COMSM1302 電腦架構概論
第二週完整教材：二進位算術與 ALU

表示數字 · 二進位加法 · 二進位減法 · 算術邏輯單元（ALU）

依據 Kira Clements（University of Bristol）課程講義編寫並延伸

2026 年 7 月

本教材的使用方式：本講義整合了第二週四份投影片（2.1 Representing Numbers、2.2 Binary Addition、2.3 Binary Subtraction、2.4 ALU）的全部內容，並補充了背景知識、詳細推導、IEEE 754 標準與 Hack ALU 完整規格。每章結尾附有「本章重點」整理；第 5 章為綜合練習題，附完整詳解。本週的主軸是一條完整的故事線：**數字如何用位元表示 → 位元如何相加 → 如何表示負數並相減 → 如何把這些運算包裝成 CPU 的核心元件（ALU）**。

目錄

1 表示數字（Representing Numbers） 3

1.1 十進位：你早就會的「位值系統」 . . . 3
1.2 二進位（Binary） . . . 3
1.3 十六進位（Hexadecimal） . . . 4
1.4 八進位（Octal） . . . 4
1.5 基底轉換（Base Conversion） . . . 4
1.6 不只是速記：實際應用 . . . 5
1.7 基底 N 的值與範圍 . . . 5
1.8 數字可以是什麼? . . . 6

2 二進位加法（Binary Addition） 7

2.1 $1+1$ 等於多少? . . . 7
2.2 半加器（Half Adder） . . . 7
2.3 直式加法與進位 . . . 7
2.4 全加器（Full Adder） . . . 8
2.5 全加器公式：Sum . . . 8

2.6 全加器公式：Carry out 9
2.7 用兩個半加器組全加器 9
2.8 漣波進位加法器（Ripple Carry Adder） 10

3 二進位減法（Binary Subtraction） 11
3.1 減法就是加法 11
3.2 符號-大小（Sign-Magnitude） 11
3.3 1 補數（1's Complement） 11
3.4 2 補數（2's Complement） 12
3.5 減法電路 12
3.6 溢位（Overflow） 13
3.7 C 語言中的溢位 14
3.8 定點數（Fixed-Point） 14
3.9 浮點數（Floating-Point） 15
3.10 浮點誤差 16

4 算術邏輯單元（ALU） 17
4.1 什麼是 ALU? 17
4.2 控制位（Control Bits） 17
4.3 多工器（Multiplexer） 18
4.4 解多工器（Demultiplexer） 18
4.5 Hack ALU 18

5 綜合練習題（附詳解） 21

A 附錄：速查表 25
A.1 2 的冪次表 25
A.2 各表示法對照（4 位元） 25
A.3 加法器公式速查 25
A.4 名詞中英對照 26
A.5 參考資料 26

1　表示數字（Representing Numbers）

1.1　十進位：你早就會的「位值系統」

十進位（decimal）是日常使用的標準數字系統——恰好用了跟手指一樣多的數字！十進位稱為**基底 10（base 10）**，因為它使用 10 個不同的數字（0–9），可用下標 10 標記，例如 123_{10}。

一個十進位數的值 = **每個數字乘上對應的 10 的冪次，再全部相加**：

10^3	10^2	10^1	10^0	十進位值
			7	$7 \times 10^0 = 7$
		8	2	$8 \times 10^1 + 2 \times 10^0 = 82$
	1	3	6	$1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 = 136$
4	9	2	0	$4 \times 10^3 + 9 \times 10^2 + 2 \times 10^1 = 4920$

這個「位值」觀念你早已內化成直覺——關鍵是：**基底不一定要是 10！**

1.2　二進位（Binary）

定義

二進位（binary）是**基底 2**：只有 2 個數字（0、1）。標記法：下標 2（101_2）或前綴 `0b`（`0b101`）。每個二進位數字稱為一個**位元（bit）**，如同開／關的開關——決定「該位置的 2 的冪次要不要加進總和」。最左邊的位元稱為**最高有效位元（MSB, Most Significant Bit）**，最右邊的稱為**最低有效位元（LSB, Least Significant Bit）**。

2^3	2^2	2^1	2^0	十進位值
0	0	0	1	$2^0 = 1$
0	1	0	1	$2^2 + 2^0 = 4 + 1 = 5$
1	1	0	0	$2^3 + 2^2 = 8 + 4 = 12$
1	0	1	1	$2^3 + 2^1 + 2^0 = 8 + 2 + 1 = 11$

二進位數的值 = **每個 1 位元所在位置對應的 2 的冪次之和**。

補充：十進位 → 二進位的兩種方法（講義未列，但實作必備）：

1. **減冪法**：反覆減去「不超過餘值的最大 2 的冪」。例：$45 = 32+8+4+1 = 2^5+2^3+2^2+2^0 = 101101_2$。
2. **除 2 取餘法**：反覆除以 2、記下餘數，最後**由下往上**讀。例：$45 \div 2 = 22 \cdots 1$；$22 \div 2 = 11 \cdots 0$；$11 \div 2 = 5 \cdots 1$；$5 \div 2 = 2 \cdots 1$；$2 \div 2 = 1 \cdots 0$；$1 \div 2 = 0 \cdots 1 \Rightarrow 101101_2$ ✓

1.3　十六進位（Hexadecimal）

定義

十六進位 (hexadecimal) 是**基底 16**，使用 16 個數字：$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f$（$a$–$f$ 代表十進位的 10–15）。標記法：下標 16（$7b1_{16}$）或前綴 `0x`（`0x7b1`）。

16^3	16^2	16^1	16^0	十進位值
			7	$7 \times 16^0 = 7$
		a	1	$10 \times 16^1 + 1 \times 16^0 = 160 + 1 = 161$
	3	1	0	$3 \times 16^2 + 1 \times 16^1 = 768 + 16 = 784$
5	0	0	f	$5 \times 16^3 + 15 \times 16^0 = 20480 + 15 = 20495$

十六進位最重要的用途是作為**二進位的速記法**：每個十六進位數字恰好對應一個 4 位元二進位數（因為 4 位元恰有 $2^4 = 16$ 種可能）：

十六進位	0	1	2	3	4	5	6	7
二進位	0000	0001	0010	0011	0100	0101	0110	0111
十六進位	8	9	a	b	c	d	e	f
二進位	1000	1001	1010	1011	1100	1101	1110	1111

同一個值只需 1/4 的十六進位數字就能表達，讀寫都更有效率、更不易出錯。

1.4　八進位（Octal）

定義

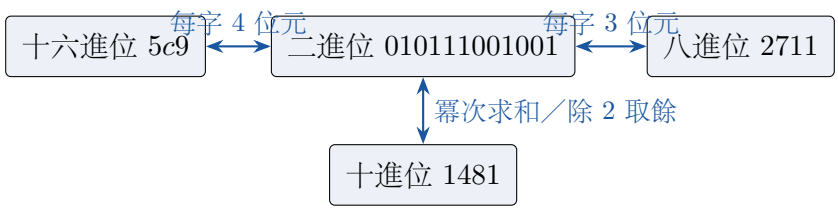
八進位 (octal) 是**基底 8**，使用 8 個數字（0–7）。標記法：下標 8（761_8）或前綴 `0o`（`0o761`）。每個八進位數字對應一個 **3 位元**二進位數（$2^3 = 8$）。

