Building the β machine

October 12th - November 16th , 2022 INFO0012 – Computation Structures

In the last session, you built an ALU (arithmetic–logic unit) and a counter within the Logisim simulation environment. These are two elements of our home-made computer, the β-machine. The next practical sessions will lead you to a fully working computer, made from logic gates up (we are using Logisim blocks, but virtually all elements are explained from logic gates up in the theoretical sessions).

Carefully follow the steps below and save regularly. Do not forget to regularly submit the intermediate steps (on eCampus, in the appropriate space)! Taking a fully working machine and then submit restricted versions as intermediate steps will not be accepted.

Each practical session is associated with one deliverable. You can work from home and you can work faster. Support is only provided by the assistant, each week, on the university premises. You cannot be more than one week late, that is, the work planned for one session has to be delivered on the day (midnight) of the next session. This allows to work from home for one session, ask details to the assistant the next week, and fix a few details. But be careful not to accumulate delays. If you are slow or late, it is expected that you work at home to compensate.

While building your machine, don't forget to regularly test it. **A machine that seems perfect but isn't working will lead to disappointing marks.**

1. Take your ALU from previous session, create a subcircuit with it, and create a nice shape for future inclusion in the beta machine.
2. Create a subcircuit with the register file, as shown in the theoretical sessions on slide 7-8. Only implement registers `r0` to `r4`, as well as `r29` to `r31`. Check that it is working properly by setting values, e.g., 1, 2, 3, 4, into registers 1, 2, 3, 4, respectively, and then outputting registers 4 and 2. Also check register 31.
3. Create a subcircuit with the instruction memory. Use a 1 kB (ROM) memory, and simply ignore bits outside the range. Also notice that the instruction memory outputs 32 bit words, so use a memory of 256 times 32 bits. The 2 least significant bits of address, and the 22 most significant bits are thus ignored (use a splitter to select the appropriate 8 bits).
4. In the main circuit create the program counter (you can refer to the previous session). You are free to create subcircuits for +4 and the register, for aesthetics, but it is not mandatory.
5. Connect the program counter to the instruction memory in the main circuit.
6. Save and submit your Logisim file. This is the deliverable for Week 1.
7. Create a subcircuit for the control logic, and use a ROM, addressed by the 6 bits of the opcode, outputting 12 bits of control.
8. Connect the above circuits so that the `ADD` operation with three registers work as expected (see Slide 7-12).
9. Implement `ADDC` (see Slide 7-15).

10. Program the control logic for all the ALU instructions. Logisim allows to save/load ROM images. If you wish, you can reverse engineer the format and create a script that will create a suitable ROM file for the control logic.

11. Load the sample program (on ecampus) into the instruction memory and check that the correct values appear in the registers. **Note that this program is quite basic and doesn't test all ALU instructions**. You can start testing more deeply your machine by using your own test program (you can get the hexadecimal form of each instructions by using BSIM which is available on ecampus).

12. Save and submit your Logisim file. This is the deliverable for Week 2.

13. Implement `JMP`, `BEQ`/`BNE`. If you used a script, modify it to support these operations.

14. Check that your machine works as expected, using your own test program. This program must be relatively complete.

15. Save and submit your Logisim file with your test program in the instruction memory along with a report listing the instructions used in that program. This is the deliverable for Week 3.

16. Create a circuit for the data memory. Again, 1 kB is largely enough. It is advised to restrict to word addresses only (that is, assume the address will always be a multiple of 4). It has a 32 bits output. Use a logisim RAM with separate load/store ports.

17. Integrate it to you circuit (see Slide 7-21). Once again, modify your script above to augment your control logic with `LD`/`ST` operations, and check that these operations work as expected.

18. Save and submit your Logisim file. Also provide a report with a list of the instructions in your test program along with your script if you made one. This is the deliverable for Week 4 (last deliverable).

The following improvements are **optional**:

1. ALU instructions using three registers are coded on 32 bits using 6 bits for opcode and three times 5 bits for register addresses, thus in total 21 bits, leaving 11 bits unused. Modify the machine so that all three registers ALU instructions behave as before but have two more arguments on bits 0-5 (Sa, 6 bits, in 2s complement) and 6-10 (Sb, 5 bits) that are used to shift operands. For instance, `ADD` becomes

   $$\begin{aligned}\texttt{ADDext(Ra,Rb,Rc,Sa,Sb)}:\ & \text{PC} \longleftarrow \text{PC} + 4\\ & \text{Reg[Rc]} \longleftarrow (\text{Reg[Ra]} << \text{Sa}) + (\text{Reg[Rb]} << \text{Sb})\end{aligned}$$

   if Sa ≥ 0, and

   $$\begin{aligned}\texttt{ADDext(Ra,Rb,Rc,Sa,Sb)}:\ & \text{PC} \longleftarrow \text{PC} + 4\\ & \text{Reg[Rc]} \longleftarrow (\text{Reg[Ra]} >> \text{-Sa}) + (\text{Reg[Rb]} << \text{Sb})\end{aligned}$$

   if Sa < 0.

2. Implement the β machine with a Von Neumann architecture.

3. Save and submit your Logisim file. This is the optional deliverable.