計算機組織與 RISC-V CPU 設計 · 第 18 章

RV32I 基礎指令集詳解

六種格式 / 40 條指令逐一剖析 / 編碼、語意與範例

摘要

本章是 RISC-V 32 位元指令集的完整參考: 依類別逐一剖析 RV32I 的全部指令——算術邏輯 (暫存器型與立即值型)、移位、比較、上位立即值 (lui/auipc)、載入/儲存、條件分支、跳躍與鏈結 (jal/jalr)、以及系統指令 (ecall/ebreak/fence)。每條指令給出格式、編碼欄位、精確語意與組合語言範例。本章依據 RISC-V 官方規格 RV32I 2.1 版 (ISA Manual 20260120) 撰寫, 是第 23 章解碼器與資料路徑實作時的權威對照表。

目錄

1 總覽

RV32I 共 **40** 條指令 (含 fence/ecall/ebreak), 使用第 11 章介紹的六種格式。全部指令一覽:

類別	格式	指令
暫存器算邏	R	add, sub, sll, slt, sltu, xor, srl, sra, or, and
立即值算邏	I	addi, slti, sltiu, xori, ori, andi, slli, srli, srai
上位立即值	U	lui, auipc
載入	I	lb, lh, lw, lbu, lhu
儲存	S	sb, sh, sw
條件分支	B	beq, bne, blt, bge, bltu, bgeu
跳躍	J / I	jal, jalr
系統/序	I	ecall, ebreak, fence

主要 opcode(bits[6:0];bits[1:0] 恆為 11):

OP-IMM = 0010011	OP = 0110011	LOAD = 0000011	STORE = 0100011
BRANCH = 1100011	JAL = 1101111	JALR = 1100111	LUI = 0110111
AUIPC = 0010111	SYSTEM = 1110011	MISC-MEM = 0001111	

2 整數運算指令

2.1 暫存器—暫存器 (R 型,opcode=0110011)

語意皆為 `rd` ← `rs1` op `rs2`; 由 funct3 與 funct7 區分運算:

指令	**funct7**	**funct3**	語意
add	0000000	000	rd = rs1 + rs2(溢位忽略)
sub	**0100000**	000	rd = rs1 − rs2
sll	0000000	001	rd = rs1 ≪ rs2[4:0](邏輯左移)
slt	0000000	010	rd = (rs1 < rs2) ? 1 : 0(帶號比較)
sltu	0000000	011	rd = (rs1 < rs2) ? 1 : 0(無號比較)
xor	0000000	100	rd = rs1 ⊕ rs2
srl	0000000	101	rd = rs1 ≫ rs2[4:0](邏輯右移, 補 0)
sra	**0100000**	101	rd = rs1 ≫ rs2[4:0](算術右移, 補符號)
or	0000000	110	rd = rs1 | rs2
and	0000000	111	rd = rs1 & rs2

注意兩件事:(1) **funct7** 的 **bit 30** 是 add/sub 與 srl/sra 的唯一區分——解碼器只需看這一位; (2) 移位量取 **rs2** 的低 **5** 位 (0–31)。

sltu 的妙用: 進位與零測試

```
# 64 位元加法 (a1:a0) + (a3:a2) -> (a5:a4)
add  a4, a0, a2      # 低字相加
```

```
sltu t0, a4, a0      # 進位: 結果 < 運算元 => 有進位
add  a5, a1, a3      # 高字相加
add  a5, a5, t0      # 加進位
# 零測試: seqz rd,rs = sltiu rd,rs,1
sltiu t1, a4, 1      # t1 = (a4==0) ? 1 : 0
```

沒有旗標暫存器的 RISC-V 以 sltu 顯式求進位——相依關係全在暫存器裡, 對亂序執行透明 (第 14 章)。

2.2 立即值運算 (I 型,opcode=0010011)

語意 `rd` ← `rs1` op imm;**imm** 為 **12** 位元帶號數**,** 符號延伸至 **32** 位元 (範圍 $-2048 \sim +2047$):