8^3	8^2	8^1	8^0	十進位值
			7	$7 \times 8^0 = 7$
		2	3	$2 \times 8^1 + 3 \times 8^0 = 16 + 3 = 19$
	6	0	1	$6 \times 8^2 + 1 \times 8^0 = 384 + 1 = 385$
4	0	1	0	$4 \times 8^3 + 1 \times 8^1 = 2048 + 8 = 2056$

八進位	二進位
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

1.5　基底轉換（Base Conversion）

既然十六進位與八進位都只是二進位的速記，**在不同基底間轉換時，先經過二進位**是最不易出錯的路徑：



例題 1.1：$5c9_{16}$ → 二進位 → 八進位 → 十進位

十六進位 → 二進位（每個數字獨立換成 4 位元）：

$$5 \to 0101, \quad c \to 1100, \quad 9 \to 1001 \qquad \Rightarrow \qquad 5c9_{16} = 0101\,1100\,1001_2$$

二進位 → 八進位（從 LSB 起每 3 位元一組）：

$$010\,111\,001\,001 \to 2, 7, 1, 1 \qquad \Rightarrow \qquad 2711_8$$

二進位 → 十進位（1 位元位置的冪次相加）：

$$2^{10} + 2^8 + 2^7 + 2^6 + 2^3 + 2^0 = 1024 + 256 + 128 + 64 + 8 + 1 = 1481$$

1.6　不只是速記：實際應用

- **八進位——Unix 檔案權限**：Linux 等類 Unix 系統以八進位顯示與設定檔案權限。例如 `chmod 754 filename`：

	user：7			group：5			other：4		
二進位	1	1	1	1	0	1	1	0	0
權限	r	w	x	r	–	x	r	–	–

  每組 3 個位元恰好是一個八進位數字：r（讀）、w（寫）、x（執行）。

- **十六進位——記憶體位址**：十六進位在計算領域無所不在，用來把冗長的二進位變得可讀，最典型的是**記憶體位址**（資料儲存位置的編號），例如 `0x7ffee4a3c8b0`。

1.7　基底 N 的值與範圍

Base N 通用公式

對基底 N、共 D 個數字 x_i（位置 i 從 0 到 $D-1$，每個 $x_i < N$）的數：

$$\text{值} = \sum_{i=0}^{D-1} x_i \cdot N^i \qquad \text{範圍（可表示的值的個數）} = N^D$$

同一串數字在不同基底下代表不同的值——「101」是什麼要看表示法：

表示法	標記	十進位值
十進位	101_{10}	101
二進位	101_2 或 `0b101`	5
十六進位	101_{16} 或 `0x101`	257
八進位	101_8 或 `0o101`	65

範圍為什麼重要? 電腦是有限的機器，每種資料型別都用固定量的記憶體儲存。範圍告訴我們不同型別能表示的最小與最大值。例如 `unsigned int` 若用 4 位元組儲存（1 位元組 byte = 8 位元），就能存 $2^{4\times 8} = 2^{32}$ 個值，即十進位整數 0 到 4,294,967,295。

1.8　數字可以是什麼?

二進位數是電腦系統的基石，但這些值可以**代表任何東西**：數量、顏色（如 RGB `0xFF6600`）、字元（如 ASCII 編碼 65 =「A」）、角度……端看我們如何**詮釋**這串位元。「表示法」正是第 3 章的核心——同一串位元，用不同表示法讀，值完全不同。

本章重點

- 位值系統：基底 N 的數值 $= \sum x_i N^i$；D 個數字可表示 N^D 個值。
- 二進位：bit、MSB ／ LSB；十六進位 = 4 位元一組的速記；八進位 = 3 位元一組。
- 跨基底轉換：先過二進位最保險；十進位 → 二進位可用減冪法或除 2 取餘法。
- 實際應用：`chmod` 的八進位權限、十六進位記憶體位址。
- 位元串本身沒有意義，**表示法賦予它意義**。

2　二進位加法（Binary Addition）

2.1　$1+1$ 等於多少?

回到加法的基本功：兩個單一位元相加，可能的結果是什麼?

A	B	十進位和
0	0	0
0	1	1
1	0	1
1	1	2

A	B	二進位和
0	0	0
0	1	1
1	0	1
1	1	10

輸入只能是 0 或 1，但輸出有三種可能：0、1、2（$0b00$、$0b01$、$0b10$）——需要**兩個位元**才裝得下。於是輸出拆成兩個單位元訊號：**和（Sum, S）**與**進位（Carry, C）**。

2.2　半加器（Half Adder）

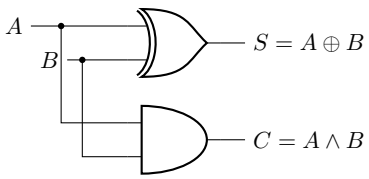
定義

半加器（half adder）：輸入兩個位元 A、B，輸出 C（進位，權重 2^1）與 S（和，權重 2^0）。

A	B	C（2^1）	S（2^0）
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

回顧第一週的邏輯運算子——哪兩個的真值表跟 C、S 一模一樣? C 欄是 AND、S 欄是 XOR!

$$\boxed{C = A \wedge B, \qquad S = A \oplus B}$$



有了這兩條式子，就能把 2 個單位元訊號相加。但要加**多位元**的數，還缺一塊拼圖。

2.3　直式加法與進位

十進位與二進位的直式加法都靠**進位（carry）**把資訊往左傳：從最右邊的數字（二進位是 bit 0，即 LSB）開始加，其進位參與緊鄰左邊那一位（bit 1）的計算，依此類推。

	十進位	
A		9
B	1	3
和	2	2
進位	1	

	二進位			
A	1	0	0	1
B	1	1	0	1
和	1	0	1	1	0
進位	1	0	0	1	

右例：$1001_2 + 1101_2 = 10110_2$（$9 + 13 = 22$ ✓）。黃色的第 5 個位元是**溢位（overflow）**——我們目前用 4 位元表示數字，裝不下第 5 位。暫時先忽略它，第 3 章會回來詳談！

2.4　全加器（Full Adder）

多位元加法時，每一位除了 A、B 還要吃前一位的進位——需要**第三個輸入** C_{in}：

定義

全加器（full adder）：輸入三個位元 C_{in}、A、B，輸出 C_{out}（進位）與 S（和）。名稱由來：一個全加器（大致上）可由**兩個半加器**組成——第一個算 $A + B$，第二個把該結果與 C_{in} 相加。

