不是均勻分布。相鄰浮點數的間距 $= 2^e \times \varepsilon$，隨數字變大而變大：在 $[1, 2)$ 區間 binary32 的格點間距是 2^{-23}，在 $[2^{24}, 2^{25})$ 區間卻是 2。「數線越遠越稀疏」是理解一切精度問題的心智模型。

8　練習題（6 題完整詳解）

8.1　練習 1：十進位 → binary32——13.625

題目

把 13.625 編碼成 IEEE 754 binary32，分別寫出符號、指數、小數三個欄位，並以十六進位表示完整的 32 位元。

解答

第一步：定符號。正數 $\Rightarrow S = 0$。

第二步：轉二進位。整數部分 $13 = 1101_2$（$13 = 8 + 4 + 1$）；小數部分 0.625：

$$0.625 \times 2 = 1.25 \text{ (取1)}, \quad 0.25 \times 2 = 0.5 \text{ (取0)}, \quad 0.5 \times 2 = 1.0 \text{ (取1)} \ \Rightarrow \ 0.101_2$$

合併：$13.625 = 1101.101_2$。

第三步：正規化。小數點左移 3 位：

$$1101.101_2 = 1.101101_2 \times 2^3$$

第四步：指數欄位。 $E = 3 + 127 = 130 = 1000\,0010_2$。

第五步：小數欄位。隱藏位元「1.」不存，取小數點後的 101101，右邊補 0 到 23 位：

$$F = 1011\,0100\,0000\,0000\,0000\,000$$

組合並轉十六進位。

$$\underbrace{0}_{S}\ \underbrace{10000010}_{E}\ \underbrace{10110100000000000000000}_{F}$$

每 4 位一組：0100 0001 0101 1010 0000 0000 0000 0000

$$\boxed{13.625 = \texttt{0x415A0000}}$$

驗算（解碼回去）：$E = 130 \Rightarrow e = 3$；$1.101101_2 \times 2^3 = 1101.101_2 = 8+4+1+0.5+0.125 = 13.625$ ✓

8.2　練習 2：binary32 → 十進位——解碼 `0xC1690000`

題目

某記憶體位置存放的 32 位元為 `0xC1690000`。把它視為 IEEE 754 binary32，求它代表的十進位值。

解答

第一步：展開成二進位。

$$\texttt{C1690000} = 1100\,0001\,0110\,1001\,0000\,0000\,0000\,0000$$

第二步：切欄位。

$$\underbrace{1}_{S}\ \underbrace{10000010}_{E}\ \underbrace{11010010000000000000000}_{F}$$

（切法：第 1 位是 S；接下來 8 位是 E；剩下 23 位是 F。）

第三步：判斷類型。 $E = 1000\,0010_2 = 130$，不是 0 也不是 255 ⇒ 正規數，照公式解碼。

第四步：解碼。

- 符號：$S = 1$ ⇒ 負數；
- 實際指數：$e = 130 - 127 = 3$；
- 有效數字：補回隱藏位元 ⇒ 1.1101001_2。

$$1.1101001_2 \times 2^3 = 1110.1001_2 = 8 + 4 + 2 + 0.5 + 0.0625 = 14.5625$$

$$\boxed{\texttt{0xC1690000} = -14.5625}$$

驗算（反向編碼）： $14.5625 = 14 + 0.5625$；$14 = 1110_2$；$0.5625 = 0.5 + 0.0625 = 0.1001_2$；$1110.1001_2 = 1.1101001 \times 2^3$；$E = 130$、$S = 1$ ✓

解碼的第一件事永遠是「看指數欄位」：$E = 0$（零或次正規）與 $E = 255$（∞ 或 NaN）不能套正規數公式。養成先分類再解碼的習慣，練習 5 會專門考這件事。

8.3　練習 3：塞不下的 0.1——循環小數與捨入誤差

題目

(a) 證明 0.1 在二進位是無限循環小數，並寫出循環模式；

(b) 求 0.1 的 binary32 編碼（十六進位）；

(c) 求儲存值與 0.1 的實際誤差，並說明誤差方向（偏大或偏小）。

解答

(a) 乘 2 取整，找循環。

$$\begin{aligned}
0.1 \times 2 &= 0.2 \quad \text{取}0\\
0.2 \times 2 &= 0.4 \quad \text{取}0\\
0.4 \times 2 &= 0.8 \quad \text{取}0\\
0.8 \times 2 &= 1.6 \quad \text{取}1\\
0.6 \times 2 &= 1.2 \quad \text{取}1\\
0.2 \times 2 &= 0.4 \quad \text{取}0 \quad \leftarrow \text{小數部分回到}0.2\text{，開始循環}
\end{aligned}$$

