計算機組織與 RISC-V CPU 設計 · 第 13 章

精簡指令集電腦 (RISC)

指令執行特性 / 大型暫存器檔 / RISC vs. CISC / 從 RISC-I 到 RISC-V

摘要

RISC 是計算機結構史上最重要的思想革命之一。本章從程式實際行為的量測出發——高階語言程式最常做的是簡單賦值、程序呼叫與迴圈——說明為何「讓最常見的事最快」勝過「讓指令集無所不能」; 接著介紹 RISC 的三大支柱: 大量暫存器 (含暫存器窗口與編譯器最佳化)、簡單指令與載入/儲存架構、管線導向設計; 並比較 RISC 與 CISC 的論戰與融合。最後梳理從 Berkeley RISC-I 到 RISC-V 的譜系。本章對應 Stallings 第 13 章。

目錄

1 起點: 程式到底在做什麼

1970–80 年代, 語意鴻溝 (高階語言與機器語言的落差) 推動 CISC: 更豐富的指令、更複雜的定址, 希望「一條指令做一件高階的事」。但對真實程式的動態量測顯示:

- 賦值陳述占最大宗——多數只是簡單搬移;
- 程序呼叫/返回是最耗時的操作——參數傳遞、暫存器保存;
- 運算元絕大多數是純量區域變數;
- 條件分支頻繁——每 3–5 條指令一個分支。

結論: 與其加碼複雜指令 (編譯器反而難以善用), 不如最佳化最常見的簡單操作。這就是 RISC 的誕生邏輯。

2 RISC 三大支柱

2.1 支柱一: 大量暫存器

既然運算元多是區域純量, 就讓它們住在暫存器裡:

- 暫存器窗口 **(register windows)**:Berkeley RISC/SPARC 的作法——每次呼叫切到新窗口, 參數暫存器與呼叫端重疊, 呼叫幾乎零成本; 窗口用盡才溢出到記憶體。
- 編譯器分配 **(compiler-based optimization)**:Stanford MIPS 的作法——固定 32 個暫存器, 由編譯器以圖著色演算法做暫存器分配。RISC-V 承襲此路線:32 個通用暫存器 + 明確的呼叫慣例 (a0–a7 傳參、s 開頭被呼叫者保存、t 開頭呼叫者可用, 第 19 章)。

2.2 支柱二: 簡單指令與載入/儲存架構

- 只有 `load`/`store` 接觸記憶體; 所有運算都在暫存器間進行;
- 指令定長、格式少、欄位位置固定 (第 11 章已見 RV32I 的實踐);
- 定址模式極簡 (基底 + 位移、PC 相對);
- 每條指令的工作量 ≈ 一個管線級能完成的量。

2.3 支柱三: 為管線而生

指令簡單規律 → 每級工作可預測 → 管線不需微碼、危障易分析; 搭配最佳化編譯器排程 (填充載入延遲槽、重排指令避免停頓)。RISC 機器的特徵是硬佈線控制 (第 15 章) 而非微程式控制。

3 RISC vs. CISC

	CISC(如 x86、VAX)	RISC(如 MIPS、Arm、RISC-V)
指令數/格式	多而變長	少而定長
定址模式	豐富 (含記憶體運算元)	極簡 (載入/儲存型)
控制單元	微程式為主	硬佈線
暫存器	較少	32 個以上
每指令語意	高 (一條做很多)	低 (一條做一件)
複雜度所在	硬體	編譯器

論戰的結局是融合: 現代 x86 把 CISC 指令在解碼時翻譯成類 **RISC** 微操作 **(uops)**, 內部核心與 RISC 無異; 而 RISC 陣營也吸納了必要的複雜性 (壓縮指令、SIMD 擴展)。ISA 層的簡潔 (對外) 與微結構的激進 (對內) 各司其職。

重點觀念

RISC 真正的遺產不是「指令少」, 而是量化設計方法: 量測真實工作負載 → 讓常見情形快 → 以編譯器與硬體協同分工。「Make the common case fast」至今仍是體系結構第一定律。

4 從 RISC-I 到 RISC-V

- **RISC-I/II**(Berkeley, 1981–):Patterson 團隊, 暫存器窗口; 衍生 SPARC。
- **MIPS**(Stanford, 1982–):Hennessy 團隊, 編譯器導向; 衍生 MIPS 公司。
- **IBM 801** → POWER/PowerPC;**Arm**(1985–): 嵌入式與行動霸主。
- **RISC-V**(Berkeley, 2010–): 第五代 Berkeley RISC。設計目標: 開放 (免授權)、模組化 (RV32I 小核心 + M/A/F/D/C/... 擴展)、乾淨 (不背歷史包袱: 無延遲槽、無旗標、無窗口)、可教學可商用。2015 年成立基金會 (現 RISC-V International), 今日已是與 x86、Arm 並立的第三大生態。

本教材第二部 (第 17 章起) 將完整剖析 RISC-V, 第三部動手實作——你將親自驗證: 一個學生團隊規模的力量, 足以造出一顆符合工業標準 ISA 的 CPU。這正是 RISC 哲學六十年來最好的注腳。

5 本章重點回顧

本章要點

- 量測驅動設計: 真實程式以簡單賦值、區域純量、頻繁呼叫與分支為主。
- RISC 三支柱: 大量暫存器 (窗口或編譯器分配)、載入/儲存架構、管線導向。
- RISC vs. CISC 已融合:x86 內部轉譯為 uops;RISC 外部保持簡潔 ISA。
- RISC-V = 開放 + 模組化 + 無歷史包袱的第五代 Berkeley RISC。

6 複習問題

1. 高階語言程式的動態量測揭示了哪些事實? 它們如何各自對應到 RISC 的設計決策?
2. 暫存器窗口與編譯器暫存器分配各有何優缺點?RISC-V 選了哪條路?
3. 「載入/儲存架構」的精確含義是什麼? 它如何簡化管線?
4. 為什麼說現代 x86「外皮是 CISC、核心是 RISC」?
5. 列舉 RISC-V 相對於 MIPS 的三項「去包袱」設計, 並說明理由。

參考資料

[1] W. Stallings, *Computer Organization and Architecture*, 6th ed., Chapter 13.

[2] D. Patterson and D. Ditzel, “The Case for the Reduced Instruction Set Computer,” *ACM SIGARCH CAN*, 1980.

[3] K. Asanović and D. Patterson, “Instruction Sets Should Be Free: The Case For RISC-V,” UCB/EECS-2014-146.