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

（檢查：三個輸入的 1 的個數 $= 2 \cdot C_{out} + S$，例如三個都是 1 時 $3 = 2 \cdot 1 + 1$ ✓。）

2.5　全加器公式：Sum

把 S 填入卡諾圖（列 AB、欄 C_{in}）：

C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

棋盤式分布——**沒有任何相鄰的 1 可以圈在一起**！用 DNF 直接寫出（這已是只用 $\neg, \wedge, \vee$ 的最簡形式）：

$$S = (C_{in} \wedge \neg A \wedge \neg B) \vee (\neg C_{in} \wedge \neg A \wedge B) \vee (C_{in} \wedge A \wedge B) \vee (\neg C_{in} \wedge A \wedge \neg B)$$

但改用 XOR 就非常精簡：

$$\boxed{S = A \oplus B \oplus C_{in}}$$

多輸入的 XOR 在**奇數個輸入為 1** 時輸出 1——正是「和位元」的行為（1 的個數為奇數時 $S = 1$）。棋盤式 K-map 正是 XOR 的 signature：看到它就該想到 XOR。

2.6 全加器公式：Carry out

把 C_{out} 填入卡諾圖：

C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

三個兩格的框（皆含右下角的 111）：

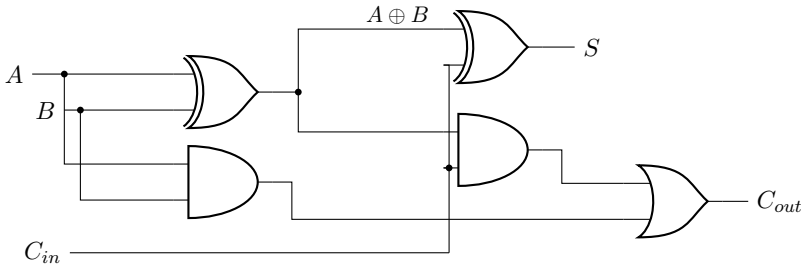
$$C_{out} = (C_{in} \wedge B) \vee (A \wedge B) \vee (C_{in} \wedge A)$$

直覺讀法：「三個輸入中至少兩個為 1」（多數決函數）。用分配律可改寫成：

$$C_{out} \equiv (A \wedge B) \vee (C_{in} \wedge (A \vee B)) \equiv \boxed{(A \wedge B) \vee (C_{in} \wedge (A \oplus B))}$$

為什麼最後一步可以把 $\vee$ 換成 $\oplus$？兩者只在 $A = B = 1$ 時不同，而此時 $(A \wedge B) = 1$ 已讓整個式子為 1，所以差異被吸收，結果不變。**工程上偏好 $\oplus$ 版本**：$A \oplus B$ 這個訊號在 Sum 電路（$S = (A \oplus B) \oplus C_{in}$）裡本來就有，**分支複用**即可，省下一個閘。

2.7 用兩個半加器組全加器

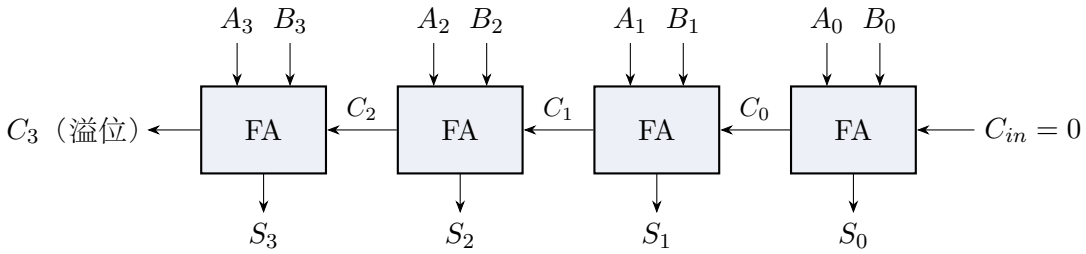


- 半加器 1（左）：算 $A \oplus B$ 與 $A \wedge B$；

- 半加器 2（中）：把 $A \oplus B$ 與 C_{in} 相加，得 $S = A \oplus B \oplus C_{in}$ 與 $C_{in} \wedge (A \oplus B)$；
- OR 閘（右）：合併兩個進位，$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$——正是上一節的偏好公式！

2.8　漣波進位加法器（Ripple Carry Adder）

從加 2 個位元（A, B）進化到加 3 個位元（A, B, C_{in}）之後，把全加器**串接**起來，就能加任意多位元的數：



定義

漣波進位加法器（ripple carry adder）：每個全加器的 C_{out} 接到下一個（更高位）全加器的 C_{in}。名稱由來：LSB 產生的進位可能像**漣漪**一樣層層影響到所有更高位的結果。注意位元編號**由右至左**（bit 0 在最右），與加法進行的方向一致。

延伸補充：漣波的代價。進位必須逐級傳遞，第 N 位的結果要等前面 $N-1$ 級都穩定後才正確——最壞情況（如 $1111 + 0001$）進位從 LSB 一路傳到 MSB，延遲與位元數**成正比**。實際 CPU 常用**進位前瞻加法器（carry-lookahead adder）**：用額外邏輯「預測」每一位的進位（generate $g_i = A_i \wedge B_i$、propagate $p_i = A_i \oplus B_i$），把延遲從 $O(N)$ 降到 $O(\log N)$，代價是更多的閘。這是「用面積換速度」的經典取捨。

本章重點

- 半加器：$S = A \oplus B$、$C = A \wedge B$（兩位元相加，輸出需兩位元）。
- 全加器：$S = A \oplus B \oplus C_{in}$（奇數個 1 則 $S = 1$）；$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$（至少兩個 1 則進位）。
- Sum 的 K-map 是棋盤式——無法圈框，這是 XOR 的特徵。
- 全加器 = 2 個半加器 + 1 個 OR；$A \oplus B$ 訊號可在 Sum 與 Carry 電路間共用。
- 漣波進位加法器：全加器串接、進位逐級傳遞；最高位溢出的進位即 overflow。

3　二進位減法（Binary Subtraction）

3.1　減法就是加法

核心觀念

$$A - B \equiv A + (-B)$$

減法等價於「加上一個負數」——這代表**已經做好的加法器硬體可以直接重用，零額外成本**！唯一缺的是：負數的二進位表示法。

常用的帶號數表示法有三種：**符號-大小（sign-magnitude）**、**1 補數（1's complement）**、**2 補數（2's complement）**。前兩種各有缺陷，第三種是現代電腦的標準。

3.2　符號-大小（Sign-Magnitude）

值的表示方式與無號二進位相同，只是 **MSB 改為代表正負號**（0 正 1 負）：

$-$	2^2	2^1	2^0	十進位值
0	0	0	1	$2^0 = 1$
0	1	0	1	$2^2 + 2^0 = 5$
1	1	0	0	$-(2^2) = -4$
1	0	1	1	$-(2^1 + 2^0) = -3$