一旦「待乘的小數部分」重複出現（0.2 再現），輸出就永遠循環：

$$0.1_{10} = 0.0\overline{0011}_2 = 0.0001100110011001100110011\ldots_2$$

分母 $10 = 2 \times 5$ 含質因數 5（不是 2 的冪次），所以**必然**除不盡——這與 1/3 在十進位無限循環同理。

(b) 正規化、取 23 位、捨入。

$$0.1 = 1.10011001100110011001100\,\underline{1100\ldots}_2 \times 2^{-4}$$

小數欄位只能存 23 位（上式底線前的部分），被切掉的部分以 1100… 開頭：round bit = 1 且後面還有 1 ⇒ **往上進位**：

$$F = 10011001100110011001100 + 1 = 10011001100110011001101$$

指數欄位 $E = -4 + 127 = 123 = 0111\,1011_2$，$S = 0$：

$$0\ 01111011\ 10011001100110011001101 = \boxed{\texttt{0x3DCCCCCD}}$$

(c) 誤差計算。儲存的有效數字是 $1.10011001100110011001101_2$，即整數 $2^{23} + F = 13421773$ 除以 2^{23}；乘上 2^{-4}：

$$\text{儲存值} = \frac{13421773}{2^{27}} = 0.100000001490116119384765625$$

$$\text{誤差} = \text{儲存值} - 0.1 \approx +1.49 \times 10^{-9}\quad \text{(因為捨入時進位，誤差偏大)}$$

相對誤差 $\approx 1.5 \times 10^{-8}$，符合單精度機器 epsilon（$2^{-23} \approx 1.2 \times 10^{-7}$）的量級。

這就是 $0.1 + 0.2 \neq 0.3$ 的根源：每個十進位小數存進去時都各自帶了 10^{-9}（float）或 10^{-17}（double）級的小誤差，運算再把誤差組合、放大。double 版的 0.1 同理可得 `0x3FB999999999999A`（52 位小數、一樣要進位）。

8.4　練習 4：十進位 → binary64——−2026.5

題目

把 −2026.5 編碼成 IEEE 754 binary64（雙精度），寫出各欄位並以十六進位表示完整的 64 位元。

解答

第一步：定符號。負數 ⇒ $S = 1$，以下處理 2026.5。

第二步：轉二進位。

$$2026 = 1024 + 512 + 256 + 128 + 64 + 32 + 8 + 2 = 11111101010_2$$

（驗算：1024+512=1536、+256=1792、+128=1920、+64=1984、+32=2016、+8=2024、+2=2026 ✓）

$$0.5 = 0.1_2 \quad \Rightarrow \quad 2026.5 = 11111101010.1_2$$

第三步：正規化。小數點左移 10 位：

$$11111101010.1_2 = 1.11111010101_2 \times 2^{10}$$

第四步：指數欄位（bias = 1023）。

$$E = 10 + 1023 = 1033 = 10000001001_2 \quad (11\text{ 位})$$

（$1033 = 1024 + 8 + 1$ ✓）

第五步：小數欄位。取小數點後的 11111010101（11 位），右邊補 41 個 0 湊滿 52 位。

組合並轉十六進位。

$$\underbrace{1}_{S}\ \underbrace{10000001001}_{E}\ \underbrace{11111010101\ \overbrace{0\cdots0}^{41}}_{F}$$

每 4 位一組：

$$1100\,0000\,1001\,1111\,1010\,1010\,0000\ldots0000$$

$$\boxed{-2026.5 = \texttt{0xC09FAA0000000000}}$$

驗算（解碼）：$E = 10000001001_2 = 1033 \Rightarrow e = 10$；$1.11111010101_2 \times 2^{10} = 11111101010.1_2 = 2026.5$；$S = 1$ 加負號 ✓

binary32 與 binary64 的流程一字不差，只是三個參數換掉：指數位數 8 → 11、bias 127 → 1023、小數位數 23 → 52。考試時先在題目旁邊寫下這三個參數再動手，可避免中途混用。

8.5　練習 5：特殊值判讀——六個位元樣式

題目

把下列六個 32 位元樣式視為 binary32，判斷每一個是「正規數、次正規數、±0、±∞、NaN」中的哪一類，並求出（可求的）數值：

編號	位元樣式（十六進位）
(a)	`0x80000000`
(b)	`0x7F800000`
(c)	`0x7FC00000`
(d)	`0x00000001`
(e)	`0x00400000`
(f)	`0x7F7FFFFF`

解答

先切欄位（$S \mid E \mid F$），再依「指數全 0 ／全 1」分類：

(a) `0x80000000` $= 1 \mid 00000000 \mid 00\ldots0$。$E = 0$ 且 $F = 0 \Rightarrow$ **負零** -0。（IEEE 754 規定 $-0 == +0$ 比較為真，但 $1/-0 = -\infty$，符號仍有語義。）

(b) `0x7F800000` $= 0 \mid 11111111 \mid 00\ldots0$。$E = 255$ 且 $F = 0 \Rightarrow +\infty$ （例如 1.0/0.0 的結果）。

(c) `0x7FC00000` $= 0\,|\,11111111\,|\,10\ldots0$。$E = 255$ 且 $F \neq 0 \Rightarrow$ **NaN**（小數最高位為 1，是 quiet NaN 的典型編碼；0/0、$\sqrt{-1}$ 都產生它）。

(d) `0x00000001` $= 0\,|\,00000000\,|\,00\ldots01$。$E = 0$ 且 $F \neq 0 \Rightarrow$ **次正規數**，公式為 $0.F_2 \times 2^{-126}$（沒有隱藏位元）：

$$0.\underbrace{00\ldots01}_{23\text{ 位}} \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149} \approx 1.4 \times 10^{-45}$$

