計算機組織與 RISC-V CPU 設計 · 第 23 章

單週期 CPU(一): 資料路徑模組

設計規劃 / ALU / 暫存器檔 / 立即值產生器 / 記憶體 / 控制器

摘要

從本章起, 我們動手建造一顆完整的 RV32I 處理器。策略是「先單週期、後管線」: 單週期機器每個時脈執行完一整條指令, 沒有危障, 是理解資料路徑的最佳起點。本章完成所有積木模組的 Verilog 實作與原理講解:ALU、暫存器檔、立即值產生器、分支比較器、主控制器、指令/資料記憶體。每個模組都對應前面章節的理論:ALU 對應第 9 章、立即值對應第 11 章、控制訊號對應第 15 章。完整原始碼位於專案的 `verilog/rtl/` 目錄, 已以 Icarus Verilog 通過模擬驗證。

目錄

1 設計規劃

1.1 規格

- ISA:RV32I(不含 fence/ecall/ebreak 的陷阱行為, 視為 NOP);
- 組織: 單週期, 哈佛式介面 (指令記憶體與資料記憶體分開, 對照第 4 章);
- 記憶體: 各 4 KiB; 指令 ROM 由 `$readmemh` 載入;
- 驗證: 組合語言測試程式 + 自動判定 PASS/FAIL 的 testbench(第 26 章)。

1.2 模組分解

檔案	模組	職責 (對應理論章)
alu.v	alu	十種運算 (第 9 章)
regfile.v	regfile	32×32 暫存器、x0 恆零 (第 17 章)
imm_gen.v	imm_gen	六種格式立即值抽取 (第 11 章)
branch_unit.v	branch_unit	六種分支條件 (第 18 章)
control.v	control	主解碼器 +ALU 控制 (第 15 章)
imem.v / dmem.v	imem/dmem	指令 ROM / 資料 RAM(位元組/半字/字)
riscv_single_cycle.v	—	頂層資料路徑接線 (第 24 章)

2 ALU

第 9 章的加減法器、移位器、比較器在此合體。介面: 兩個 32 位元輸入、4 位元運算碼、輸出與 zero 旗標:

```
always @(*) begin
    case (alu_op)
        ALU_ADD : y = a + b;
        ALU_SUB : y = a - b;
        ALU_SLL : y = a << b[4:0];
        ALU_SLT : y = ($signed(a) < $signed(b)) ? 32'd1 : 32'd0;
        ALU_SLTU: y = (a < b) ? 32'd1 : 32'd0;
        ALU_XOR : y = a ^ b;
        ALU_SRL : y = a >> b[4:0];
        ALU_SRA : y = $signed(a) >>> b[4:0];
        ALU_OR  : y = a | b;
        ALU_AND : y = a & b;
        default : y = 32'b0;
    endcase
end
assign zero = (y == 32'b0);
```

程式 1 alu.v(核心)

要點:(1) `$signed()` 讓 < 與 >>> 依二補數語意工作 (第 9 章);(2) 移位量取 `b[4:0]`——正是規格「取 rs2 低 5 位」的直接翻譯;(3) 加法與減法會由合成器共享同一個加法器 (B 取反 + 進位, 圖見第 9 章)。

3 暫存器檔

```
reg [31:0] regs [0:31];
always @(posedge clk)
    if (we && (wa != 5'd0)) regs[wa] <= wd;   // x0 永遠不被寫

assign rd1 = (ra1 == 5'd0) ? 32'b0 : (bypass1 ? wd : regs[ra1]);
assign rd2 = (ra2 == 5'd0) ? 32'b0 : (bypass2 ? wd : regs[ra2]);
```

程式 2 regfile.v(核心)

2 讀 1 寫:R 型指令一個週期要讀 rs1、rs2 並寫 rd, 所以需要兩個非同步讀埠、一個同步寫埠 (第 12 章的結構危障對策)。x0 恆零以「讀時遮罩 + 寫時忽略」實現。BYPASS 參數提供「寫穿旁路」——管線版必需 (WB 與 ID 同週期存取同一暫存器)、單週期版必須關閉 (會形成組合迴圈; 見第 22 章陷阱 4——這是我們開發時真實遇到並修正的 bug)。

4 立即值產生器

第 11 章六種格式的立即值拼接, 一字不差地翻譯成 Verilog:

```
case (opcode)
    OP_IMM, OP_LOAD, OP_JALR:  // I 型
        imm = {{20{instr[31]}}, instr[31:20]};
    OP_STORE:                  // S 型
        imm = {{20{instr[31]}}, instr[31:25], instr[11:7]};
    OP_BR:                     // B 型 (imm[0] 恆 0)
        imm = {{19{instr[31]}}, instr[31], instr[7],
               instr[30:25], instr[11:8], 1'b0};
    OP_LUI, OP_AUIPC:          // U 型
        imm = {instr[31:12], 12'b0};
    OP_JAL:                    // J 型
        imm = {{11{instr[31]}}, instr[31], instr[19:12],
               instr[20], instr[30:21], 1'b0};
    default: imm = 32'b0;
endcase
```