4 位元可表示 $+7$ 到 -7——只有 15 個數，但 4 位元明明有 $2^4 = 16$ 種組合。**缺陷：0 有兩種表示**：

$$0000_2 = +0_{10} \qquad 1000_2 = -0_{10}$$

3.3　1 補數（1's Complement）

負數定義為其正數版本的**補數（complement）**——把每個位元 $0 \leftrightarrow 1$ 翻轉：

十進位	0	1	2	3	4	5	6	7
二進位	0000	0001	0010	0011	0100	0101	0110	0111
十進位	-7	-6	-5	-4	-3	-2	-1	-0
二進位	1000	1001	1010	1011	1100	1101	1110	1111

問題依舊：**0 仍有兩種表示**（0000 與 1111），沒有用滿整個範圍；而且**算術在 0 附近不一致**——1111 + 1 會得到 0000（帶溢位），兩者在十進位裡都是 0，跨越 0 的加法需要額外修正。

3.4　2 補數（2's Complement）

定義

2 補數(2's complement)是現代電腦的標準表示法：MSB 仍然管正負，但**同時帶有權重** -2^{N-1} ——其餘位元的權重與無號二進位相同。N 位元 2 補數的範圍是 $+(2^{N-1}-1)$ 到 $-(2^{N-1})$。

-2^3	2^2	2^1	2^0	十進位值
0	0	0	1	$2^0 = 1$
0	1	0	1	$2^2 + 2^0 = 5$
1	1	0	0	$-(2^3) + 2^2 = -8 + 4 = -4$
1	0	1	1	$-(2^3) + 2^1 + 2^0 = -8 + 2 + 1 = -5$

4 位元可表示 $+7$ 到 -8：**0 只有一種表示**，16 種組合全部用滿——比前兩種方案多出一個可表示的值。正數的表示與無號二進位相同，但要注意**至少補一個 0 當 MSB**（否則 MSB 會被讀成負權重）。

例題 3.1：求 -7 的 2 補數表示

步驟 0：決定位元數。N 位元 2 補數範圍是 $+(2^{N-1}-1)$ 到 $-(2^{N-1})$，表示 -7 至少需要 4 位元（範圍 $+7 \sim -8$）。

轉換法：取 1 補數再加 1（「flip and add 1」）：

無號 $+7$	0	1	1	1
翻轉所有位元（1 補數）	1	0	0	0
加 1（2 補數）	1	0	0	1

驗算：$1001_2 = -8+1 = -7$ ✓（此法雙向通用：對 1001 再做一次「翻轉加一」會得回 0111 = +7。）

為什麼「翻轉加一」有效？ 對 N 位元數 B：翻轉所有位元得到 $(2^N-1)-B$（每一位都從權重和裡「反選」），再加 1 得 $2^N - B$。在丟棄第 N 位溢出的模 2^N 算術裡，$2^N - B \equiv -B$——這正是 2 補數能讓加法器「免費」處理負數的數學原因。

3.5　減法電路

$$A - B \equiv A + (-B) \equiv A + (\mathrm{NOT}(B) + 1)$$

要讓加法器架構支援減法，只需兩個小改動：

1. 用 **NOT 閘**把 B 的每個位元翻轉；
2. 漣波進位加法器**第一個全加器的** C_{in} **本來閒置**（原本接 0）——把它設成 1，正好完成「加 1」!

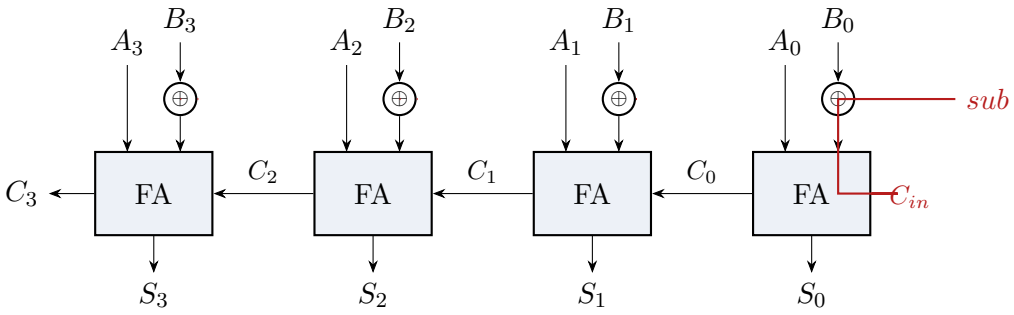
例題 3.2：$3-7$

$A=3=0011_2$，$B=7=0111_2$，$\mathrm{NOT}(B)=1000_2$，$C_{in}=1$：

A	0	0	1	1
$\mathrm{NOT}(B)+1$	1	0	0	1
和	1	1	0	0
進位		1	1	

結果 1100_2 以 2 補數解讀：$-(2^3)+2^2=-4$ ✓（$3-7=-4$）。

加減兩用器（adder-subtractor）：只差一個「選擇要不要翻轉 B 並加 1」的機制。標準做法（延伸補充）：加一條控制線 sub，每個 B_i 先過一個 XOR 閘（另一輸入接 sub），sub 同時接到 C_{in}：



原理：$B_i\oplus 0=B_i$（照常加法）、$B_i\oplus 1=\neg B_i$（翻轉）——XOR 是「可控反相器」。$sub=0$ 時算 $A+B$；$sub=1$ 時算 $A+\neg B+1=A-B$。一條線同時完成兩件事。

3.6　溢位（Overflow）

之前加法時我們把多出來的進位「先忽略」，現在它有了重要用途：使用 2 補數加法時，**溢位用來判斷「丟棄多餘位元後的結果是否正確」**。

例題 3.3：兩個經典的錯誤結果

(a) $6+5=-5$?

$A=6$	0	1	1	0
$B=5$	0	1	0	1
和	0	1	0	1	1
進位		1			

4 位元下 $1011_2=-8+2+1=-5$——兩個正數相加得到負數，**錯**！（正確答案 11 超出 4 位元 2 補數的上限 $+7$。）

(b) $-7+(-2)=7$?

