計算機組織與 RISC-V CPU 設計 · 第 24 章

單週期 CPU(二): 整合與指令流程

頂層資料路徑 / PC 邏輯 / 寫回選擇 / 每類指令的資料流 / 時脈與關鍵路徑

摘要

本章把第 23 章的積木接成一顆完整的 CPU: 定義頂層介面、畫出完整資料路徑、寫出 PC 更新與寫回選擇邏輯, 然後逐類指令追蹤資料流——R 型、立即值、載入、儲存、分支、jal/jalr、lui/auipc 各自走過哪些多工器與部件。最後分析單週期組織的效能特徵: 時脈週期由最長路徑 (載入指令) 決定, 以及這如何自然地引出第 25 章的管線化。

目錄

1 頂層介面與資料路徑

```
module riscv_single_cycle (
    input  wire        clk, rst_n,
    output wire [31:0] imem_addr,    // -> 指令記憶體
    input  wire [31:0] imem_data,
    output wire        dmem_read, dmem_write,
    output wire [31:0] dmem_addr,    // -> 資料記憶體
    output wire [2:0]  dmem_funct3,
    output wire [31:0] dmem_wdata,
    input  wire [31:0] dmem_rdata,
    output wire [31:0] dbg_pc        // 除錯觀察
);
```

程式 1 riscv_single_cycle.v: 介面

CPU 核心只暴露兩組記憶體埠——把記憶體放在核外 (top 模組接線), 日後換成快取或匯流排介面 (第 3、4 章) 都不必改核心。

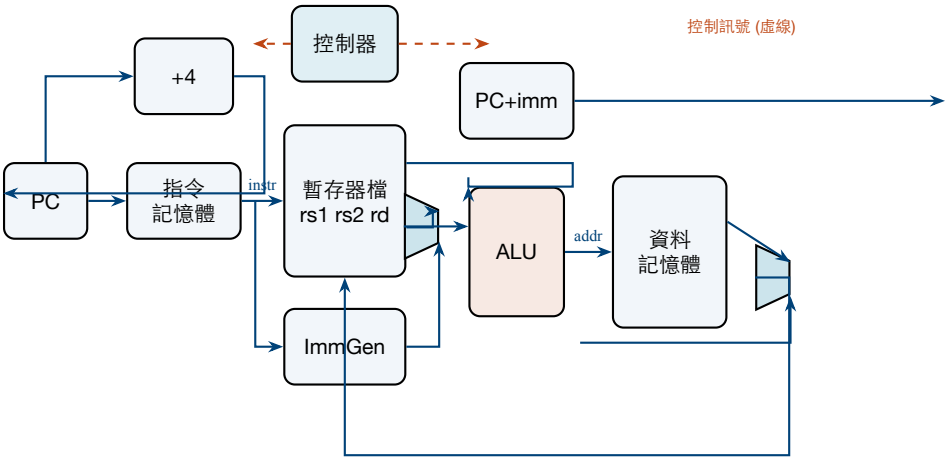

圖 1 單週期資料路徑總覽 (細節見原始碼)

2 三段關鍵邏輯

2.1 PC 更新

```
wire [31:0] pc_plus4      = pc + 32'd4;
wire take_branch          = branch & br_taken;
wire [31:0] branch_target = pc + imm;                  // B/J 型共用
wire [31:0] jalr_target   = (rs1_data + imm) & ~32'b1; // I 型, 清 bit0

assign pc_next = jump        ? branch_target :  // jal
                 jalr        ? jalr_target   :  // jalr
                 take_branch ? branch_target :  // beq/bne/...
                               pc_plus4;

always @(posedge clk or negedge rst_n)
    if (!rst_n) pc <= 32'b0;  else pc <= pc_next;
```

程式 2 PC 邏輯

四路優先選擇正好對應第 10 章「下一指令參考」的四種來源。注意 jal 與分支共用 pc+imm 加法器——立即值產生器已經依格式 (J/B) 給出正確的 imm。

2.2 ALU 輸入選擇

```
wire [31:0] alu_a = alu_a_pc ? pc  : rs1_data;   // auipc 用 PC
wire [31:0] alu_b = alu_src  ? imm : rs2_data;   // I/S 型用 imm
```

2.3 寫回選擇

```
assign wb_data = (wb_sel == 2'b01) ? dmem_rdata :  // 載入
                 (wb_sel == 2'b10) ? pc_plus4   :  // jal/jalr
                 (wb_sel == 2'b11) ? imm        :  // lui
                                     alu_y;        // 其餘
```

