計算機組織與 RISC-V CPU 設計 · 第 9 章

計算機算術

二補數整數 / 加減法硬體 / 乘法與 Booth 演算法 / 除法 / IEEE 754 浮點數

摘要

本章討論 ALU 的核心工作: 計算機如何表示並運算數值。整數部分: 二補數表示法、符號延伸、溢位規則、加減法器硬體、無號與 Booth 乘法、復原式除法。浮點部分:IEEE 754 標準的格式 (單精度/雙精度)、偏移指數、隱含位元、特殊值 (無窮大、NaN、非正規數)、捨入模式與浮點運算流程。本章對應 Stallings 第 9 章; 其中的加法器與 ALU 設計將在第 22 章直接以 Verilog 實作, 二補數運算規則是理解 RV32I 算術指令 (第 18 章) 的基礎。

目錄

1 ALU 概觀

ALU 是實際執行算術與邏輯運算的部件; 其他部件 (控制單元、暫存器、記憶體、I/O) 主要是為它「備料」與「收貨」。ALU 接收暫存器提供的運算元, 把結果存回暫存器, 並在旗標 **(flags)** 中回報狀態 (溢位、零、負、進位)。注意:RISC-V 刻意不設旗標暫存器, 比較與分支合一 (`beq`、`blt` 直接比較兩暫存器), 避免旗標成為管線的隱性相依——這是新舊 ISA 設計哲學的有趣對照。

2 整數表示

2.1 符號—大小 vs. 二補數

n 位元符號—大小表示 (最高位為符號) 有兩個缺點: 加減法需依符號分別處理; 有 $+0$ 與 -0 兩個零。因此現代機器一律採二補數 **(two's complement)**:

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

最高位權重為負, 其餘同無號數。32 位元範圍:-2^{31} 到 $2^{31}-1$。

位元組合 **(4** 位元**)**	無號	符號—大小	二補數
0111	7	+7	+7
1000	8	-0	-8
1111	15	-7	-1

取負: 逐位取反再加 1。符號延伸: 加寬時複製符號位 (8 位元 $1111\,1011 = -5$ 延伸為 16 位元 $1111\,1111\,1111\,1011 = -5$)。RV32I 的 `lb`(載入位元組) 自動符號延伸、`lbu` 零延伸, 即源於此。

3 整數算術

3.1 加法、減法與溢位

二補數加法與無號加法用同一個電路: 直接相加、丟棄最高進位。這是二補數勝出的根本原因。減法 $A - B$ 即 $A + (\overline{B} + 1)$: 把 B 逐位取反、進位輸入設 1。

定義 **3.1** (溢位規則)**.** 兩個同號數相加, 若結果符號相反, 則溢位。等價的硬體判準: 最高位的進位輸入 $\oplus$ 進位輸出 $= 1$。異號相加永不溢位。

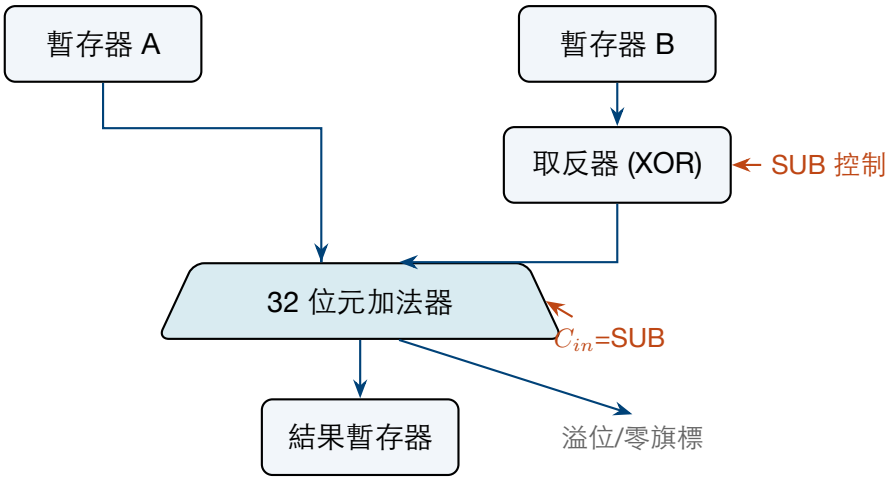

圖 1 加/減法硬體: 同一加法器,SUB=1 時 B 取反且 $C_{in} = 1$

重點觀念

RISC-V 的 `add/sub` 不觸發溢位陷阱 (規格明訂: 忽略溢位, 結果取低 32 位元)。需要檢查時用軟體序列 (如 `slt` 比較符號)。同一加法器也負責 `auipc`、位址計算與 PC+4——你將在第 22 章看到這種共享。

3.2 乘法: 無號長乘法

紙筆長乘法的硬體化: 部分積累加。以暫存器 A(累加)、Q(乘數)、M(被乘數) 實現,n 位元乘法迴圈 n 次:

1. 若 $Q_0 = 1$,$A \leftarrow A + M$;
2. $\{C, A, Q\}$ 整體右移 1 位;
3. 重複 n 次後,$\{A, Q\}$ 即 $2n$ 位元乘積。

3.3 Booth 演算法 (二補數乘法)

Booth 演算法優雅地處理帶號乘法: 掃描乘數相鄰位元對 (Q_0, Q_{-1}):

Q_0	Q_{-1}	動作
0	0	只右移 (算術移位)
0	1	$A \leftarrow A + M$, 再右移
1	0	$A \leftarrow A - M$, 再右移
1	1	只右移

