Computer Systems and -architecture

Project 6

1 Ba INF 2012-2013

Bart Meyers
bart.meyers@ua.ac.be

Don't hesitate to contact the teaching assistants of this course. You can reach them 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 2012, 23u55**
- Evaluation and feedback: **December, 18 2012**

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 10 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`. You will therefore have to alter your **program counter**, and allow it to jump to a given address instead of just the next insruction.

   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.

11	10	9	8	7	6	5	4	3	2	1	0	Description
`000`			rA			rB			funct			8 R-type Instructions[1]
`001`			target address									jal: $r7 := pc+1 ; pc := addr”
`010`			rA			immediate (unsigned)						ori[4]: $rA = $rA|imm
`011`			rA			immediate (unsigned)						lui[3,4]: $rA = imm<<6
`100`			rA			immediate (signed[2])						jr: pc := $rA+imm
`101`			rA			rB			imm (sign.[2])			bne: $rA!=$rB ? pc := pc+1+imm
`110`			rA			rB			imm (uns.)			lw: $rB = MEM[$rA+imm]
`111`			rA			rB			imm (uns.)			sw: MEM[$rA+imm] = $rB

[1] 8 R-type instructions from your ALU. The ALU opcode is given in the funct field.

[2] Two’s complement.

[3] “Load upper immediate”: put the 6-bit immediate in the upper 6 bits (shift left x6 and store in register).

[4] The `lui` and `ori` instructions can be used together to implement a `li` pseudo-instruction which loads a 12-bit immediate into a register.

2. In order to translate from the instruction OP-code to the ALU OP-codes and to get all control lines right, you will have to add a **Control Unit** circuit to your datapath.

   - Input is the instruction OP-code (3 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.

   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 (12 bits).
   - Output is the immediate value (12 bits), depending on the instruction this will be a 3/6/9-bit value that is unsigned/sign extended/shifted to 12 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.