3 逐類指令的資料流

add x5,x6,x7	IMEM→ 解碼 → 讀 x6,x7→ALU(加)→wb_sel=ALU→ 寫 x5;PC+4
addi x5,x6,-1	同上, 但 alu_src=1:ALU 的 B 輸入改走 ImmGen(I 型符號延伸)
lw x5,8(x6)	讀 x6→ALU 算 EA=x6+8→dmem 讀 →wb_sel=MEM→ 寫 x5。走完最長的一條鏈
sw x7,8(x6)	ALU 算 EA;rs2(x7) 直接接 dmem_wdata;mem_write=1; 不寫暫存器
beq x5,x6,L	讀 x5,x6→branch_unit 比較; 同時 PC+imm 備妥;taken 則 pc_next 改向
jal ra,L	PC+imm→pc_next;PC+4→ 寫 ra(wb_sel=PC+4)
jalr x0,0(ra)	rs1+imm 清 bit0→pc_next;PC+4 寫入 x0 被硬體丟棄
lui x5,0x12345	ImmGen 給 0x12345000;wb_sel=IMM 直接寫回——ALU 完全閒置
auipc x5,0x1	alu_a_pc=1:ALU 算 PC+0x1000→ 寫回

驗證快照

第 26 章的 testbench 執行 81 條指令的整合測試 (算邏、記憶體、分支、呼叫), 單週期版在第 **98** 個週期寫入成功魔數 `0xC0DE600D`; 暫存器傾印與手算期望值完全一致 (例如 `srl x12, -16, 3` = `0x1FFFFFFE`、`lh` 符號延伸 `0xFFFF8000`)。

4 效能分析: 為什麼要管線化

單週期的時脈週期 τ 必須容納最慢指令的整條組合路徑。以載入為例:

$$\tau \geq t_{IMEM} + t_{DEC/RF} + t_{ALU} + t_{DMEM} + t_{MUX} + t_{setup}$$

假設各段 2/1.5/2/2.5/0.5 ns, 則 $\tau \geq 8.5$ ns(約 118 MHz); 但 R 型只需約 6 ns、分支約 5 ns——所有指令都被最慢者拖累 (CPI=1, 但 τ 大)。對照第 12 章公式 $T = I_c \times \text{CPI} \times \tau$: 管線化把 τ 降到單級延遲 (約 2.5 ns),CPI 略升 (危障停頓), 總體大勝——這正是第 25 章的任務。

另一個代價: 單週期中每個部件每週期只用一次, 加法器就要三個 (ALU、PC+4、PC+imm), 資源利用率低; 多週期/管線設計能分時共用。

5 本章重點回顧

本章要點

- 頂層 = 積木 + 三段膠水邏輯:PC 四路選擇、ALU 兩輸入多工、寫回四路多工。
- 每類指令啟用不同的控制訊號組合, 在同一張資料路徑上走不同的「高亮路徑」。
- 單週期:CPI=1、但 τ 由載入指令的最長鏈決定; 資源不可分時共用。
- 核心只暴露記憶體埠, 記憶體在頂層接線——為未來接快取/匯流排留下介面。

6 練習

1. 追蹤 `sw x7, 12(x2)` 在資料路徑上的完整資料流, 列出所有活躍控制訊號的值。
2. 若把 wb_sel=IMM(lui 專用路徑) 移除, 改讓 lui 走 ALU(A=0,B=imm), 需要改哪些訊號? 有何優缺點?
3. 用本章的段延遲數字, 計算 R 型、載入、分支各自的最短週期; 若指令組成為 40%/25%/20%/15%(R/載入/分支/儲存), 單週期與「理想變週期」機器的平均每指令時間各是多少?
4. 為 CPU 增加 `ebreak` 停機: 解碼到 ebreak 時凍結 PC, 輸出 halted 訊號。
5. 把 imem/dmem 合併成單一埠記憶體會發生什麼? 這是哪一類危障?(第 12 章)

參考資料

[1] D. Patterson and J. Hennessy, *Computer Organization and Design, RISC-V Edition*, Ch. 4.3–4.4.

[2] S. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V Edition*, Ch. 7.3.

[3] 本教材原始碼:`verilog/rtl/riscv_single_cycle.v`、`top_single_cycle.v`。