$A = -7$		1	0	0	1
$B = -2$		1	1	1	0
和	1	0	1	1	1
進位		1			

丟棄第 4 位後得 $0111_2 = +7$——兩個負數相加得到正數，**錯**！（正確答案 -9 低於下限 -8；結果太小裝不下也稱**下溢 underflow**。）

溢位判斷規則

用 4 位元（或任何固定位元數）的 2 補數運算時，超出範圍的位元必須丟棄；若丟棄後**結果的正負號改變**——**正 + 正 = 負**，或**負 + 負 = 正**——就發生了溢位錯誤。（正數加負數**永遠不會**溢位：結果的絕對值不會比兩個運算元大。）

3.7　C 語言中的溢位

整數溢位是程式「莫名其妙不對」的常見原因：

```
int num = INT_MAX;
// prints the maximum value for an int (2147483647 for 32-bit int)
printf("%i\n", num);

num++;
// prints the value after overflow (-2147483648 for 32-bit int)
printf("%i\n", num);
```

$2147483647 = 2^{31} - 1$（32 位元 `int` 的上限）加 1 後，繞回到最小值 $-2^{31} = -2147483648$——正是「正 + 正 = 負」的溢位。兩點實務教訓：

- 寫程式時假設變數用**最壞情況（最小）的記憶體大小**儲存，比較安全；
- 同一段計算在 A 機器正常，搬到用更少記憶體存該型別的 B 機器上可能出錯。

3.8　定點數（Fixed-Point）

整數表示法無法表示非整數，會損失精度。**定點數（fixed-point）**：與二進位相同，但**指定部分位元代表分數**（仍是 2 的冪次，只是次方為負）：

標記	2^1	2^0	2^{-1}	2^{-2}	十進位值
00.01	0	0	0	1	$2^{-2} = 0.25$
01.01	0	1	0	1	$2^0 + 2^{-2} = 1.25$
11.00	1	1	0	0	$2^1 + 2^0 = 3$
10.11	1	0	1	1	$2^1 + 2^{-1} + 2^{-2} = 2.75$

小數點的位置由需求決定：**範圍（range）**——最大最小值，與**精度（precision）**——值之間的增量。上例（2 整數位 + 2 小數位）只能表示 0 到 3.75、增量 0.25。若要負的分數，同樣可以讓 MSB 帶 2 補數的負權重。

3.9　浮點數（Floating-Point）

定義

浮點數（floating-point）是更靈活的分數表示法——不必事先固定整數位與小數位的數量，而是用公式：

$$(-1)^S \cdot M \cdot B^E$$

S = 符號（sign）、M = 尾數（mantissa）、B = 基底（base，二進位取 2）、E = 指數（exponent）。就是「二進位的科學記號」。

兩個關鍵設計：

- **偏移指數（biased exponent）**：指數加上一個偏移量（bias）後儲存，讓負指數也能存成正值（免去指數自己再帶一套符號系統）。k 位元指數的 bias 為 $2^{k-1}-1$（3 位元 $\Rightarrow$ bias $= 3$）。
- **正規化尾數（normalised mantissa）**：把數字調整成「小數點前恰有一個 1」（$1.xxx_2 \times 2^E$）——既然開頭必為 1，就**不儲存這個隱含的前導 1**，省一個位元的精度。

最常見的浮點精度由 **IEEE 754** 技術標準定義，它同時規定了無限大（∞）與 NaN（Not a Number）等特殊值的表示。

例題 3.4：迷你浮點格式 S(1)-E(3)-M(4)

用 1 位符號、3 位指數（bias = 3）、4 位尾數，編碼 2.125 與 -0.625：

步驟 1——轉成二進位（同定點數）：

$$2.125_{10} = 10.001_2 \text{（符號 0）} \qquad -0.625_{10} = 0.101_2 \text{（符號 1）}$$

步驟 2——正規化（小數點前留一個 1）：

$$10.001_2 = 1.0001_2 \times 2^1 \qquad 0.101_2 = 1.01_2 \times 2^{-1}$$

步驟 3——填入格式（指數加 bias 3；尾數去掉前導 1）：

S	偏移指數 $2^2\,2^1\,2^0$	尾數 $2^{-1}\,2^{-2}\,2^{-3}\,2^{-4}$	十進位驗算
0	1 0 0	0 0 0 1	$(-1)^0(2^0+2^{-4})\cdot 2^{(4-3)} = 1.0625 \times 2 = 2.125$ ✓
1	0 1 0	0 1 0 0	$(-1)^1(2^0+2^{-2})\cdot 2^{(2-3)} = -1.25 \times 0.5 = -0.625$ ✓

（3 位元指數配 bias 3：實際指數 $-3 \sim 4$ 存成 $0 \sim 7$；例如存 2 = 實際指數 -1。尾數欄不含隱含的前導 1。）

3.10　浮點誤差

浮點數的小數部分是**負的 2 的冪次的選和**（$2^{-1}=\frac{1}{2}$、$2^{-2}=\frac{1}{4}$、$2^{-3}=\frac{1}{8}\cdots\cdots$）。因此許多「看起來簡單」的分數**無法精確表示、只能近似**，而某些「看起來複雜」的分數反而可以精確表示：

- 0.1_{10}：二進位是**無限循環**的 $0.000110011001100\ldots_2$——任何有限位元的浮點格式都只能存近似值。這正是著名的 `0.1 + 0.2 != 0.3` 問題的根源（IEEE 754 雙精度下 $0.1+0.2=0.30000000000000004\ldots$）。
- $0.375_{10}=0.011_2=2^{-2}+2^{-3}$：有限、可精確表示 ✓。

延伸補充：IEEE 754 單精度（`float`）的實際參數：$S(1)$–$E(8)$–$M(23)$，bias $=127$；指數全 0 與全 1 保留給特殊值（±0、次正規數、±∞、NaN）。雙精度（`double`）為 $S(1)$–$E(11)$–$M(52)$，bias $=1023$。實務守則：**永遠不要用 `==` 比較浮點數**，應改用「差的絕對值小於容忍度」；金融計算應使用整數（以分為單位）或十進位型別。

本章重點

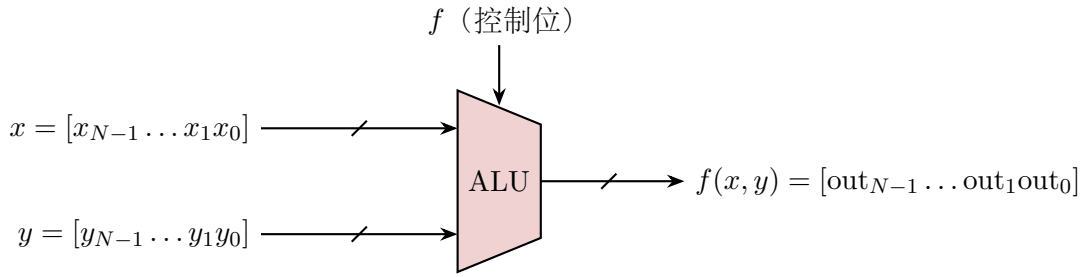
- $A-B=A+(-B)$：重用加法器，只需負數表示法。
- 符號-大小與 1 補數都有「雙零」缺陷；**2 補數**是標準：MSB 權重 -2^{N-1}、範圍 $+(2^{N-1}-1)\sim-(2^{N-1})$、0 唯一。
- 求 $-B$：翻轉所有位元再加 1；電路上用 NOT（或 XOR）＋第一級 $C_{in}=1$。
- 溢位：正 ＋ 正 ＝ 負或負 ＋ 負 ＝ 正即錯誤；正 ＋ 負永不溢位。C 的 `INT_MAX + 1` 繞回最小值。
- 定點數：固定的整數位／小數位，範圍與精度的取捨。
- 浮點數：$(-1)^S M\cdot B^E$、偏移指數、隱含前導 1 的正規化尾數；IEEE 754；0.1 在二進位無限循環 ⇒ 浮點誤差不可避免。

4　算術邏輯單元（ALU）

4.1　什麼是 ALU?

