計算機組織與 RISC-V CPU 設計 · 第 10 章

指令集: 特性與功能

機器指令要素 / 運算元型態 / 運算類型 / 位元組序

摘要

指令集是「機器語言」的規格, 是程式設計師 (或編譯器) 所見機器的最完整定義, 也是結構與組織的分界線。本章討論指令集設計的基本議題: 機器指令的要素 (opcode、來源/目的運算元、下一指令位址)、每指令位址數的取捨、運算元的型態 (數值、字元、邏輯資料)、運算的分類 (資料傳輸、算術、邏輯、轉換、I/O、系統控制、控制轉移), 以及位元組序 (endianness)。本章對應 Stallings 第 10 章, 並以 RV32I 作為貫穿範例。

目錄

1 機器指令特性

1.1 指令的要素

每條機器指令包含 (顯式或隱式的) 四類資訊:

1. 運算碼 **(opcode)**: 要做什麼 (加、載入、分支……);
2. 來源運算元參考: 運算的輸入, 可能在主記憶體、暫存器或指令本身 (立即值);
3. 結果運算元參考: 輸出放哪;
4. 下一指令參考: 通常隱含為循序下一條; 分支/跳躍指令顯式指定。

RV32I 的對應

`add x5, x6, x7`:opcode= 加法; 來源 = 暫存器 x6、x7; 目的 = 暫存器 x5; 下一指令 =PC+4(隱含)。`beq x5, x6, label`:opcode= 相等則分支; 來源 =x5、x6; 無目的運算元; 下一指令 = 條件成立時 PC+ 偏移, 否則 PC+4。

1.2 指令表示與組合語言

指令以位元組合表示 (RV32I: 固定 32 位元, 欄位劃分見第 18 章)。人類使用助憶碼 **(mnemonic)**:`ADD`、`LW`、`BEQ`……; 運算元亦以符號表示。組合語言到機器碼的翻譯由組譯器完成 (第 19 章實際操作)。

1.3 每指令的位址數

古典的分類法: 若運算如 $Y = (A - B) \div (C + D \times E)$:

- 三位址:`SUB Y,A,B` ——運算式最自然, 指令較長;
- 二位址:`MOVE Y,A; SUB Y,B` ——一個運算元兼作目的,x86 風格;
- 一位址:`LOAD A; SUB B; STOR Y` ——隱含累加器 AC, 早期機器;
- 零位址: 堆疊機,`PUSH`/`POP`/`ADD` 操作堆疊頂 (JVM 位元組碼)。

位址多 → 指令長但條數少、暫存器利用充分; 位址少 → 指令短而簡單但條數多。**RV32I** 是三位址暫存器型: 算術指令皆為 `op rd, rs1, rs2`。

1.4 指令集設計的根本議題

運算劇目 (多少種運算?)、資料型態、指令格式 (長度、欄位)、暫存器數目、定址模式——這些決策互相牽動, 並直接影響編譯器的難易。歷史的鐘擺:CISC(豐富指令、貼近高階語言)→ RISC(精簡指令、依賴編譯器, 第 13 章)→ 現代的務實混合。

2 運算元型態

- 位址: 本質是無號整數, 參與位址算術。
- 數值: 二補數整數 (第 9 章)、IEEE 754 浮點、(商用機器的) 十進位壓縮 BCD。
- 字元:ASCII(7 位元 + 擴充)、今日的 Unicode/UTF-8。
- 邏輯資料: 把字視為 32 個獨立位元, 供位元運算 (AND/OR/XOR/移位) 操作; 同一串位元, 解讀方式由指令決定——這是機器層的重要觀念: 資料本身沒有型態, 指令賦予它意義。

RV32I 僅支援 32 位元整數運算, 記憶體存取粒度為位元組/半字/字 (`lb/lh/lw`), 帶號/無號由指令區分 (`lb` vs `lbu`); 浮點、向量等以擴展提供——「小核心 + 選配擴展」正是 RISC-V 對「運算劇目」問題的回答。

3 運算類型

3.1 資料傳輸

最基本的指令類別。須指定: 來源與目的 (記憶體/暫存器/堆疊)、資料量、(涉及記憶體時) 定址模式。RV32I:`lw/lh/lb/lhu/lbu`(載入)、`sw/sh/sb`(儲存)、暫存器間搬移以 `mv rd,rs`(實為 `addi rd,rs,0`)。

3.2 算術與邏輯

加減乘除、取負、遞增遞減;AND/OR/NOT/XOR; 移位:

