CS-382 Computer Architecture and Organization Fall 2022

Project 2 · Your First CPU!

Lecturer: Shudong Hao *Date: See Canvas*

This project is a group project with two members max. You can certainly choose to work by yourself and it's actually encouraged!

1 Description

1.1 Hardware

Briefly, in this project, you will use Logisim-Evolution to create a sequential datapath, aka your own CPU! It should include all the essential parts we have discussed in class: instruction memory, register file, ALU, data memory, and of course, PC. This CPU doesn't need to be complicated, but needs to be fulfill the following basic requirements:

- At least four general purpose registers in the register file;
- At least be able to do addition and another arithmetic operation of your choice;
- Be able to load the data from memory;
- A sequential implementation of the datapath.

Note: you do not need to implement branching instructions and store instruction; that's for extra credit (see next section).

The past three labs are very useful as a start, and you can certainly reuse them in the project.

1.2 Assembly and Machine Code Design

As you have learned in class, the instructions heavily depend on the actual hardware architecture of the CPU. Therefore, in order to make your CPU work, you have to design your own instruction set architecture. Thinking about how we mapped the instructions to hardware data signals, we can go from the low level machine code to higher level assembly code:

- For each instruction, determine how many bits you need to represent the machine code, and which field will be used for what purpose. We recommend using 8 bits, or multiples of 8 bites;
- Given those instructions, you need to decide their mnemonics, and the operands.

Once you have figured these out, write them down as a manual, so other engineers (wink!) can use your CPU to do great things!

Note: for a basic implementation, we don't need to design a control unit, an ALU control, and a complicated `opcode` field in the machine code. Since we only need to implement two types of calculation, we can just encode one bit in the machine code to decide which operation we want to do.

1.3 Simulation

Next step to test your great innovation is to write a small assembly program in the language you designed on your own. It doesn't have to be complicated like ARM; just a sequence of instructions will be fine. Then, you need to write a small program to translate that assembly into an image file (containing only hexadecimals and addresses), so we can load it to the RAM, and start executing it.

The program is basically a small assembler, and you don't need to check syntax errors. Just write a bunch of rule-based translation, so it can translate each instruction into corresponding machine code in hexadecimal. You can write this program in any of the following languages you like: C, Java, Python, JavaScript, or even ARM assembly.

1.4 A Great Name!

Lastly, give your baby CPU a badass name and be proud of it! Write your CPU's name in the `circ` file.

2 Extra Credits

The Description section is the basic requirement for the project, but this project has so much potential to be fantastic. Extra credits will certainly be offered. Some ideas include (from easier to harder):

- Making your assembler recognize data and text segment so it can generate two image files. They will be loaded into instruction and data memory separately;
- Adding extra fun components, such as LED display to show calculation results;
- Implementing store instruction so we can store the result into data memory;
- Implementing branching related instructions. This needs you be able to maintain a table for all the labels, and your assembler should be able to calculate offsets. The datapath is more complicated too;
- Creating a pipeline version of the datapath (without hazard);

- ...

Just use your imagination and amaze me!

Note: extra credits are only considered when the basic requirements have been fulfilled. The credits will be offered from 20 pts to 50 pts, based on the work.

3 Deliverables

If this is done by two people, each of you need to submit the same copy of the deliverable on Canvas. These files are required for everyone:

- Logisim-Evolution `.circ` file of your CPU, including: name and pledge, partner's name (if done in group), CPU's name;
- A compilable or executable program that can assemble your assembly program into image files;
- A demo program in the assembly language you designed;
- A user manual (PDF file) including:
  - Name of the CPU;
  - Your name (and your partner's name if done in group);
  - A clear and detailed job description of each person in your group. If done by one person then you can skip this part;
  - How to use your assembler program (*e.g.*, what command, what library to link, *etc*), and how to load it to the instruction and/or data memory;
  - The architecture description of your CPU, *e.g.*, how many general purpose registers and how can we refer them in an assembly program, what functions your CPU can do, *etc*;
  - For each instruction implemented, write the general format (such as `ADD Rm,Rn,Rt`), and its binary encoding. You can follow a similar description as in Chapter 3.2 in textbook. When describing binary encoding, you need to specify why you need this number of bits for that field.

You should feel very rewarded after this project! Enjoy!