RISCV ISA - Documentation pour le projet de IN330

Équipe pédagogique IN330

2025-2026

About

- RISCV is an open instruction set initially developed by Berkeley University, used among others by Western Digital, Alibaba and Nvidia.
- We are using the rv64g instruction set: **R**isc-**V**, 64 bits, **G**eneral purpose (base instruction set, and extensions for floating point, atomic and multiplications), without compressed instructions. In practice, we will use only 32 bits instructions (and 64 bits registers).
- The rest of the documentation is adapted from https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md and https://github.com/jameslzhu/riscv-card/blob/master/riscv-card.pdf

Registers

RISC-V defines various types, depending on which extensions are included: The general registers (with the program counter), control registers, floating point registers (F extension), and vector registers (V extension). We won't use control nor F or V registers.

General registers The RV64I base integer ISA includes 32 registers, named `x0` to `x31`. Each register is a 64-bit integer and can be represented in a C program with a variable of type `int64_t`. **The program counter `PC`** is separate from these registers, in contrast to other processors such as the ARM. The first register, `x0`, has a special function: reading it always returns 0 and writes to it are ignored.

In practice, the programmer doesn't use this notation for the registers. In assembler, they are given standardized names as part of the RISC-V **application binary interface** (ABI) [1].

Register	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
Alias	zero	ra	sp	gp	tp	t0	t1	t2	s0	s1	a0	a1	a2	a3	a4	a5

Register	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30	x31
Alias	a6	a7	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	t3	t4	t5	t6

Instruction encoding

All of the instructions listed here have a 32-bit encoding. They can be represented in a C program using a value of type `uint32_t`.

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

- "imm[x:y]" means "bits x to y from binary representation of imm". "imm[y|x]" means "bits y, then x of imm". U-types are ommited.
- Immediates are encoded in 2's complement.
- In the J-type format the immediate encodes a signed offset in multiples of 2 bytes (from current PC)
- for B-type and J-type, imm[0] is always 0, thus removed from the encoding.

[1] In the official documentation these conventions are given

RV64I Base Integer Instructions - (excerpt for IN330)

Inst	Syntax	format	Opcode	funct3	funct7	Description (C)
`add`	add rd, rs1, rs2	R	`0110011`	`0x0`	`0x00`	`rd = rs1 + rs2`
`sub`	sub rd, rs1, rs2	R	`0110011`	`0x0`	`0x20`	`rd = rs1 - rs2`
`addi`	add rd, rs1, imm	I	`0010011`	`0x0`		`rd = rs1 + imm`
`ld`	ld rd, imm(rs1)	I	`0000011`	`0x3`		`rd = M[rs1+imm]`
`sd`	sd rs2, imm(rs1)	S	`0100011`	`0x3`		`M[rs1+imm] = rs2`
`beq`	beq rs1,rs2, imm	B	`1100011`	`0x0`		`if(rs1 == rs2) PC += imm`
`bne`	bne rs1,rs2,imm	B	`1100011`	`0x1`		`if(rs1 != rs2) PC += imm`
`blt`	blt rs1,rs2,imm	B	`1100011`	`0x4`		`if(rs1 <  rs2) PC += imm`
`bge`	bge rs1,rs2,imm	B	`1100011`	`0x5`		`if(rs1 >= rs2) PC += imm`
`jal`	jal rd,imm	J	`1101111`			`rd = PC+4; PC += imm`

Pseudo-instructions

Pseudoinstruction	Base Instruction(s)	Meaning
`j offset`	`jal x0, offset`	Jump
`li rd, imm`	`addi rd, zero, imm`	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copy register

(The `li` replacement is in reality a bit different).