ImperialViolet

RISC-V assembly (31 Dec 2016)

RISC-V is a new, open instruction set. Fabrice Bellard wrote a Javascript emulator for it that boots Linux here (more info). I happen to have just gotten a physical chip that implements it too (one of these) and what's cool is that you can get the source code to the chip on GitHub.

The full, user-level instruction set is documented but there's a lot of information in there. I wanted a brief summary of things that I could keep in mind when reading disassembly. This blog post is just a dump of my notes; it's probably not very useful for most people! It also leaves out a lot of things that come up less frequently. There's also an unofficial reference card which is good, but still aimed a little low for me.

RISC-V is little-endian and comes in 32 and 64 bit flavours. In keeping with the RISC-V documents, the flavour (either 32 or 64) is called `XLEN` below in the few places where it matters.

For both, `int` is 32 bits. Pointers and `long` are the native register size. Signed values are always sign extended in a larger register and unsigned 8- and 16-bit values are zero extended. But unsigned 32-bit values are sign-extended. Everything has natural alignment in structs and the like, but alignment isn't required by the hardware.

The stack grows downwards and is 16-byte aligned upon entry to a function.

Instructions are all 32 bits and 32-bit aligned. There is a compressed-instruction extension that adds 16-bit instructions and changes the required alignment of all instructions to just 16 bits. Unlike Thumb on ARM, RISC-V compressed instructions are just a short-hand for some 32-bit instruction and 16- and 32-bit instructions can be mixed freely.

There are 31 general purpose registers called x1–x31. Register x0 drops all writes and always reads as zero. Each register is given an alias too, which describes their conventional use:

Register	Alias	Description	Saved by
x0	zero	Zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5–7	t0–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–17	a1–a7	Arguments and return values	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporary	Caller

There are two types of immediates, which will be written as ‘I’ and ‘J’. Type ‘I’ intermediates are 12-bit, signed values and type ‘J’ are 20-bit, signed values. They are always sign-extended to the width of a register before use.

Arithmetic

There's no carry flag, instead one has to use a comparison instruction on the result in order to get the carry in a register.

A `U` suffix indicates an unsigned operation. Note that the multiplication and division instructions are part of the ‘M’ extension, but I expect most chips to have them.

Instruction	Effect	Notes
`ADD dest,src1,src2`	dest = src1 + src2	
`SUB dest,src1,src2`	dest = src1 - src2	
`ADDI dest,src1,I`	dest = src1 + I	
`MUL dest,src1,src2`	dest = src1 × src2	
`MULH[|U|SH] dest,src1,src2`	dest = (src1 × src2) >> XLEN	This returns the high word of the multiplication result for signed×signed, unsigned×unsigned or

		signed×unsigned, respectively
`DIV[U] dest,src1,src2`	dest = src1/src2	There's no trap on divide-by-zero, rather special values are returned. (See documentation.)
`REM[U] dest,src1,src2`	dest = src1%src2	

Bitwise

These should all be obvious:

Instruction
`AND dest,src1,src2`
`OR dest,src1,src2`
`XOR dest,src1,src2`
`ANDI dest,src1,I`
`ORI dest,src1,I`
`XORI dest,src1,I`

Shifts

Instructions here are Shift [Left|Right] [Logical|Arithmetic].

Note that only the minimal number of bits are read from the shift count. So shifting by the width of the register doesn't zero it, it does nothing.

Instruction	Effect	Notes
`SLL dest,src1,src2`	dest = src1 << src2%XLEN	
`SRL dest,src1,src2`	dest = src1 >> src2%XLEN	
`SRA dest,src1,src2`	dest = src1 >> src2%XLEN	
`SLLI dest,src1,0..31/63`	dest = src1 << I	
`SRLI dest,src1,0..31/63`	dest = src1 >> I	
`SRAI dest,src1,0..31/63`	dest = src1 >> I	

Comparisons

These instructions set the destination register to one or zero depending on whether the relation is true or not.

