

Discussion 03

Floating Point; RISC-V Intro

Aditya Balasubramanian

`aditbala [at] berkeley [dot] edu`

Announcements

Agenda

- Floating Point
- RISC-V Intro

Floating Point

Floating Point Conversion

Floating Points

| | | |
|------|----------|----------------------|
| 1 | 8 | 23 |
| Sign | Exponent | Significand/Mantissa |

For normalized floats,

$$Value = (-1)^{Sign} * 2^{Exp+Bias} * 1.mantissa_2$$

For denormalized floats,

$$Value = (-1)^{Sign} * 2^{Exp+Bias+1} * 0.mantissa_2$$

$$Bias = -(2^{\# \text{ of exponent bits}} - 1)$$

Decimal -> Floating Point

$$Value = (-1)^{Sign} * 2^{Exp+Bias} * 1.mantissa_2$$

10.75

- Convert to Binary w.r.t the floating point
 - 1010.11
- Shift floating point to match formula format
 - 1.01011 * 2³
- Read Output
 - Mantissa = 01100...0
 - Exp = 3 - Bias

Step Size

- Given a certain exponent, step size is the change in decimal value when we add 1 to the mantissa of binary FP

$$2^{exp+bias} * 1.mantissa \Rightarrow 1M...M.MMMM$$

- Step Size is difference between `1MM...M.MMM0` and `1MM...M.MMM1`
- 2^{-4} in this example

RISC-V

Assembly Basics

Assembly is...

- The direct output of compiled code
- Is not the final form of code
- Is still human-readable
- A set of instructions that can be directly understood by the system, after maybe some minor adjustments
- Built up of a single operation at a time
- Even more dumb than regular computer programs
- Read line by line when executed, except when told not to

Storage: Registers

- On-chip memory
- RV32 has 32 of them numbered x0-x31
- They are all functionally the same but conventionally different
- They're all 32 bits wide
- Anything can be stored in them (no types)
- NOT A VARIABLE

| # | Name | Description | # | Name | Desc |
|--|------|-------------------------------------|-----|------|-----------------|
| x0 | zero | Constant 0 | x16 | a6 | Args |
| x1 | ra | Return Address | x17 | a7 | |
| x2 | sp | Stack Pointer | x18 | s2 | Saved Registers |
| x3 | gp | Global Pointer | x19 | s3 | |
| x4 | tp | Thread Pointer | x20 | s4 | |
| x5 | t0 | Temporary Registers | x21 | s5 | |
| x6 | t1 | | x22 | s6 | |
| x7 | t2 | | x23 | s7 | |
| x8 | s0 | Saved Registers | x24 | s8 | |
| x9 | s1 | | x25 | s9 | |
| x10 | a0 | Function Arguments or Return Values | x26 | s10 | |
| x11 | a1 | | x27 | s11 | |
| x12 | a2 | Function Arguments | x28 | t3 | Temporaries |
| x13 | a3 | | x29 | t4 | |
| x14 | a4 | | x30 | t5 | |
| x15 | a5 | | x31 | t6 | |
| Caller saved registers | | | | | |
| Callee saved registers (except x0, gp, tp) | | | | | |

Register Specifics

There are 4 general categories (and some special!):

1. Argument registers: `a0` - `a7`
2. Return value registers: `a0` , `a1`
3. Saved registers: `s0` - `s11`
4. Temporary registers: `t0` - `t6`

Special registers

- Return address: `ra` NOT RETURN VALUE!!!

Zero: `x0`

Stack pointer: `sp`

Other pointers: `tp` , `gp`

RISC-V Greensheet!

Loads

`l<u>x</u> rd imm(rs1)`

- Loads `n` bits worth of data from memory address: `rs1 + imm` where `rs1` `<u>should</u>` already be a valid address.
- `n` will depend on the instruction: `lb` = 8 bits, `lh` = 16 bits, `lw` = 32 bits, `ld` (not used usually) = 64 bits
- If we have too few bits...
 - Sign-extend
- If we have too many bits...
 - Take the 32-most LSB bits!

Stores

$s_{<u>x</u>} rd\ imm(rs1)$

- Saves n bits worth of data to the memory address: $rs2 + imm$
- Truncates to n LSB bits to be stored

Jump Instructions, `jal`, `jalr`, `j`, `jr`

| | Saves return address | Jump and no return |
|-----------------------|--------------------------------|----------------------|
| PC-relative address | <code>jal ra, label</code> | <code>j label</code> |
| Address in a register | <code>jalr ra, rs1, imm</code> | <code>jr rs1</code> |

`j` and `jr` are shorthand for common combinations of certain instructions and register usages.

```
j label # jal x0, label
jr rs1 # jalr x0, 0(rs1)
```

Thank you