

Jim's Dev Blog

RISC-V 指令集架構介紹 - RV32I

📅 2018-05-08 | 📁 [RISC-V](#) | 💬 [0 Comments](#)

RV32I為 32-bit基本整數指令集，有 32個 32-bit暫存器(x0-x31)，總共有 47道指令，以下介紹各個指令的用途與格式。

整數運算指令 (Integer Computational Instructions)

整數暫存器與常數指令 (Integer Register-Immediate Instructions)

指令為暫存器與常數之間的運算

- **ADDI**

addi rd, rs1, simm12

常數部分為 sign-extended 12-bit，會將 12-bit做 sign-extension成 32-bit 後，再與 rs1暫存器做加法運算，將結果寫入 rd暫存器，*addi rd, rs1, 0*可被使用來當做 mov指令。

- **SLTI**

slti rd, rs1, simm12

常數部分為 sign-extended 12-bit，會將 12-bit做 sign-extension成 32-bit 後，再與 rs1暫存器當做 signed number做比較，若 rs暫存器1小於常數，則將數值 1寫入 rd暫存器，反之則寫入數值 0。

- **SLTIU**

sltiu rd, rs1, simm12

常數部分為 sign-extended 12-bit，會將 12-bit 做 sign-extension 成 32-bit 後，再與 rs1 暫存器當作 unsigned number 做比較。若 rs1 暫存器小於常數，則將數值 1 寫入 rd 暫存器，反之則寫入數值 0。

- **ANDI/ORI/XORI**

andi/ori/xori rd, rs1, simm12

常數部分為 sign-extended 12-bit，會將 12-bit 做 sign-extension 成 32-bit 後，再與 rs1 暫存器做 AND/OR/XOR 運算，將結果寫入 rd 暫存器。

- **SLLI/SRLI/SRAI**

slli/srli/srai rd, rs1, uimm5

常數部分為 unsigned 5-bit，範圍為 0~31，為 shift amount，將 rs1 暫存器做 shift 運算，結果寫入 rd 暫存器，SLLI 為 logical 左移，會補 0 到最低位元，SRLI 為 logical 右移，會補 0 到最高位元，SRAI 為 arithmetic 右移，會將原本的 sign bit 複製到最高位元。

- **LUI (Load upper immediate)**

lui rd, uimm20

將 unsigned 20-bit 放到 rd 暫存器的最高 20-bit，並將剩餘的 12-bit 補 0，此指令可與 ADDI 搭配，一起組合出完整 32-bit 的數值。

- **AUIPC(add upper immediate to pc)**

auipc rd, uimm20

unsigned 20-bit 放到最高 20 位元，剩餘 12 位元補 0，將此數值與 pc 相加寫入 rd 暫存器。

整數暫存器與暫存器指令 (Integer Register-Register Instructions)

指令為暫存器與暫存器之間的運算

- **ADD/SUB**

add/sub rd, rs1, rs2

將 rs1 暫存器與 rs2 暫存器做加法/減法運算，將結果寫入 rd 暫存器。

- **SLT/SLTU**

slt/sltu rd, rs1, rs2

將 rs1 暫存器與 rs2 暫存器當做 signed/unsigned number 做比較，若 rs1 暫存器小於 rs2 暫存器，則將數值 1 寫入 rd 暫存器，反之則寫入數值 0。

- **AND/OR/XOR**

and/or/xor rd, rs1, rs2

將 rs1 暫存器與 rs2 暫存器做 AND/OR/XOR 運算，將結果寫入 rd 暫存器。

- **SLL/SRL/SRA**

sll/srl/sra rd, rs1,, rs2

將 rs1 暫存器做 shift 運算，結果寫入 rd 暫存器，rs2 暫存器的最低 5-bit 為 shift amount。

NOP 指令

NOP 指令即為不改變任何暫存器狀態，除了 pc 以外。NOP 指令會被編碼成 *addi x0, x0, 0* 替代。

控制轉移指令 (Control Transfer Instructions)

分別有兩種控制轉移指令，無條件跳躍(Unconditional jumps)與條件分支(Conditional branches)

無條件跳躍 (Unconditional Jumps)

- **JAL (jump and link)**

jal rd, simm21

常數部分為 sign-extended 21-bit，要注意的是此常數必須為 2 的倍數，即最低

位元為 0，因為此道指令編碼的常數位元數只有 20 位元，所以只會將 signed 21-bit 的最高 20 位元放入指令編碼中，跳躍範圍為 $-+1\text{MiB}$ ，同時也會將下一道指令的位址 $pc+4$ 寫入 rd 暫存器中，在標準的 calling convention 中，rd 暫存器會使用 x1。如果只是單純的 jump，並非是呼叫函示需要儲存其返回位址 $pc+4$ ，可用 *jal x0, simm21* 取代。