程式 3 imm_gen.v(核心)

注意所有格式的符號延伸都複製 `instr[31]`——第 11 章說的「符號位固定在 bit 31」讓這段程式碼如此整齊。

5 分支比較器

第 12 章提到「把分支判定提前」; 我們把它獨立成模組, 單週期與管線版共用:

```
case (funct3)
    3'b000: taken = (rs1 == rs2);                  // beq
    3'b001: taken = (rs1 != rs2);                  // bne
    3'b100: taken = ($signed(rs1) <  $signed(rs2)); // blt
    3'b101: taken = ($signed(rs1) >= $signed(rs2)); // bge
    3'b110: taken = (rs1 <  rs2);                  // bltu
    3'b111: taken = (rs1 >= rs2);                  // bgeu
    default: taken = 1'b0;
endcase
```

程式 4 branch_unit.v(核心)

6 主控制器

第 15 章的「硬佈線控制」實體: 輸入 opcode/funct3/funct7[5], 輸出十組控制訊號。全表如下 (這張表就是第 15 章「控制單元真值表」的 RV32I 版):

指令類	RegWrite	ALUSrc	WBSel	MemRd	MemWr	Branch	Jump	Jalr	ALUAPc
R 型	1	0	ALU	0	0	0	0	0	0
I 型算邏	1	1	ALU	0	0	0	0	0	0
載入	1	1	MEM	1	0	0	0	0	0
儲存	0	1	—	0	1	0	0	0	0
分支	0	0	—	0	0	1	0	0	0
lui	1	—	IMM	0	0	0	0	0	0
auipc	1	1	ALU	0	0	0	0	0	1
jal	1	—	PC+4	0	0	0	1	0	0
jalr	1	1	PC+4	0	0	0	0	1	0

ALU 控制子電路把 funct3/funct7[5] 譯成 4 位元 alu_op; 唯一的細節是 funct3=000 時,R 型看 funct7[5] 分 add/sub,I 型永遠是 **addi**(沒有 subi, 第 18 章):

```
case (funct3)
    3'b000: arith_op = (funct7b5 && opcode == OP_R) ? ALU_SUB : ALU_ADD;
    3'b001: arith_op = ALU_SLL;
    3'b010: arith_op = ALU_SLT;
    3'b011: arith_op = ALU_SLTU;
    3'b100: arith_op = ALU_XOR;
    3'b101: arith_op = funct7b5 ? ALU_SRA : ALU_SRL;
    3'b110: arith_op = ALU_OR;
    default: arith_op = ALU_AND;
endcase
```

程式 5 control.v:ALU 控制

7 記憶體

imem: 組合讀 ROM,`addr[31:2]` 做字索引 (PC 對齊 4)。**dmem**: 同步寫、組合讀;lb/lh/lbu/lhu 的位元組抽取與延伸、sb/sh 的位元組致能, 完整實作了第 18 章的載入/儲存表:

```
3'b000: case (boff)                          // lb: 選位元組+符號延伸
    2'd0: rdata = {{24{word[7]}},  word[7:0]};
    2'd1: rdata = {{24{word[15]}}, word[15:8]};
    ...
3'b100: case (boff)                          // lbu: 零延伸
    2'd0: rdata = {24'b0, word[7:0]};
    ...
```

程式 6 dmem.v: 讀取端 (節錄)

8 本章重點回顧

本章要點

- 模組化分解:ALU/暫存器檔/立即值/分支/控制/記憶體, 各自對應一章理論。
- ALU 用 $signed 處理帶號比較與算術右移; 移位量取低 5 位。
- 暫存器檔 2 讀 1 寫、x0 恆零; 旁路參數化 (管線開、單週期關)。
- 立即值產生器 = 六種格式的位元拼接; 符號位恆為 instr[31]。
- 主控制器是純組合真值表;ALU 控制看 funct3+funct7[5]。

9 練習

1. 為 ALU 增加一個「b 直通」運算 (y=b), 說明它能簡化哪條指令的資料路徑。
2. 修改 regfile 為 3 讀埠: 哪類指令集擴展會需要? 硬體代價是什麼?
3. 手算 `0xFE628CE3`(beq x5,x6,-8) 經 imm_gen 後的輸出值, 驗證 B 型拼接。
4. dmem 目前不支援未對齊存取; 為 lw 加上位址對齊檢查訊號 misaligned。
5. 為 control.v 加入非法指令偵測輸出 illegal(所有未知 opcode)。

參考資料

[1] S. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V Edition*, Ch. 7.

[2] D. Patterson and J. Hennessy, *Computer Organization and Design, RISC-V Edition*, Ch. 4.

[3] 本教材原始碼:`verilog/rtl/*.v`(以 Icarus Verilog 13.0 驗證)。