指令	**funct3**	語意
addi	000	rd = rs1 + imm(addi rd,rs,0 即 mv;addi x0,x0,0 即 nop)
slti	010	rd = (rs1 < imm) ? 1:0(帶號)
sltiu	011	rd = (rs1 < imm) ? 1:0(無號;imm 先符號延伸再視為無號)
xori	100	rd = rs1 $\oplus$ imm(xori rd,rs,-1 即 not)
ori	110	rd = rs1 | imm
andi	111	rd = rs1 & imm
slli	001	rd = rs1 $\ll$ shamt(imm[4:0]= 移位量,imm[11:5]=0000000)
srli	101	rd = rs1 $\gg$ shamt(邏輯;imm[11:5]=0000000)
srai	101	rd = rs1 $\gg$ shamt(算術;imm[11:5]=**0100000**)

沒有 **subi**: 用 addi 的負立即值即可——正交性省下一條指令。

2.3 上位立即值 (U 型)

- `lui rd, imm20`:rd = imm20 $\ll$ 12(低 12 位清 0)。
- `auipc rd, imm20`:rd = **PC** + (imm20 $\ll$ 12)——PC 相對定址的基石, 支撐位置無關碼與大範圍跳轉。

載入 32 位元常數與遠端定址

```
# li t0, 0x12345678 展開為:
lui  t0, 0x12345      # t0 = 0x12345000
addi t0, t0, 0x678    # t0 = 0x12345678
# 注意: 若低 12 位 >= 0x800, addi 會減(符號延伸),
# 組譯器自動把 lui 的值加 1 補償。
# 讀取距 PC 約 +0x1234 處的全域變數:
auipc t1, 0x1         # t1 = PC + 0x1000
lw    t2, 0x234(t1)
```

3 載入與儲存

唯一接觸記憶體的指令 (載入/儲存架構, 第 13 章)。有效位址 EA = `rs1` + imm12(帶號)。

指令	格式	**funct3**	語意
lb	I	000	rd = 符號延伸 (M[EA][7:0])
lh	I	001	rd = 符號延伸 (M[EA][15:0])
lw	I	010	rd = M[EA][31:0]
lbu	I	100	rd = 零延伸 (M[EA][7:0])
lhu	I	101	rd = 零延伸 (M[EA][15:0])
sb	S	000	M[EA][7:0] = rs2[7:0]
sh	S	001	M[EA][15:0] = rs2[15:0]
sw	S	010	M[EA][31:0] = rs2

重點觀念

為什麼載入分帶號/無號、儲存不分? 因為載入要把窄資料放進 32 位元暫存器, 高位補什麼必須指明 (C 的 `char` 用 lb、`unsigned char` 用 lbu); 儲存只是截斷低位, 無此問題。第 24 章實作記憶體存取單元時, 這張表直接翻譯成位元組選擇與延伸邏輯。

4 條件分支 (B 型,opcode=1100011)

比較 rs1 與 rs2, 成立則 $PC \leftarrow PC + \text{imm13}$(B 型立即值,±4 KiB,2 位元組對齊):

指令	**funct3**	條件
beq	000	rs1 == rs2
bne	001	rs1 != rs2
blt	100	rs1 $<$ rs2(帶號)
bge	101	rs1 $\geq$ rs2(帶號)
bltu	110	rs1 $<$ rs2(無號)
bgeu	111	rs1 $\geq$ rs2(無號)

沒有 bgt/ble? 交換運算元即可:`bgt rs,rt,L` = `blt rt,rs,L`(組譯器偽指令代勞)。與 x0 比較合成零測試:`beqz rs,L` = `beq rs,x0,L`。

5 跳躍與鏈結

- `jal rd, imm21`(J 型):rd ← PC+4;PC ← PC + imm(±1 MiB)。慣例 rd=ra 作函式呼叫;rd=x0 即無條件跳躍 `j`。
- `jalr rd, imm12(rs1)`(I 型):rd ← PC+4;PC ← (rs1+imm) & ∼1。用途: 函式返回 (`ret`=`jalr x0,0(ra)`)、函式指標呼叫、switch 跳表、搭配 auipc 跳到任意 32 位元位址。