定義

算術邏輯單元（ALU, Arithmetic Logic Unit）是 CPU 的核心部件：接收兩個 N 位元輸入 x、y，在其間執行一個函數 f，輸出 N 位元結果 $f(x,y)$。f 從一組**預先定義的運算**中選出，可以是邏輯的或算術的。



兩個要點：

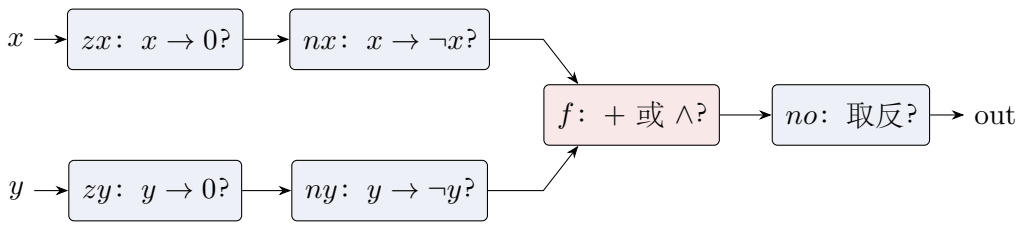
- **資料匯流排（data bus）**：一次傳送多位元訊號的位元陣列（圖中斜線記號表示多位元線）。
- **邏輯運算是逐位（bitwise）的**：$\text{out}_0 = x_0 \wedge y_0$、$\text{out}_1 = x_1 \wedge y_1$……各位獨立；**算術運算則有進位**在位元之間傳遞（第 2 章的漣波進位）。

4.2　控制位（Control Bits）

ALU 怎麼知道要算哪個函數？——用**控制位（control bits，又稱 select bits）**。以本課程的 **Hack ALU** 為例，它用 6 個控制位，每個都是二選一：

控制位	作用
zx	是否把 x 訊號**歸零**（ZERO the x signal）
nx	是否把 x 訊號**取反**（NOT the x signal）
zy	是否把 y 訊號**歸零**（ZERO the y signal）
ny	是否把 y 訊號**取反**（NOT the y signal）
f	選擇 **ADD（加法）或 AND**（f=1 加、f=0 且）
no	是否把**輸出**取反（NOT the out signal）

處理順序是一條固定的管線（zx, nx 先於 zy, ny 並行，再 f、最後 no）：

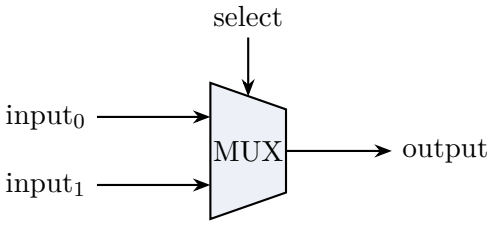


4.3　多工器（Multiplexer）

控制位的「二選一」在硬體上如何實現？答案是多工器。

定義

N 對 1 多工器（N-to-1 multiplexer, MUX）：接收 N 個輸入，用控制訊號**選擇其中一個**傳到輸出。選擇 N 個輸入需要 $\log_2(N)$ 個選擇位（$2^X = N$）。

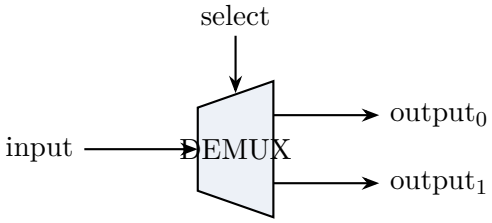


2 對 1 多工器只需 1 個選擇位：select = 0 輸出 input$_0$；select = 1 輸出 input$_1$。布林式：output $= (\neg s \wedge i_0) \vee (s \wedge i_1)$。Hack ALU 的每個控制位背後就是一排（16 個位元同時切換的）2 對 1 MUX。

4.4　解多工器（Demultiplexer）

定義

1 對 N 解多工器（1-to-N demultiplexer, DEMUX）：反方向——接收 1 個輸入，由控制訊號決定把它**送到 N 條輸出線的哪一條**；未被選中的輸出線輸出 0。同樣需要 $\log_2(N)$ 個選擇位。



1 對 2 解多工器：select = 0 時輸入走 output$_0$、select = 1 時走 output$_1$。（MUX 是「多路進、選一路出」，DEMUX 是「一路進、選一路出去」——一對互為鏡像的流量指揮工具。）

4.5　Hack ALU

Hack ALU 的輸入輸出都是 **16 位元資料匯流排上的 2 補數值**。把控制位設成下表的組合，就會計算出對應的函數（這是精簡版；完整的 ALU 真值表還列出其他組合可得的更多函數）：

zx	nx	zy	ny	f	no	out
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	0	0	1	0	$x+y$
0	1	0	0	1	1	$x-y$
0	0	0	0	0	0	$x \,\&\, y$
0	1	0	1	0	1	$x \mid y$

例題 4.1：追蹤 111111 為什麼輸出 1（講義 Example 1）

以 4 位元示範（實際是 16 位元，原理相同）：

1. $zx=1$、$zy=1$：x、y 都歸零 → 兩者皆為 0000_2；
2. $nx=1$、$ny=1$：兩者都取反 → 皆為 1111_2（即 -1）；
3. $f=1$：相加 → $1111_2+1111_2=1110_2$（丟棄溢出位；$-1+-1=-2$ ✓）；
4. $no=1$：輸出取反 → $\neg 1110_2=0001_2=1_{10}$。

（2 補數下 $\neg t=-t-1$，所以 $\neg(-2)=2-1=1$ ✓。）

例題 4.2：010101 為什麼輸出 $x \mid y$（講義 Example 2）

1. $zx=0$、$zy=0$：x、y 保持原值；
2. $nx=1$、$ny=1$：取反 → $\neg x$、$\neg y$；
3. $f=0$：AND → $\neg x \wedge \neg y$；
4. $no=1$：取反 → $\neg(\neg x \wedge \neg y)$。

但表上寫的是 $x \mid y$？——**德摩根定律**：$\neg(\neg x \wedge \neg y) \equiv x \vee y$，一模一樣！第一週的定律在 ALU 設計裡直接派上用場。

延伸補充：Hack ALU 的完整能力（nand2tetris）。六個控制位共 $2^6=64$ 種組合，官方文件列出其中 18 種「有名字」的函數：

$0,\ 1,\ -1,\ x,\ y,\ \neg x,\ \neg y,\ -x,\ -y,\ x{+}1,\ y{+}1,\ x{-}1,\ y{-}1,\ x{+}y,\ x{-}y,\ y{-}x,\ x \,\&\, y,\ x \mid y$

