Computer Systems and -architecture

Project 5

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: **November, 27 2011, 23u55**
- Evaluation and feedback: **November, 29 2011**

Project

1. Build a circuit that implements a **16-bit program counter (PC)** that selects an instruction in a RAM element of 16-bit words. By default, the PC is increased each clock cycle, and the next instruction is read from memory. You should have the following inputs and outputs:

name	in/out	width	meaning
C	I	1 bit	clock input
instruction address	O	16 bits	the address of the instruction in the instruction memory

2. Build a **register file** made of sixteen 16-bit (Logisim) registers. The register file must be able to read from and write to specified registers. Register 0 is a special case: it always contains zero, and writing to it doesn't modify its contents. The register file has the following in- and outputs:

name	in/out	width	meaning
rs	I	4 bits	register `$rs` index number
rt	I	4 bits	register `$rt` index number
rd	I	4 bits	register `$rd` index number
D	I	16 bits	used as input for the write operation
write	I	1 bit	write to `$rd` enabled?
C	I	1 bit	clock input
S	O	16 bits	register `$rs` content
T	O	16 bits	register `$rt` content

3. Use your register file, your program counter, an *instruction* RAM element (16-bit addresses, 16-bit words), a *data* RAM element (16-bit addresses, 16-bit words) and your own ALU to implement a partial datapath. **Provide and discuss a number of test cases (and files) that demonstrate the operation of all instructions.**

   - The datapath must be able to perform so-called register operations. These are the 14 operations you implemented in your ALU. This time, operands are read from, and the result is stored into registers. The right registers are selected by specifying the rs, rt and rd index inputs. For binary operations (e.g. add, eq, ...), the registers are used as follows:

     `$rC = $rA` *`operation`* `$rB`

     For unary operations (e.g. not, sl, ...), the registers are used as follows ($rt is unused):

     `$rC =` *`operation`* `$rA`

     The 16-bit instructions for the register operations are formatted as follows:

     - `15-12` : ALU operation op code (`0000` to `1101`)
     - `11-8` : $rA
     - `7-4` : $rB
     - `3-0` : $rC

     Example: To add the values of register 1 and register 8, and put them in register 5, the following instruction is loaded:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1

   - The datapath must be able to perform the load word (lw – reading from *data* RAM, op-code `1110`) and store word (sw – writing to *data* RAM, op-code `1111`) operations. These are immediate instructions, and similar to the MIPS lw/sw instructions, a constant can be used to denote an offset. The meaning of these instructions is as follows:

     ```
     lw:  $rB = MEM[$rA + offset]
     sw:  MEM[$rA + offset] = $rB
     ```

The 16-bit instructions for the memory operations are formatted as follows:

- `15-12` : lw/sw operation op code (`1110` or `1111`)
- `11-8` : $rA
- `7-4` : $rB
- `3-0` : memory index offset for the lw/sw operations

Example: To store the value of register 10 in memory, 4 address spaces beyond the address stored in register 2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	1	0	1	0	0	1	0	0

4. Write simple programs with the instructions below. You can assemble a program written for your datapath with `rassemble.py` (for Python3). The assembling process creates two RAM images to load into the datapath RAM elements. The following instructions are supported:

   - `zero $rX`: $rX = 0
   - `and $rX, $rY, $rZ`: $rZ = $rX AND $rY
   - `or $rX, $rY, $rZ`: $rZ = $rX OR $rY
   - `not $rX, $rY`: $rY = NOT $rX
   - `inv $rX, $rY`: $rY = - $rX
   - `add $rX, $rY, $rZ`: $rZ = $rX + $rY
   - `sub $rX, $rY, $rZ`: $rZ = $rX - $rY
   - `sl $rX, $rY`: $rY = $rX << 1
   - `sr $rX, $rY`: $rY = $rX >> 1
   - `ssl $rX, $rY`: $rY = $rX * 2
   - `ssr $rX, $rY`: $rY = $rX / 2
   - `lt $rX, $rY, $rZ`: $rZ = $rX < $rY
   - `gt $rX, $rY, $rZ`: $rZ = $rX > $rY
   - `eq $rX, $rY, $rZ`: $rZ = $rX == $rY
   - `lw $rX, $rY, c`: $rY = MEM[$rX + c]
   - `sw $rX, $rY, c`: MEM[$rX + c] = $rY