

Computer Systems and -architecture

Project 6

1 Ba INF 2011-2012

Ruben Van den Bossche
ruben.vandenbossche@ua.ac.be

Sam Verboven
sam.verboven@ua.ac.be

Don't hesitate to contact the teaching assistants of this course. You can reach them in room M.G.2.12 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, 18 2011, 23u55**
- Evaluation and feedback: **December, 20 2011**

Project

1. In the previous assignment, we used the ALU OP-codes as instruction OP-codes and added two additional instructions (`lw` and `sw`). In order to add more than 16 instructions to our datapath, we need to alter our OP-codes and instruction set: next to the 14 ALU (R-type) instructions, we also support immediate instructions as well as branch and jump instructions.

We also extend our instructions with 4 additional bits, and therefore now need a 20-bit instruction memory. The size of our data memory remains unaltered at 16 bits. Although, from the hardware perspective, instructions of a *not-a-power-of-two* size is inefficient, it suits our demonstrational needs.

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 (4 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. 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 instruction.

4. Some registers now have a special meaning:

- **\$r0** becomes **\$zero**
- **\$r14** becomes **\$ra**
- **\$r15** becomes **\$sp**

5. Implement the instructions described in the table below. You will have to alter your datapath slightly in order to get the right operation codes for the R-type instructions and the lw/sw instructions you already implemented.

19-16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0000	rs		rt			rd			funct			14 R-type Instructions ¹					
0001	rs		rt			immediate (unsigned)											lui ^{3,5} : \$rt = imm << 8
0010	rs		rt			immediate (unsigned)											ori ⁵ : \$rt = \$rs imm
0011	rs		rt			immediate (signed ²)											addi: \$rt = \$rs + imm
0100	rs		rt			immediate (unsigned)											andi: \$rt = \$rs & imm
0101	rs		rt			immediate (signed ²)											lw: \$rt = MEM[\$rs+imm]
0110	rs		rt			immediate (signed ²)											sw: MEM[\$rs+imm] = \$rt
0111	target address																jump: \$pc = addr
1000	rs		rt			offset (signed ²)											jr: \$pc = \$rs+imm
1001	rs		rt			offset (signed ²)											beq: (\$rs=\$rt) ? \$pc=\$pc+1+imm
1010	rs		rt			offset (signed ²)											bne: (\$rs≠\$rt) ? \$pc=\$pc+1+imm
1011	target address																jal: \$ra = \$pc + 1; \$pc = addr

¹ 14 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 8-bit immediate in the upper 8 bits (shift left x8 and store in register).

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.

6. 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!