Computer Systems and -architecture

Project 6: Full Datapath

1 Ba INF 2023-2024

Kasper Engelen
`kasper.engelen@uantwerpen.be`

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 one `tgz` or `zip` archive, as explained on the course's website, and submit your report to the exercises on Blackboard.

- Report deadline: **Monday December 18, 2023, 22u00**
- Evaluation and feedback: **Thursday/Friday December 21/22, 2023**

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 operations as instructions and added two additional instructions (`lw` and `sw`). Next to these instructions, in this assignment we also support **branch and jump instructions**.

   We introduce a number of new instructions for `jump` and `branch`. Because you should be able to branch, you will have to connect your **program counter** to your datapath so that it can jump to a given address instead of just the next instruction.

   Implement the instructions described in the table below ("imm" stands for "immediate", "uns" stands for "unsigned" and "sig" stands for "signed, two's complement"). You already have implemented the R-type instructions and the lw/sw instructions in the previous assignment.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	name	instruction	description
0000				0	rd			0000				0000				zero[1]	`zero rd`	$rd := 0
0001				0	rd			0	rs			0001				not[1]	`not rd rs`	$rd := !$rs
0001				0	rd			imm (sig.)				0010				jr	`jr rd imm`	$pc := $rd + imm
0001				target address								0011				j	`j imm`	$pc := addr
0001				target address								0100				jal[3]	`jal imm`	$r7:= $pc + 1; $pc := addr
0001				0	rd			0	rs			1010				inv[1]	`inv rd rs`	$rd := -$rs
0001				0	rd			0	rs			1011				sll[1]	`sll rd rs`	$rd := $rs << 2
0001				0	rd			0	rs			1100				srl[1]	`srl rd rs`	$rd := $rs >> 2
0001				0	rd			0	rs			1101				sla[1]	`sla rd rs`	$rd := $rs * 2
0001				0	rd			0	rs			1110				sra[1,2]	`sra rd rs`	$rd := $rs / 2
0001				0	rd			0	rs			1111				cp[1]	`cp rd rs`	$rd := $rs
0010				0	rd			0	rs			0	rt			and[1]	`and rd rs rt`	$rd := $rs & $rt
0011				0	rd			0	rs			0	rt			or[1]	`or rd rs rt`	$rd := $rs | $rt
0100				0	rd			0	rs			0	rt			add[1]	`add rd rs rt`	$rd := $rs + $rt
0101				0	rd			0	rs			0	rt			sub[1]	`sub rd rs rt`	$rd := $rs - $rt
0110				0	rd			0	rs			0	rt			lt[1]	`lt rd rs rt`	$rd := $rs < $rt ? 1 : 0
0111				0	rd			0	rs			0	rt			gt[1]	`gt rd rs rt`	$rd := $rs > $rt ? 1 : 0
1000				0	rd			0	rs			0	rt			eq[1]	`eq rd rs rt`	$rd := $rs = $rt ? 1 : 0
1001				0	rd			0	rs			0	rt			neq[1]	`neq rd rs rt`	$rd := $rs != $rt ? 1 : 0
1010				0	rd			0	rs			imm (uns.)				lw	`lw rd rs imm`	$rd := MEM[$rs+imm]
1011				0	rd			0	rs			imm (uns.)				sw	`sw rd rs imm`	MEM[$rs+imm] := $rd
1100				0	rd			imm (sig.)								ldi	`ldi rd imm`	$rd := imm
1101				0	rd			0	rs			imm (sig.)				addi	`addi rd rs imm`	$rd := $rs + imm
1110				0	rd			0	rs			imm (sig.)				beq	`beq rd rs imm`	$rd == $rs ? $pc := $pc + 1 + imm
1111				0	rd			0	rs			imm (sig.)				blt	`blt rd rs imm`	$rd < $rs ? $pc := $pc + 1 + imm

[1] R-type instruction.

[2] Integer division.

[3] Register r7 will be reserved for the return address of the `jal` instruction.

2. 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_2324_zit1_datapath.py -f -t <test-file> -c <circ-file>
   ```

   You can use labels for branching and jumping in your tests. 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.)