原理: 連續的 1 串 $2^{j+k-1} + \cdots + 2^j = 2^{j+k} - 2^j$——一串 1 只需一次減 (串頭) 一次加 (串尾), 對「1 很多」或帶號的乘數特別高效。

Booth:7×3、4 位元

$M = 0111$,$Q = 0011$,$A = 0000$,$Q_{-1} = 0$。

1. $(Q_0, Q_{-1}) = (1, 0)$:$A = A - M = 1001$; 右移 $\rightarrow A = 1100, Q = 1001, Q_{-1} = 1$

2. $(1,1)$: 只右移 $\rightarrow A=1110, Q=0100, Q_{-1}=1$
3. $(0,1)$:$A=A+M=0101$; 右移 $\rightarrow A=0010, Q=1010, Q_{-1}=0$
4. $(0,0)$: 只右移 $\rightarrow A=0001, Q=0101$

結果 $\{A,Q\}=0001\,0101_2=21$。□(RV32M 的 `mul`/`mulh` 指令, 硬體可用 Booth 編碼陣列乘法器實現。)

3.4 除法

復原式除法 (restoring division) 同樣以移位—減法迴圈實現: 左移 $\{A,Q\}$、$A \leftarrow A-M$; 若 $A<0$ 則商位 0 並復原 (加回 M), 否則商位 1。n 次後 Q 為商、A 為餘數。帶號除法先取絕對值再修正符號。RV32M:`div`/`divu`/`rem`/`remu`; 除以零不陷入例外, 商定義為全 1(-1)、餘數為被除數——又是 RISC-V 務實風格。

4 浮點表示:IEEE 754

4.1 格式

以 $\pm 1.M \times 2^E$ 的正規化科學記號表示實數:

- 符號 S:1 位。
- 偏移指數: 實際指數 + 偏移 (單精度 127、雙精度 1023)。使用偏移而非二補數, 讓浮點數可以按無號整數直接比大小。
- 隱含位元: 正規化數的前導 1 不儲存, 白賺一位精度; 有效位數 24(單)/53(雙)。

	單精度 **(binary32)**	雙精度 **(binary64)**
寬度	1+8+23	1+11+52
偏移	127	1023
正規化範圍 (約)	$10^{\pm 38}$	$10^{\pm 308}$
精度 (十進位)	約 7 位	約 16 位

4.2 特殊值

指數	尾數	意義
全 0	0	± 0
全 0	非 0	非正規數 (次正規): 無隱含 1, 漸進下溢
全 1	0	$\pm\infty$(如 $1/0$)
全 1	非 0	**NaN**(如 $0/0$、$\sqrt{-1}$)

編碼 -0.75 為單精度

$-0.75 = -1.1_2 \times 2^{-1}$。S=1;E $= -1 + 127 = 126 = 0111\,1110_2$;M $= 100\ldots0$。結果:$1\,01111110\,10000000000000000000000_2$ = `0xBF400000`。

4.3 浮點加法與乘法流程

加法/減法:(1) 檢查零;(2) 對齊: 指數小者尾數右移, 直到指數相等;(3) 尾數相加減;(4) 正規化: 調整使尾數回到 $1.x$ 形式, 並捨入; 過程中檢查指數上溢/下溢。

乘法: 指數相加 (減一次偏移)、尾數相乘、正規化與捨入、符號 XOR。

捨入模式 (IEEE 754 定義四種): 就近捨入 (偶數優先, 預設); 向 0; 向 $+\infty$; 向 $-\infty$。就近—取偶避免系統性偏差。

注意

浮點加法不滿足結合律: 對齊時小數被移出而喪失。$(10^{20} + (-10^{20})) + 1 = 1$, 但 $10^{20} + (-10^{20} + 1) = 0$。平行程式重排運算順序會改變結果——數值計算的基本常識, 根源就在本節。RISC-V 的浮點支援為 F(單)/D(雙) 擴展, 含 32 個獨立浮點暫存器 f0–f31 與 fcsr 狀態暫存器; 本教材的 RV32I 實作不含浮點, 但原理相同。

5 本章重點回顧

本章要點

- 二補數: 唯一的零、加減共用同一電路; 減法 = 取反加一; 溢位 = 同號相加變號。
- 乘法: 移位累加;Booth 演算法以「1 串 = 頭減尾加」高效處理帶號數。
- 除法: 移位—減法 (復原式);RISC-V 除零不例外, 回傳定義值。
- IEEE 754:S + 偏移指數 + 隱含 1 尾數; 特殊值 $\pm0, \pm\infty$, NaN, 非正規數。
- 浮點加法 = 對齊 + 相加 + 正規化 + 捨入; 不滿足結合律。

6 複習問題

1. 以 8 位元二補數表示 -73; 再對它符號延伸為 16 位元。
2. 為什麼二補數的加法可以與無號加法共用電路? 溢位偵測有何不同?
3. 以 Booth 演算法計算 $5 \times (-3)$(4 位元), 寫出每一步的 A、Q、Q_{-1}。
4. RV32M 的 `mulh` 為什麼存在?64 位元乘積如何以兩條指令取得?
5. 把 13.625 編碼為 IEEE 754 單精度 (十六進位)。
6. 非正規數解決什麼問題?NaN 與 $\pm\infty$ 分別在何時產生?
7. 說明浮點加法不滿足結合律的原因, 並舉一數值例。

參考資料

[1] W. Stallings, *Computer Organization and Architecture*, 6th ed., Chapter 9.

[2] IEEE Std 754-2019, *IEEE Standard for Floating-Point Arithmetic*.

[3] D. Goldberg, “What Every Computer Scientist Should Know About Floating-Point Arithmetic,” *ACM Computing Surveys*, 1991.