- **JALR (jump and link register)**

jalr rd, rs1, simm12

常數部分為 sign-extended 12-bit，跳躍的位址為 rs 暫存器加上 sign-extended 12-bit，並把下一道指令的位址 $pc+4$ 寫入 rd 暫存器中。

條件跳躍 (Conditional Branches)

- **BEQ/BNE/BLT/BLTU/BGE/BGEU**

beq/bne/blt/bltu/bge/bgeu rs1, rs2, simm13

常數部分為 sign-extended 13-bit，要注意的是此常數必須為 2 的倍數，即最低位元為 0，因為此道指令編碼的常數位元數只有 12 位元，所以只會將 signed 13-bit 的最高 12 位元放入指令編碼中，跳躍範圍為 $-+4\text{Kib}$ ，BEQ/BNE 將 rs1 暫存器與 rs2 暫存器做相同與不同的比較，若成立則跳躍，BLT/BLTU 將 rs1 暫存器與 rs2 暫存器分別做 signed/unsigned 小於比較，若成立則跳躍，BGE/BGEU 將 rs1 暫存器與 rs2 暫存器分別做 signed/unsigned 大於等於比較，若成立則跳躍，跳躍的位址則為 pc 加上 sign-extended 13-bit。

載入與儲存指令 (Load and Store Instructions)

RV32I 必須使用載入與儲存指令去存取記憶體，前面的運算指令只能夠對暫存器做操作。

- **LW/LH/LHU/LB/LBU**

lw/lh/lhu/lb/lbu rd, rs1, simm12

常數部分為 sign-extended 12-bit，載入位址則為 rs1 暫存器加上 sign-extended 12-bit，LW 為載入 32-bit 資料寫入 rd 暫存器，LH/LHU 為載入 16-bit

資料分別做 unsigned/signed extension 成 32-bit 後寫入 rd 暫存器，LB/LBU 為載入 8-bit 資料分別做 unsigned/signed extension 成 32-bit 後寫入 rd 暫存器。

- **SW/SH/SB**

sw/sh/sb rs2, rs1, simm12

常數部分為 sign-extended 12-bit，儲存位址則為 rs1 暫存器加上 sign-extended 12-bit，SW 為將 rs2 暫存器完整 32-bit 資料寫入記憶體，SH 為將 rs2 暫存器最低 16-bit 資料寫入記憶體，SB 為將 rs2 暫存器最低 8-bit 資料寫入記憶體。

Memory model

定義了一組 FENCE 指令，用來做不同 thread 之間，記憶體的同步。

控制與狀態暫存器指令 (Control and Status Register Instructions)

CSR Instructions

- **CSRRW/CSRRS/CSRRC/CSRRWI/CSRRSI/CSRRCI**

定義了一組 CSR 指令，可用來讀取寫入 CSR。

Timers and Counters

- **RDCYCLE[H]**

rdcycle 用來讀取最低 31-bit cycle CSR，rdcycleh 用來讀取最高 31-bit cycle 數。

- **RDTIME[H]**

用來讀取 time CSR。

- **RDINSTRET**

用來讀取 instret CSR。

Environment Call and Breakpoints

- **ECALL**

使用來呼叫 system call。

- **EBREAK**

Debugger 用來切換進 Debugging 環境。

Reference

[1] [The RISC-V Instruction Set Manual](#)

RISC-V

◀ C語言: ++*p, *p++和 *++p的不同

RISC-V 指令集架構介紹 - Integer ▶
Calling convention

0 Comments **jimdevblog****1 Login** ▾ **Recommend** **Tweet** **Share****Sort by Best** ▾

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

ALSO ON JIMDEVBLOG

C語言: ++*p, *p++和 *++p的不同



1 comment • a year ago

**Edison Tseng** — Thanks for sharing:)**使用 GDB 來偵錯 LLVM**

1 comment • a year ago

**ZHEN WEI** — Thanks for sharing!
How does the clang launch the opt and llc?**編譯 LLVM 與 Clang**

2 comments • a year ago

**tclin** — Thanks. Subscribe  Add Disqus to your site Add Disqus Add© 2019  Jim Lin由 Hexo 強力驅動