Computer Systems and -architecture

Project 6

1 Ba INF 2014-2015

Bart Meyers
bart.meyers@uantwerpen.be

Don't hesitate to contact the teaching assistant of this course. You can reach him in room M.G.3.17 or by e-mail.

Time Schedule

Projects are solved individually. Projects build on each other, to converge into a unified whole at the end of the semester. At the evaluation moment, you will present your solution by giving a demo and answering some questions.

For all of your projects, you submit a report of the project you made by filling in `verslag.html` completely. A report typically consists of 1000 words and a number of drawings/screenshots. Put all your files in a tgz archive, as explained on the course's website, and submit your report to the exercises on Blackboard.

- Report deadline: **24 August 2015**
- Evaluation and feedback: **31 August 2015**

Project

Read sections 4.1, 4.2, 4.3 and 4.4 of Chapter 4. You can use all Logisim libraries for this assignment.

1. In the previous assignment, we used the ALU OP-codes as instruction OP-codes and added two additional instructions (`lw` and `sw`). Next to these 14 instructions, in this assignment we also support immediate instructions as well as branch and jump instructions.

   We introduce a number of new instructions, including instructions for `jump` and `branch`. Because you should be able to branch, you will have to connect your **program counter** to your datapath so that it can jump to a given address instead of just the next instruction.

   Implement the instructions described in the table below. You already have implemented the R-type instructions and the lw/sw instructions in the previous assignment.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Description
0000-1011				rd				rs				rt				12 R-type Instructions[1]
1100				rs				rt				imm (uns.)				lw: $rt = MEM[$rs+imm]
1101				rs				rt				imm (uns.)				sw: MEM[$rs+imm] = $rt
1110				rs				rt				imm (sign.[2])				bne: $rs!=$rt ? pc := pc+1+imm
1111				rs				immediate (unsigned)						00		ori[4]: $rs = $rs|imm
1111				rs				immediate (unsigned)						01		lui[3,4]: $rs = imm<<6
1111				target address										10		jal: $r15 := pc+1 ; pc := addr"
1111				rs				immediate (signed[2])						11		jr: pc := $rs+imm

[1] R-type instructions from your ALU.

[2] Two's complement.

[3] "Load upper immediate": put the 6-bit immediate in the upper 6 bits.

[4] The `lui` and `ori` instructions can be used together to implement a `li` pseudo-instruction which loads a 16-bit immediate into a register.

2. In order to get all control lines right, you will have to add a **Control Unit** circuit to your datapath.

   - Input is the instruction (16 bits).
   - Outputs are the ALU OP-code as well as all control lines for i.e. the program counter, instruction and data memory, multiplexers and the register file. Choose your control lines wisely: this can make the implementation a lot easier!

   More information on the implementation of a control unit can be found in Section 4.4 of *Computer organization and design*.

3. Similarly you can create an **Immediate** circuit (this is different from the book's datapath):

   - Input is the instruction (16 bits).
   - Output is the immediate value (16 bits), depending on the instruction this will be a 4/6/10-bit value that is unsigned/sign extended/shifted to 16 bits.

4. Once done, your datapath can correctly execute a program written in machine language, as the behaviour of arithmetic, branching and memory operations is now fully implemented! You can use the script `Test.py` as follows (note the `-f` flag to denote the simulation of a full datapath:

   ```
   python Test.py -f <test-file> <circ-file>
   ```

   When testing the full datapath, you can only perform checks at the end of the program. (This is because of branching: it would not make sense to check a register value in the middle of a loop, as it can have a different value in a different iteration of the loop.)

5. To prepare for the next lab session, read section 4.9 of Chapter 4.