Computer Systems and -architecture

Extra Exercise: Simple Datapath

1 Ba INF 2022-2023

Brent van Bladel
brent.vanbladel@uantwerpen.be

Don't hesitate to contact the teaching assistant of this course. You can reach him in room M.G.305 or by e-mail.

In this exercise, we will create a small and simple datapath. Try to create everything from scratch without looking at previous projects.

1. Start by implementing a 8-bit program counter. You can use the logisim adder and register. This program counter simply increases by 1 every clock cycle. You don't need to consider jumps or branches.

name	in/out	width	meaning
C	I	1 bit	clock input
reset	I	1 bit	if set, the PC is reset to 0
instruction address	O	8 bit	the address of the instruction in the instruction memory

2. Implement an 8-bit ALU. You can use the logisim adder and subtractor.

name	in/out	width	meaning
OP code	I	2 bit	operation code
A	I	8 bit	first input of the operation
B	I	8 bit	second input of the operation
R	O	8 bit	result of the operation

Your ALU should support the following operations:

OP code	operation	meaning
00	NoOp	R = A
01	AND	R = A && B
10	ADD	R = A + B
11	SUB	R = A - B

3. Implement a register file that stores 8-bit data in 16 registers. You can use the logisim register.

name	in/out	width	meaning
rs	I	4 bit	register `rs` index number
rt	I	4 bit	register `rt` index number
rd	I	4 bit	register `rd` index number
Data	I	8 bit	used as input for the write operation, i.e., the new $rd value
C	I	1 bit	clock input
reset	I	1 bit	reset all registers?
S	O	8 bit	`$rs`; register `rs` content
T	O	8 bit	`$rt`; register `rt` content

Make sure that register R0 always contains 0. Note that data will be written to register rd every clock cycle! If an instruction does not require data to be written, simply use r0 as input for register rd.

4. Create a simple datapath by combining the previous circuits to implement a set of 11-bit instructions. The datapath should implement the following instruction set:

10	9	8	7	6	5	4	3	2	1	0	name	instruction	description
000			rs				rt				copy	`cp rs rt`	$rs := $rt
001			rs				rt				and	`and rs rt`	$r15 := $rs && $rt
010			rs				rt				add	`add rs rt`	$r15 := $rs + $rt
011			rs				rt				sub	`sub rs rt`	$r15 := $rs - $rt
100			immediate (signed)								load imm	`ldi imm`	$r15 := imm
101			immediate (signed)								and imm	`andi imm`	$r15 := $r15 && imm
110			immediate (signed)								add imm	`addi imm`	$r15 := $r15 + imm
111			immediate (signed)								sub imm	`subi imm`	$r15 := $r15 - imm

   Note that register 15 is being used to store the result of operations. Also note that we do not have DATA RAM, we only have our registers file as memory.

5. Write a program that calculates the first few fibonacci numbers. Note that you will have to convert your binary instructions to hex manually, in order to load them into Logisim.

Extra: If you want to exercise further, try expanding this instruction set by making the instructions 12-bit long. This allows you to add 8 more instructions to the set. You can choose which and how many instructions you want to add. Consider (a) other ALU operations (you will need to expand your ALU), (b) load and store instructions (you will need to introduce a RAM element), and (c) jump and branch instructions (you will need to expand your program counter).
A few examples of possible extra instructions:

11	10	9	8	7	6	5	4	3	2	1	0	name	instruction	description
1000				rs				rt				less than	`lt rs rt`	if $rs < $rt then $r15 := 1 else $r15 := 0
1001				offset (signed)								branch not zero	`brnz off`	if $r15 != 0 then $pc := $pc + 1 + offset
1010				target address								jump	`j address`	$pc := address
...														