這是 binary32 **最小的正數**。

(e) `0x00400000` $= 0\,|\,00000000\,|\,10\ldots0$。同為**次正規數**：

$$0.1_2 \times 2^{-126} = 2^{-1} \times 2^{-126} = 2^{-127} \approx 5.9 \times 10^{-39}$$

(f) `0x7F7FFFFF` $= 0\,|\,11111110\,|\,11\ldots1$。$E = 254$（差一步就全 1）$\Rightarrow$ **正規數**，且是**最大的有限值**：

$$1.\underbrace{11\ldots1}_{23} \times 2^{254-127} = (2 - 2^{-23}) \times 2^{127} \approx 3.4028 \times 10^{38}$$

再大一個 ULP 就變成 (b) 的 $+\infty$。

記憶法：數線的兩端各有一段保留區。指數全 0 管「最靠近 0 的地帶」（零與次正規數），指數全 1 管「超出範圍的地帶」（無限大與 NaN）；正規數住在中間 $E \in [1, 254]$。

8.6　練習 6：整數的精度極限——$2^{24}+1$ 與安全整數

題目

(a) 說明為什麼 $16\,777\,216$ $(= 2^{24})$ 可以被 binary32 精確表示，但 $16\,777\,217$ $(= 2^{24}+1)$ 不行；

(b) 在 binary32 中計算 $16\,777\,216.0 + 1.0$，說明捨入過程與結果；

(c) 由此推論：binary64 能精確表示的連續整數範圍是多少？這與 JavaScript 的 `Number.MAX_SAFE_INTEGER` 有什麼關係？

解答

(a) 有效位數只有 24 位。 binary32 的有效數字是「1 個隱藏位元 + 23 位小數」共 24 位。

$$2^{24} = 1.\underbrace{00\ldots0}_{23} \times 2^{24} \quad \text{（小數全 0，塞得下）} \checkmark$$

$$2^{24} + 1 = 1\underbrace{00\ldots0}_{23}1_2 = 1.\underbrace{00\ldots0}_{23}1_2 \times 2^{24}$$

需要**第 24 位小數**才放得下最後那個 1——超出 23 位的欄位，無法精確表示。✗

(b) 相加後捨入，結果不變。真實總和 $16\,777\,217 = 1.\underbrace{0\ldots0}_{23}1 \times 2^{24}$，夾在兩個可表示值中間：

$$16\,777\,216 = 1.\underbrace{0\ldots0}_{22}0\,\mathbf{0} \times 2^{24} \qquad 16\,777\,218 = 1.\underbrace{0\ldots0}_{22}1\,\mathbf{0} \times 2^{24}$$

被切掉的部分恰是「1 後面全 0」——**正中間的平手局面**。依「平手取偶」規則，取小數最後一位為 0 的那個：16 777 216 的尾位是 0（偶）、16 777 218 的尾位是 1（奇），所以捨回 16 777 216：

$$\boxed{16\,777\,216.0 + 1.0 = 16\,777\,216.0 \quad \text{(float 中)}}$$

加了等於沒加——迴圈計數器若用 `float`，數到 2^{24} 就永遠卡住，這是真實世界發生過的 bug 類型。

(c) binary64 的安全整數。同樣推理，53 位有效數字 $\Rightarrow [-2^{53}, 2^{53}]$ 內的整數全部精確可表示，且 $2^{53}+1$ 是第一個表示不了的整數。JavaScript 的 `Number` **只有** binary64 一種數值型別，因此

$$\texttt{Number.MAX\_SAFE\_INTEGER} = 2^{53} - 1 = 9\,007\,199\,254\,740\,991$$

超過它，整數運算就開始「跳號」（如 $2^{53}+1 == 2^{53}$ 為 `true`）；需要更大的整數時得改用 `BigInt`。

總結一句話：浮點數是「用固定的**有效位數**換取巨大的 **動態範圍**」。2^{24}（float）與 2^{53}（double）是「每個整數都有格子」的分界線；越過去，格點間距變成 2、4、8……精度是相對的，不是絕對的。

9　速查表

	binary32	**binary64**
欄位（$S/E/F$）	1/8/23	1/11/52
偏移量	127	1023
正規數公式	$(-1)^S\,1.F \times 2^{E-127}$	$(-1)^S\,1.F \times 2^{E-1023}$
次正規數公式（E=0）	$(-1)^S\,0.F \times 2^{-126}$	$(-1)^S\,0.F \times 2^{-1022}$
$\pm\infty$	E=255、F=0	E=2047、F=0
NaN	E=255、$F\neq 0$	E=2047、$F\neq 0$
安全整數上限	2^{24}	2^{53}

編碼五步驟：定符號 → 轉二進位 → 正規化 $1.f \times 2^e$ → $E = e + \text{bias}$ → 填 F（必要時就近捨入、平手取偶）。

解碼三步驟：切欄位 → 看 E 分類（全 0 ／全 1 特殊處理）→ 套公式。