Instruction	Effect	Notes
`SLT dest,src1,src2`	dest = src1<src2	(signed compare)
`SLTU dest,src1,src2`	dest = src1<src2	(unsigned compare)
`SLTI dest,src1,I`	dest = src1<I	(signed compare)
`SLTIU dest,src1,I`	dest = src1<I	(unsigned compare, but remember that the

	immediate is sign-extended first)

Control flow

The JAL[R] instructions write the address of the next instruction to the destination register for the subsequent return. This can be `x0` if you don't care.

Many instructions here take an immediate that is doubled before use. That's because no instruction (even with compressed instructions) can ever start on an odd address. However, when you write the value in assembly you write the actual value; the assembler will halve it when encoding.

Instruction	Effect	Notes
`JAL dest,J`	dest = pc+2/4; pc+=2*J	
`JALR dest,src,I`	dest = pc+2/4; pc=src1+(I&~1)	Note that the least-significant bit of the address is ignored
`BEQ src1,src2,I`	if src1==src2, pc+=2*I	
`BNE src1,src2,I`	if src1!=src2, pc+=2*I	
`BLT src1,src2,I`	if src1<src2, pc+=2*I	signed compare

`BLTU src1,src2,I` if src1<src2, unsigned compare
pc+=2*I

`BGE src1,src2,I` if src1>=src2, signed compare
pc+=2*I

`BGEU src1,src2,I` if src1>=src2, unsigned compare
pc+=2*I

Memory

The suffix on these instructions denotes the size of the value read or written: ‘D’ = double-word (64 bits), ‘W’ = word (32 bits), ‘H’ = half-word (16 bits), ´B’ = byte. Reading and writing 64-bit values only works on 64-bit systems, and `LWU` only exists on 64-bit systems because it's the same as `LW` otherwise.

Alignment is not required, but might be faster. Also note that there's no consistent direction of data flow in the textual form of the assembly: the register always comes first.

Instruction	Effect	Notes
`L[D|W|H|B] dest,I(src)`	dest = *(src + I)	result is sign extended
`L[D|W|H|B]U dest,I(src)`	dest = *(src + I)	result is zero extended
`S[D|W|H|B] src1,I(src2)`	*(src2 + I) = src1	

Other

Instruction	Effect	Notes
`LUI dest,J`	dest = J<<12	“Load Upper Immediate”. The result is sign extended on 64-bit.
`AUIPC dest,J`	dest = pc + J<<12	“Add Upper Immediate to PC”. The result of the shift is sign-extended on 64-bit before the addition.

W instructions

These instructions only exist on 64-bit and are copies of the same instruction without the ‘W’ suffix, except that they operate only on the lower 32 bits of the registers and, when writing the result, sign-extend to 64 bits. They are `ADDW`, `SUBW`, `ADDIW`, `DIV[U]W`, `REM[U]W`, `S[L|R]LW`, `SRAW`, `S[L|R]LIW` and `SRAIW`

Pseudo-instructions

For clarity, there are also a number of pseudo-instructions defined. These are just syntactic sugar for one of the primitive instructions, above. Here's a handful of the more useful ones:

Pseudo-instruction	Translation
nop	addi x0,x0,0
mv dest,src	addi dest,src,0
not dest,src	xor dest,src,-1

seqz dest,src	sltiu dest,src,1
snez dest,src	sltu dest,x0,src
j J	jal x0,J
ret	jalr x0,x1,0
call offset	auipc x6, offset >> 12; jalr x1,x6,offset & 0xfff
li dest,value	(*possibly several instructions to load arbitrary value*)
la dest,symbol	auipc dest, symbol >> 12; addi dest,dest,offset & 0xfff
l[d|w|h|b] dest,symbol	auipc dest, symbol >> 12; l*x* dest,offset & 0xfff(dest)
s[d|w|h|b] src,symbol	auipc dest, symbol >> 12; s*x* dest,offset & 0xfff(dest)

Note that the instructions that expand to two instructions where the first is `AUIPC` aren't quite as simple as they appear. Since the 12-bit immediate in the second instruction is sign extended, it could end up negative. If that happens, the J immediate for `AUIPC` needs to be one greater and then you can reach the value by subtracting from there.