Computer Systems and -architecture

Project 6

1 Ba INF 2015-2016

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 in pairs of two students. Projects build on each other, to converge into a unified whole at the end of the semester. During the semester, you will be evaluated three times. At these evaluation moments, you will present your solution of the past projects by giving a demo and answering some questions. You will immediately receive feedback, which you can use to improve your solution for the following evaluations.

For every project, you submit a small report of the project you made by filling in `verslag.html` completely. A report typically consists of 500 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: **December, 16 2015, 23u55**
- Evaluation and feedback: **December, 18 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 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 ("imm" stands for "immediate", "uns" stands for "unsigned" and "sig" stands for "signed"). You already have implemented the R-type instructions and the lw/sw instructions in the previous assignment.

12	11	10	9	8	7	6	5	4	3	2	1	0	Description
0000				rs			rt			imm (uns.)			lw: \$rt = MEM[\$rs+imm]
0001				rs			rt			imm (uns.)			sw: MEM[\$rs+imm] = \$rt
0010				rs			rt			imm (sig.[2])			bne: \$rs!=\$rt ? pc := pc+1+imm
0011				rs			rt			imm (sig.[2])			be: \$rs==\$rt ? pc := pc+1+imm
0100				rs			imm (sig.[2])						bltz: \$rs<0 ? pc := pc+1+imm
0101				rs			imm (sig.[2])						jr: pc := \$rs+imm
0110				target address								0	j: pc := addr"
0110				target address								1	jal: \$r7:= pc+1 ; pc := addr"
0111				rs			imm (uns.)				00		ori[4]: \$rs = \$rs|imm
0111				rs			imm (uns.)				01		andi[4]: \$rs = \$rs&imm
0111				rs			imm (uns.)				10		lui[3,4]: \$rs = imm<<4
0111				rs			imm (uns.)				11		addi: \$rs = \$rs+imm
1000-1111				rd			rs			rt			8 R-type Instructions[1]

[1] R-type instructions from your ALU.

[2] Signed, two's complement.

[3] "Load upper immediate": put the 4-bit immediate in the upper 4 bits.

[4] The `lui` and `ori` instructions can be used together to implement a `li` pseudo-instruction which loads a 8-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 (13 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 (13 bits).
   - Output is the immediate value (8 bits), depending on the instruction this will be a 3/4/6/8-bit value that is unsigned/sign extended/shifted to 8 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.