CS&A: Lab Sessions

Project: Datapath (1)

Ruben Van den Bossche

1BA INF - 2010-2011

1 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 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: **November, 28 2010, 23u55**
- Evaluation and feedback: **November, 30 2010**

2 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. The special case of branching/jumping must also be implemented. In this case, the PC must go back or forward according to the branch offset. You should have te following inputs and outputs:

name	in/out	width	meaning
branch?	I	1 bits	selects whether we want to branch or increase the PC
branch offset	I	16 bits	the branch offset w.r.t. the PC
instruction	O	16 bits	the selected instruction from memory

2. Build a **register file** made of four 16-bit (Logisim) registers. The register file must be able to read from and write to specified registers. The register file has the following in- and outputs:

name	in/out	width	meaning
rs	I	2 bits	register `$rs` index number
rt	I	2 bits	register `$rt` index number
rd	I	2 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, a 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:

     `$rd = $rs` *`operation`* `$rt`

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

     `$rd =` *`operation`* `$rs`

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

     - `0-3` : ALU operation op code (`0000` to `1101`)
     - `4-5` : $rs
     - `6-7` : $rt
     - `8-9` : $rd
     - `10-15` : `000000`

   - The datapath must be able to perform the load word (lw – reading from RAM, op-code `1110`) and store word (sw – writing to 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:  $rd = MEM[$rs + offset]
     sw:  MEM[$rs + offset] = $rd
     ```

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

     - `0-3` : lw/sw operation op code (`1110` or `1111`)
     - `4-5` : $rs
     - `6-7` : $rd
     - `8-15` : memory index offset for the lw/sw operations

   Examples:

   - To add the values of register1 and register3, and put them in register0, the following instruction is loaded:

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

   - To store the value of register0 in memory, 4 address spaces beyond the address stored in register2:

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