CS&A: Lab Sessions

Project 2e session: Datapath

Ruben Van den Bossche

1BA INF - 2010-2011

1 Time Schedule

The project for the second session is solved individually. There will be an evaluation during the examination period. At this evaluation moment, you will present your solution of the project by giving a demo and answering some questions.

For this 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 assignments on Blackboard.

- Report deadline: **August, 21 2011, 23u55**

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 the "branch" mode
branch offset	I	16 bits	the branch offset w.r.t. the PC
jump?	I	1 bits	selects the "jump" mode
jump address	I	16 bits	the jump address
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 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. 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.*

4. Use your register file, your program counter, your control unit, 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.

   Implement the instructions described in the table below. 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!

   Provide and discuss a number of test cases (and files) that demonstrate the operation of all instructions.

0-3	4	5	6	7	8	9	10	11	12	13	14	15	
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[4]: $pc = addr
1000	rs		rt		offset (signed[2])								jr[4]: $pc = $rs+imm
1001	rs		rt		offset (signed[2])								beq[4]: ($rs=$rt) ? $pc=$pc+1+imm
1010	rs		rt		offset (signed[2])								bne[4]: ($rs≠$rt) ? $pc=$pc+1+imm

[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] You will have to adapt your branch method to support the behaviour of these instructions.

[5] The `lui` and `ori` instructions can be used together to implement a `li` pseudo-instruction which loads a 16-bit immediate into a register.

5. Exceptions are a very important part of a datapath and control. In this exercise, you will add a basic form of exception handling to your datapath: when an exception is detected, your program counter should halt at the instruction that caused the exception. Both arithmetic overflow and undefined instructions should be detected and supported.

   Think about enhanced versions of exception control. Why is it hard to add a more advanced form of exception handling to a datapath with our instruction set?

6. Demonstrate the proper operation of your datapath by providing a number of small RASM-programs. Implement the programs below. *Hint: if you run out of registers, you can use temporary variables in memory to store loop counters, intermediate results, ...*

   (a) A program that calculates the Fibonacci numbers and stores them in memory. After which number does overflow occur?

   (b) A program that calculates the greatest common divisor (using the Euclidean algorithm, but without recursion) of two integers read from memory. Store the result back in memory.

   (c) A program that finds the biggest element in an array of integers stored in memory. Store the biggest element back in memory.

   (d) A program that writes a 1 to memory if an array of integers in memory contains duplicates, and writes 0 if it doesn't.

   (e) A program that sorts an array of integers in memory.