1 Feb 21, 2025: More about RISC-V instructions

Last time, we spoke about three RISC-V instruction types.

Today, we're going to discuss Jump instructions.

1.1 Jump-and-link-register, jalr

- The Jump-and-link-register or jalr is actually an $I-$type instruction, as briefly mentioned during the previous lecture.
- The syntax for this instruction is

  jalr rd rs1 imm

  What does it do?

  - First it saves the current Program Counter, incremented by 4 bytes, into the destination register rd i.e. it sets
    $$R[\text{rd}] = \text{PC} + 4$$
  - Then, it lets us jump unconditionally to the destination of rs1 offset by imm by setting the program counter
    $$\text{PC} = R[\text{rs1}] + \text{imm}$$
- Note: the Jump-register jr instruction is actually a pseudo instruction, which uses jalr as the underlying "true" instruction, just with the destination register rd as $x0$ and with 0 as the imm offset. What does this mean?
  - Since $x0$ is hardwired to zero, effectively we don't store a return address and just unconditionally jumo to $R[\text{rs1}]$ (since imm is zero).
- Since it's an I-type instruction, the 32 bit word for the jalr instruction uses the same format.

  Include figure of the jalr instruction as a 32 bit word

1.2 $U-$ type Instructions

- Recall that we had an issue with $I-$ type instructions in that we could only use 12 bits for the immediate, thus limiting its range of values. The $U-$ type instructions fix this issue.
- The U stands for "upper".
- There are only two $U-$ type instructions: lui and aui.

1.2.1 Load Upper Immediate, lui

- Syntax:

$$\text{lui rd immu}$$

- This has the action of:

$$\begin{aligned} \text{imm} &= \text{immu} << 12 && \text{Bitshift an immediate by 12 places} \\ R[\text{rd}] &= \text{imm} && \text{Store the shifted immediate into rd} \end{aligned}$$

  Note that since the immediate is shifted to the left by 12 places, its 12 least significant bits are now zeros. Thus, this operation clears the first 12 bits of the register rd.

1.2.2 Add Upper Immediate to PC, aui

- Syntax:

$$\text{aui rd immu}$$

- This has the action of:

$$\begin{aligned} \text{imm} &= \text{immu} << 12 && \text{Bitshift an immediate by 12 places} \\ R[\text{rd}] &= \text{PC} + \text{imm} && \text{Store the shifted immediate into rd} \end{aligned}$$

  i.e. it adds an upper immediate to the PC.

- How is this useful? We'll seein the near future.

Note. *Just to avoid confusion, the* u *in* immu *stands for "upper", not "unsigned".*

Note. ***auipc*** *is similar, but different from* ***aui****. We will cover it later.*

1.2.3 U- Type Instruction Format

Include figure of instruction format for lui and auipc.

Where does the load-immediate instruction come from?

lui + **addi** can create any 32-bit value in a register, and using them together allows us to load any immediate:

- **lui**: set upper 20 bits (and zero out lower 12 bits) within a register
- **addi**: set lower 12 bits by adding sign-extended 12 bit immediate to the same register.
- The **li** pseudoinstruction (load immediate) resolves to **lui** + **addi** as needed.

- For example,

$$\textbf{li x10 0x87654321} \implies \begin{cases}\textbf{lui x10 0x87654}\\ \textbf{addi x10 x10 0x321}\end{cases}$$

- However, things are not always so simple. For example, if we're trying to carry out the instruction

  li x10 0xBOBACAFE

  we might naively try to do it as

$$\textbf{li x10 0xBOBACAFE} \implies \begin{cases}\textbf{lui x10 0xBOBAC}\\ \textbf{addi x10 x10 0xAFE}\end{cases}$$

- Unforunately,

  addi

  sign extends and **0xAFE** has a 1 as its topmost bit, so it gets sign-extended to **0x11111AFE**. Thus, when we add it to **0xBOBAC000** we actually get

$$\begin{array}{r}\textbf{0xBOBAC000}\\ +\ \underline{\textbf{0x11111AFE}}\end{array}$$

  which we can calculate by summing the smallest 3 bits separately, summing the the largest 5 bits separately, and then adding them up. Doing so gives us

$$\begin{array}{r}\textbf{0xBOBAC}\\ +\ \underline{\textbf{0x11111}}\\ \textbf{0xBOBAB}\end{array}$$

  and

$$\begin{array}{r}\textbf{0x000}\\ +\ \underline{\textbf{0xAFE}}\\ \textbf{0xAFE}\end{array}$$

  whose sum is **0xBOBA$\underline{\text{B}}$AFE** wherein the lowest entry in the upper-immediate is 1 less than it should be (this problem of the -1 holds in general when the lower 12-bit number is negative).

<u>What's the solution?</u>
If the 12$-$bit immediate is negative, we **proactively add 1** to the upper 20-bit load, and this fixes the issue.

Now that we've discussed unconditional jumps, the next kind of instruction to talk about are conditional jumps, or, **branches**! Before we do so, however, we need to go deeper into the inner workings of RISC-V by discussing what **labels** are.

1.3 Updating the Program Counter (PC)

had to leave class suddenly; will upload updates notes soon