- 邏輯移位: 移出丟棄、補 0(`sll/srl`);
- 算術右移: 複製符號位 (`sra`), 相當於帶號除以 2 的冪 (向負無窮捨入);
- 旋轉: 位元繞回 (RV32I 基礎集無,B 擴展提供)。

用邏輯運算抽取欄位

從 32 位元指令字 `t0` 抽取 bits[19:15](rs1 欄位):

```
srli t1, t0, 15      # 右移 15 位
andi t1, t1, 0x1F    # 遮罩取 5 位
```

移位 + 遮罩是機器層處理封裝資料的萬用手法; 你在第 23 章寫指令解碼器時,Verilog 的位元切片 `instr[19:15]` 做的是同一件事。

3.3 轉換、I/O 與系統控制

轉換: 型態/編碼變換 (如整數 ↔ 浮點,RISC-V F 擴展的 `fcvt`)。I/O: 專用指令 (隔離式) 或記憶體映射 (RISC-V)。系統控制: 特權指令——RISC-V 的 `csrrw` 等 CSR 指令、`mret`、`wfi`(第 21 章)。

3.4 控制轉移

- 分支 **(branch)**: 條件成立則 PC ← 目標。兩種風格: 條件碼 (x86: 先 `CMP` 設旗標, 再 `Jcc`)vs. 比較與分支合一 (RISC-V:`beq/bne/blt/bge/bltu/bgeu` 直接比較兩暫存器)。
- 跳躍 **(jump)**: 無條件轉移;RV32I 的 `jal`(跳並鏈結)、`jalr`(暫存器間接跳躍)。
- 程序呼叫: 需保存返回位址。歷史作法: 存入暫存器、存入被呼叫端開頭、壓入堆疊 (最通用, 支援遞迴與可重入)。RISC-V:`jal ra, func` 把返回位址放 `ra`(x1), 葉函式免記憶體存取; 非葉函式由軟體把 ra 壓上堆疊 (第 19 章呼叫慣例)。

重點觀念

「呼叫 =`jal` 寫 ra; 返回 =`jalr x0, 0(ra)`」——RISC-V 沒有專用 CALL/RET/PUSH/POP 指令, 以通用指令 + 軟體慣例合成。這是 RISC 哲學 (第 13 章) 的直接體現: 硬體提供正交的原語, 複雜語意交給軟體。

4 位元組序 (Endianness)

多位元組資料在記憶體中如何排列?32 位元值 `0x12345678` 存於位址 400:

位址	大端序 **(big-endian)**	小端序 **(little-endian)**
400	12	78
401	34	56
402	56	34
403	78	12

大端序: 最高有效位元組在最低位址 (IBM 主機、網路位元組序); 小端序: 最低有效位元組在最低位址 (x86、**RISC-V**)。差異影響: 跨機資料交換 (網路協定固定大端)、聯集/指標詭技、十六進位傾印的閱讀。RISC-V 規定記憶體為小端序 (指令提取一律小端), 部分實作可選資料大端模式。

5 本章重點回顧

本章要點

- 指令四要素:opcode、來源、目的、下一指令 (通常隱含 PC+4)。
- 位址數目是指令長度與程式長度的取捨;RV32I 為三位址暫存器型。
- 資料無型態, 指令賦予意義; 帶號/無號、邏輯/算術移位由指令區分。
- 運算分類: 傳輸、算術、邏輯、轉換、I/O、系統控制、控制轉移。
- RISC-V: 比較分支合一、jal/jalr 合成呼叫返回、小端序、記憶體映射 I/O。

6 複習問題

1. 寫出計算 $Y = (A - B)/(C + D \times E)$ 的三位址、二位址、一位址與零位址 (堆疊) 程式。

2. RISC-V 為什麼不需要 PUSH/POP 指令? 堆疊操作如何合成?
3. 條件碼風格與比較分支合一風格各有何優缺點 (提示: 管線相依、指令數)?
4. `sra` 與 `srl` 對 `0xFFFF0000` 右移 4 位的結果各為何?
5. 值 `0xDEADBEEF` 存於小端序機器位址 0x100, 寫出 0x100–0x103 各位元組。
6. 為什麼「資料本身沒有型態」? 以 `lb` 與 `lbu` 讀同一位元組為例說明。

參考資料

[1] W. Stallings, *Computer Organization and Architecture*, 6th ed., Chapter 10.

[2] RISC-V International, *The RISC-V Instruction Set Manual, Volume I*, ver. 20260120.