Hack ALU 另外輸出兩個狀態旗標：zr（out $=0$ 時為 1）與 ng（out <0 時為 1）——之後寫組合語言的條件跳躍時會用到。硬體實作恰好就是本章的元件組合：每個控制位一排 Mux16，f 用 Add16（漣波進位加法器）與 And16（逐位 AND），全部由第一週的 NAND 蓋起來。

例題 4.3：驗證 $x-y$ 的控制位 010011

zx=0, nx=1：$x \to \neg x$；zy=0, ny=0：y 不動；f=1：$\neg x + y$；no=1：$\neg(\neg x + y)$。
用 2 補數恆等式 $\neg t = -t - 1$ 化簡：

$$\neg x + y = (-x - 1) + y = y - x - 1$$

$$\neg(y - x - 1) = -(y - x - 1) - 1 = x - y + 1 - 1 = x - y \quad \checkmark$$

ALU 設計者巧妙地用「取反」湊出了「加 1 減 1」的效果，不需要任何額外的加法硬體。

本章重點

- ALU：CPU 核心，N 位元 $x, y \to f(x, y)$；邏輯運算逐位、算術運算帶進位。
- 控制位選擇函數；Hack ALU 用 6 個：zx, nx, zy, ny, f, no。
- MUX：N 選 1，需 $\log_2 N$ 個選擇位；DEMUX：1 分 N；控制位背後就是 MUX。
- 追蹤控制位管線（歸零 → 取反 → 加/且 → 取反）即可推出任何組合的輸出；德摩根與 $\neg t = -t - 1$ 是兩把關鍵鑰匙。
- 六個簡單開關的組合就能算出 18 種函數——極簡設計的威力。

5　綜合練習題（附詳解）

練習 1：基底轉換

把 $0b10110111$ 轉成十進位、十六進位與八進位。

解答

十進位：$2^7+2^5+2^4+2^2+2^1+2^0 = 128+32+16+4+2+1 = 183$。

十六進位（4 位元一組，由右往左）：$1011\,0111 \to b, 7 \Rightarrow$ `0xb7`。

八進位（3 位元一組，不足左補 0）：$10\,110\,111 \to 010\,110\,111 \to 2,6,7 \Rightarrow$ `0o267`。

驗算：$2\times 64 + 6\times 8 + 7 = 128+48+7 = 183$ ✓

練習 2：十進位轉二進位

把 77_{10} 轉成二進位（兩種方法各做一次），再寫出其十六進位。

解答

減冪法：$77 = 64+13 = 64+8+5 = 64+8+4+1 = 2^6+2^3+2^2+2^0 = 1001101_2$。

除 2 取餘法：$77 \to 38$ 餘 1；$38 \to 19$ 餘 0；$19 \to 9$ 餘 1；$9 \to 4$ 餘 1；$4 \to 2$ 餘 0；$2 \to 1$ 餘 0；$1 \to 0$ 餘 1。由下往上讀：1001101_2 ✓（兩法一致）。

十六進位：$0100\,1101 \to$ `0x4d`。

練習 3：範圍計算

(a) 3 個位元組的無號整數能表示哪些值？(b) 6 位元 2 補數的範圍？(c) 為什麼 4 位元 2 補數能表示的值比符號-大小多一個？

解答

(a) 3 位元組 $= 24$ 位元 $\Rightarrow 2^{24} = 16{,}777{,}216$ 個值，即 $0 \sim 16{,}777{,}215$。

(b) $N=6$：$+(2^5-1) \sim -(2^5)$，即 $+31 \sim -32$。

(c) 符號-大小有兩種 0（+0、−0），浪費一種組合；2 補數的 0 唯一（0000），省下的組合用來多表示一個負數（-8），把 $2^4 = 16$ 種組合用好用滿。

練習 4：二進位加法與溢位

以 4 位元 2 補數計算下列各式，並判斷是否溢位：(a) 0110+0111；(b) 1100+1110；(c) 0101+1011。

解答

(a) $0110 + 0111 = 1101$（無進位丟棄）。$6+7$：兩正數相加，結果 $1101_2 = -8+4+1 = -3$ 是負數 $\Rightarrow$ **溢位**✗（真正的 13 超過上限 7）。

(b) 1100+1110 = 1 1010，丟棄第 4 位得 1010。$-4+(-2)$：兩負數相加，結果 $1010_2 = -8+2 = -6$ 仍是負數 $\Rightarrow$ **正確**✓（$-4-2=-6$）。

(c) $0101+1011=1\,0000$，丟棄得 0000。$5+(-5)$：一正一負**永不溢位**，結果 0 ✓。注意：有進位被丟棄**不代表**溢位——判準是「同號相加變號」，不是「有沒有進位」。

練習 5：2 補數轉換

(a) 求 -6 的 4 位元 2 補數。(b) 求 -12 的 5 位元 2 補數。(c) 10110_2（5 位元 2 補數）是十進位多少?

解答

(a) $+6=0110 \xrightarrow{\text{翻轉}} 1001 \xrightarrow{+1} 1010$。驗算：$-8+2=-6$ ✓

(b) $+12=01100 \xrightarrow{\text{翻轉}} 10011 \xrightarrow{+1} 10100$。驗算：$-16+4=-12$ ✓

(c) MSB 權重 $-2^4=-16$：$-16+4+2=-10$。（或反向：翻轉加一 $\to 01010=10$，故原數為 -10 ✓）

練習 6：用加法器做減法

以 4 位元電路計算 $6-2$：寫出 A、NOT(B)、C_{in} 與逐位計算過程。

解答

$A=6=0110$，$B=2=0010$，NOT$(B)=1101$，$C_{in}=1$：

A		0	1	1	0
NOT(B)		1	1	0	1
C_{in}					1
和	(1)	0	1	0	0

逐位：bit 0：$0+1+1=10_2 \to S$=0 進 1；bit 1：$1+0+1=10_2 \to S$=0 進 1；bit 2：$1+1+1=11_2 \to S$=1 進 1；bit 3：$0+1+1=10_2 \to S$=0 進 1（丟棄）。結果 $0100_2=4$ ✓（一正一負相加，丟棄進位、無溢位）。

練習 7：全加器公式

只用真值表驗證 $C_{out}=(A\wedge B)\vee(C_{in}\wedge(A\oplus B))$ 與 K-map 求出的 $(C_{in}\wedge B)\vee(A\wedge B)\vee(C_{in}\wedge A)$ 等價。

解答

C_{in}	A	B	$A \oplus B$	$C_{in} \wedge (A \oplus B)$	式 1	式 2
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	1	1

（式 1 $= (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$；式 2 為三項版本。）兩欄完全相同 ✓。注意最後一列：A=B=1 時 $A \oplus B = 0$，但 $A \wedge B = 1$ 撐住了輸出——這就是 $\vee$ 能換成 $\oplus$ 的原因。