呼叫與返回

```
main:
    jal  ra, sum      # 呼叫: ra = 返回位址
    ...               # 返回後從這裡繼續
sum:
    add  a0, a0, a1   # 函式本體 (參數 a0,a1 -> 回傳 a0)
```

```
6     jalr x0, 0(ra)    # 返回 (偽指令: ret)
```

6 系統與記憶體序指令

- `ecall`: 環境呼叫——使用者程式請求作業系統服務 (系統呼叫); 觸發例外, 進入更高特權模式 (第 21 章)。
- `ebreak`: 斷點——交還除錯環境。
- `fence pred, succ`: 記憶體序柵欄: 保證 pred 集合 (R/W/I/O) 中先前的操作, 先於 succ 集合中後續操作對其他 hart/裝置可見 (第 16 章 RVWMO)。單 hart 簡單實作可當 nop。

7 完整編碼範例

手工組譯

`sw x7, 12(x2)`:S 型;imm=12=`0000000 01100`;rs2=7;rs1=2;funct3=010;opcode=0100011。

$$\underbrace{0000000}_{imm[11:5]}\ \underbrace{00111}_{rs2}\ \underbrace{00010}_{rs1}\ \underbrace{010}_{f3}\ \underbrace{01100}_{imm[4:0]}\ \underbrace{0100011}_{op} = \texttt{0x00712623}$$

`beq x5, x6, -8`(往回 8 位元組):imm13=-8=`1 111111 1100 0`;B 型拆位:imm[12]=1, imm[10:5]=111111, imm[4:1]=1100, imm[11]=1。

$$\underbrace{1}_{12}\ \underbrace{111111}_{10:5}\ \underbrace{00110}_{rs2}\ \underbrace{00101}_{rs1}\ \underbrace{000}_{f3}\ \underbrace{1100}_{4:1}\ \underbrace{1}_{11}\ \underbrace{1100011}_{op} = \texttt{0xFE628CE3}$$

8 RV32I 沒有的東西 (刻意)

乘除法 (→M 擴展)、浮點 (→F/D)、原子操作 (→A)、位元旋轉 (→Zbb)、條件搬移 (→Zicond)、延遲槽 (拒絕)、自動增量定址 (拒絕)。「沒有」是特性: 基礎集小, 實作與驗證的最小成本就低; 需要的功能以擴展選配。

9 本章重點回顧

本章要點

- RV32I=40 條指令、六種格式;opcode+funct3+funct7 三級解碼。
- 算邏指令:R 型 (暫存器) 與 I 型 (12 位元帶號立即值);sub/sra 靠 bit 30 區分。
- lui+addi 合成任意常數;auipc 是 PC 相對定址的基石。
- 載入區分帶號/無號延伸; 儲存只截斷。分支比較合一、無旗標。
- jal/jalr + ra 慣例合成呼叫/返回;ecall/ebreak/fence 連接特權層與記憶體模型。

10 複習問題

1. 為什麼 RV32I 不需要 subi、not、neg、mv 指令? 各如何合成?
2. `sltiu rd, rs, 1` 為什麼等於「rs 是否為零」?
3. 手工組譯:`addi x10, x11, -5`、`lw x6, 8(x5)`、`jal x1, +16`。
4. `li t0, 0xFFFFF800` 需要幾條指令?`li t0, 0x7FF` 呢?
5. 寫出以 blt/bge 合成 `ble a0,a1,L` 的方法。
6. lb 與 lbu 讀取值 0x80 的結果各是什麼 (32 位元十六進位)?
7. jalr 的目標為何要 & ∼1?(提示:C 擴展的 2 位元組對齊)

參考資料

[1] RISC-V International, *ISA Manual Vol. I*, ver. 20260120, Ch. “RV32I Base Integer Instruction Set, Version 2.1”.

[2] D. Patterson and A. Waterman, *The RISC-V Reader*, 2017.

[3] RISC-V Reference Card. https://riscv.org/technical/specifications/