練習 8：定點數

用「2 整數位 + 2 小數位」的定點格式：(a) 表示 2.5；(b) 1.3 能精確表示嗎？最接近的值是？(c) 這個格式的範圍與精度各是多少？

解答

(a) $2.5 = 2 + 0.5 = 2^1 + 2^{-1} = 10.10$。

(b) 不能。可表示的值只有 0.25 的倍數；1.3 介於 1.25 (01.01) 與 1.50 (01.10) 之間，最接近是 1.25（誤差 0.05）。

(c) 範圍：$0 \sim 3.75$；精度（增量）：$2^{-2} = 0.25$。增加小數位可提高精度、增加整數位可擴大範圍——位元總數固定時兩者互相排擠，這正是定點數的根本取捨。

練習 9：浮點數編碼

用第 3 章的 S(1)–E(3)–M(4) 格式（bias = 3）：(a) 編碼 5.5；(b) 解碼 1 011 1000。

解答

(a) $5.5 = 101.1_2$，正規化：$1.011_2 \times 2^2$。符號 $S = 0$；偏移指數 $= 2 + 3 = 5 = 101_2$；尾數（去掉前導 1）$= 0110$。

$$\Rightarrow \ 0\,101\,0110$$

驗算：$(2^0 + 2^{-2} + 2^{-3}) \times 2^{5-3} = 1.375 \times 4 = 5.5$ ✓

(b) $S = 1$（負）；偏移指數 $011_2 = 3 \Rightarrow$ 實際指數 $3 - 3 = 0$；尾數 $1000 \Rightarrow 1.1000_2 = 1.5$。

$$(-1)^1 \times 1.5 \times 2^0 = -1.5$$

練習 10：MUX 與 DEMUX

(a) 8 對 1 多工器需要幾個選擇位？(b) 寫出 2 對 1 MUX 的布林式並化成最少的閘。(c) 用 MUX 的觀點解釋 Hack ALU 的 f 控制位。

解答

(a) $\log_2 8 = 3$ 個。

(b) $\text{out} = (\neg s \wedge i_0) \vee (s \wedge i_1)$：1 NOT + 2 AND + 1 OR = 4 個閘（2 輸入閘的最簡形式）。

(c) ALU 內部**同時**算出 $x + y$（Add16）與 $x \,\&\, y$（And16），f 是一排 16 位元 2 對 1 MUX 的選擇位，從兩個現成結果中挑一個往下送——「先全算、再挑選」是組合電路的常見模式。

練習 11：Hack ALU 控制位推導

不查表，推導出讓 Hack ALU 輸出下列函數的控制位：(a) 常數 0；(b) $\neg x$；(c) $y - x$。

解答

(a) 0：把兩個輸入都歸零後 AND（或加）即可：$zx{=}1, nx{=}0, zy{=}1, ny{=}0, f{=}1$（或 0），$no{=}0$ $\Rightarrow$ 101010。（$0 + 0 = 0$ ✓）

(b) $\neg x$：保留 x、把 y 變成全 1（$1111\ldots = -1$，歸零再取反），AND 之（$t \wedge 1 = t$），最後取反：$zx{=}0, nx{=}0, zy{=}1, ny{=}1, f{=}0, no{=}1 \Rightarrow 001101$。追蹤：$x \wedge 1111 = x \xrightarrow{no} \neg x$ ✓

(c) $y - x$：對稱於例題 4.3 的 $x - y$，把 n 施在 y 上：$zx{=}0, nx{=}0, zy{=}0, ny{=}1, f{=}1, no{=}1 \Rightarrow$ 000111。驗證：$x + \neg y = x + (-y - 1) = x - y - 1$，取反：$-(x - y - 1) - 1 = y - x$ ✓

練習 12：溢位觀念題

一個程式在 64 位元機器上把兩個大整數相加結果正確，搬到 32 位元機器卻輸出負數。解釋原因，並提出兩種對策。

解答

原因：兩數之和超過 32 位元 `int` 的上限 $2^{31} - 1$。在 64 位元機器上（或用 64 位元型別時）範圍夠大、結果正確；32 位元下發生「正 + 正 = 負」的 2 補數溢位，高位被丟棄後 MSB 變成 1。

對策：（1）改用更大的固定型別（如 `long long` / `int64_t`），即「假設最壞情況的記憶體大小」；（2）加法前檢查：若 $a >$ `INT_MAX` $- b$ 則會溢位，先行處理（或使用編譯器內建的溢位偵測函數）。

A 附錄：速查表

A.1 2 的冪次表

2^n	值	2^n	值	2^n	值
2^0	1	2^6	64	2^{12}	4096
2^1	2	2^7	128	2^{16}	65,536
2^2	4	2^8	256	2^{20}	1,048,576
2^3	8	2^9	512	2^{24}	16,777,216
2^4	16	2^{10}	1024	2^{31}	2,147,483,648
2^5	32	2^{11}	2048	2^{32}	4,294,967,296
2^{-1}	0.5	2^{-2}	0.25	2^{-3}	0.125
2^{-4}	0.0625	2^{-5}	0.03125	2^{-6}	0.015625

A.2 各表示法對照（4 位元）

位元組合	無號	符號-大小	2 補數
0000	0	$+0$	0
0001	1	1	1
0111	7	7	7
1000	8	-0	-8
1001	9	-1	-7
1111	15	-7	-1

A.3 加法器公式速查

半加器	$S = A \oplus B, \quad C = A \wedge B$
全加器	$S = A \oplus B \oplus C_{in}, \quad C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$
減法	$A - B = A + \neg B + 1$（NOT 每一位、首級 $C_{in} = 1$）
溢位	正 + 正 = 負或負 + 負 = 正 $\Rightarrow$ 錯誤；正 + 負永不溢位
2 補數恆等式	$\neg t = -t - 1, \quad -t = \neg t + 1$

A.4 名詞中英對照

英文	中文	英文	中文
base	基底	sign-magnitude	符號-大小
bit / byte	位元／位元組	1's / 2's complement	1 補數／ 2 補數
MSB / LSB	最高／最低有效位元	overflow / underflow	溢位／下溢
hexadecimal	十六進位	fixed-point	定點數
octal	八進位	floating-point	浮點數
half adder	半加器	mantissa	尾數
full adder	全加器	biased exponent	偏移指數
carry	進位	normalise	正規化
ripple carry adder	漣波進位加法器	ALU	算術邏輯單元
carry-lookahead	進位前瞻	control / select bits	控制位／選擇位
data bus	資料匯流排	multiplexer (MUX)	多工器
bitwise	逐位	demultiplexer	解多工器

A.5 參考資料

- Kira Clements, *COMSM1302 Week 2 lecture slides*（2.1–2.4），University of Bristol.
- Nisan & Schocken, *The Elements of Computing Systems*（nand2tetris），Ch. 2 Boolean Arithmetic（Hack ALU 完整規格與 HDL 實作）.
- IEEE 754 技術標準；Wikipedia：*Single-precision floating-point format.*
- Exploring Binary，*Why 0.1 Does Not Exist in Floating-Point.*
- Wikipedia：*Adder (electronics)*、*Carry-lookahead adder*、*Two's complement.*