

Gebze Technical University

Computer Engineering

CSE 331 - 2018 Fall

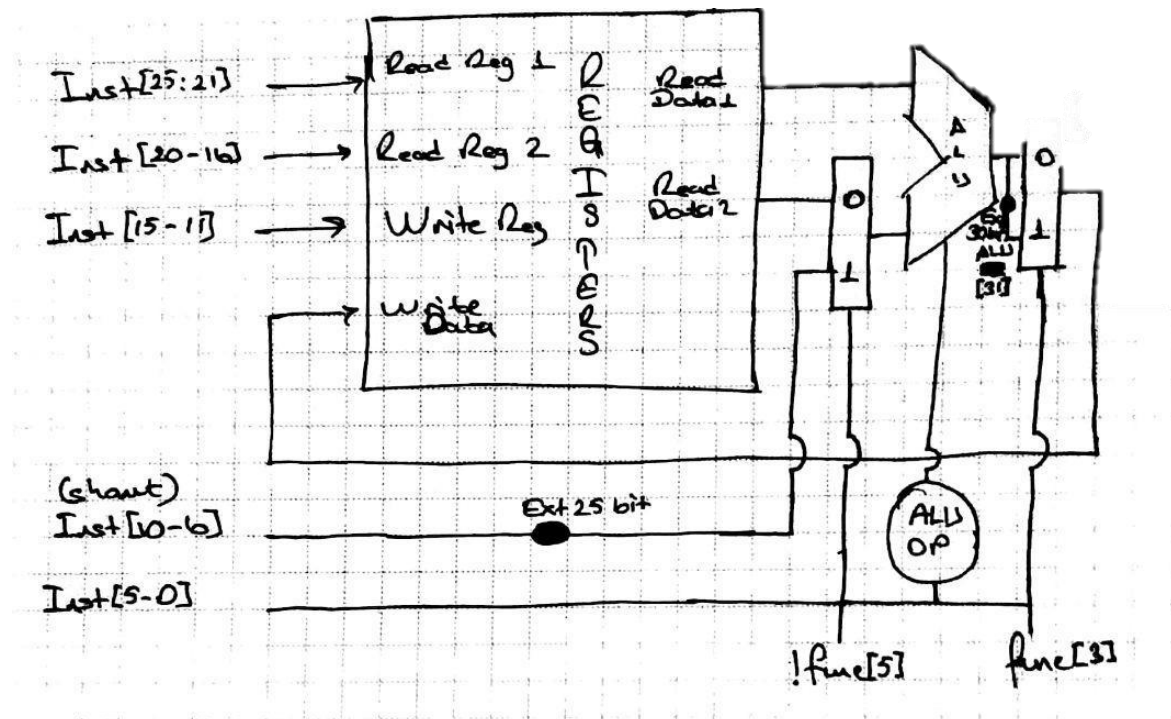
ASSIGNMENT 3 REPORT

HALİL ONUR ÇEÇEN

161044057

Making of Single Cycle Datapath of a mips processor

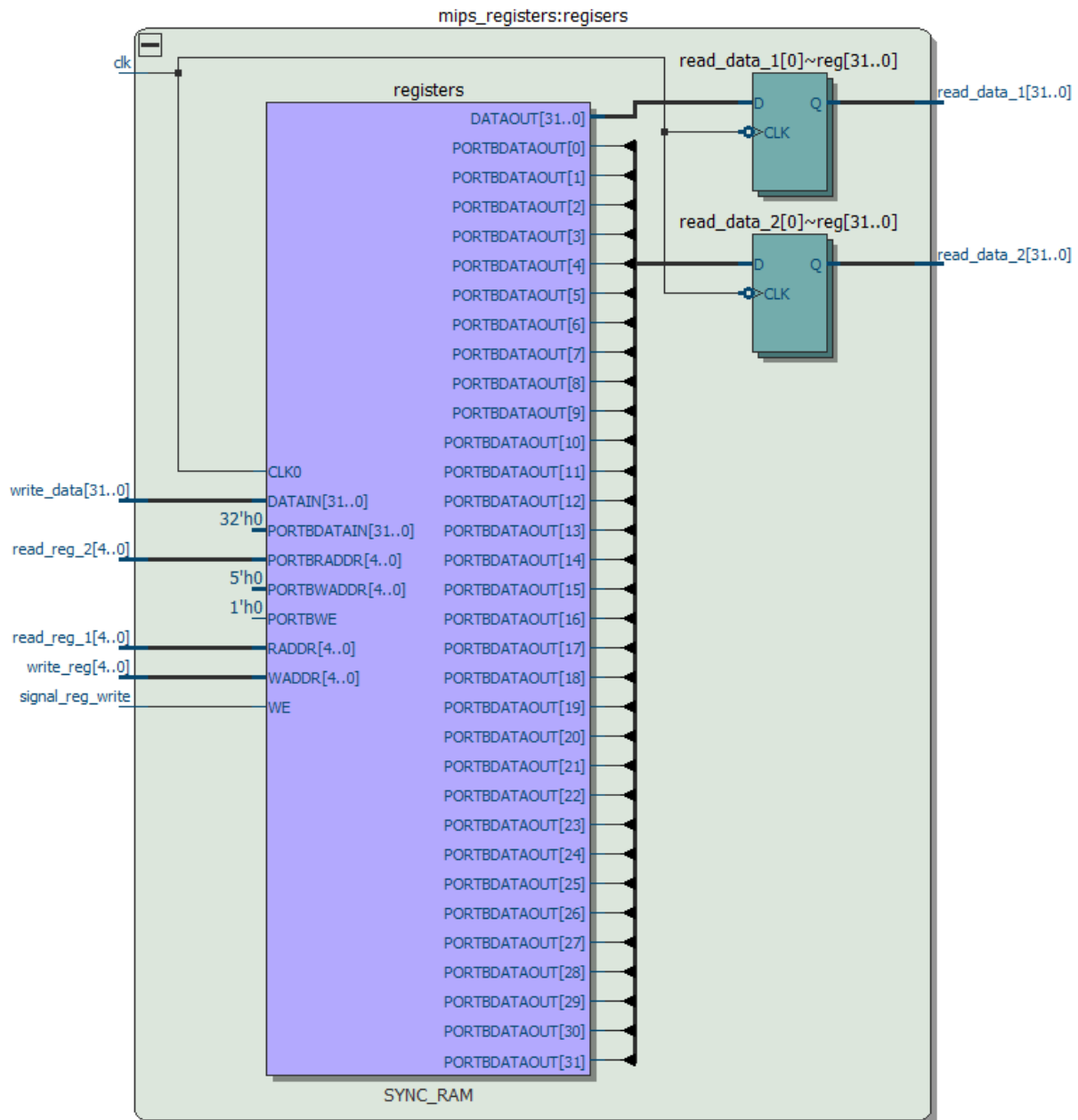
To make the required datapath I used following schema.



Main Mips32 Module(Explained)

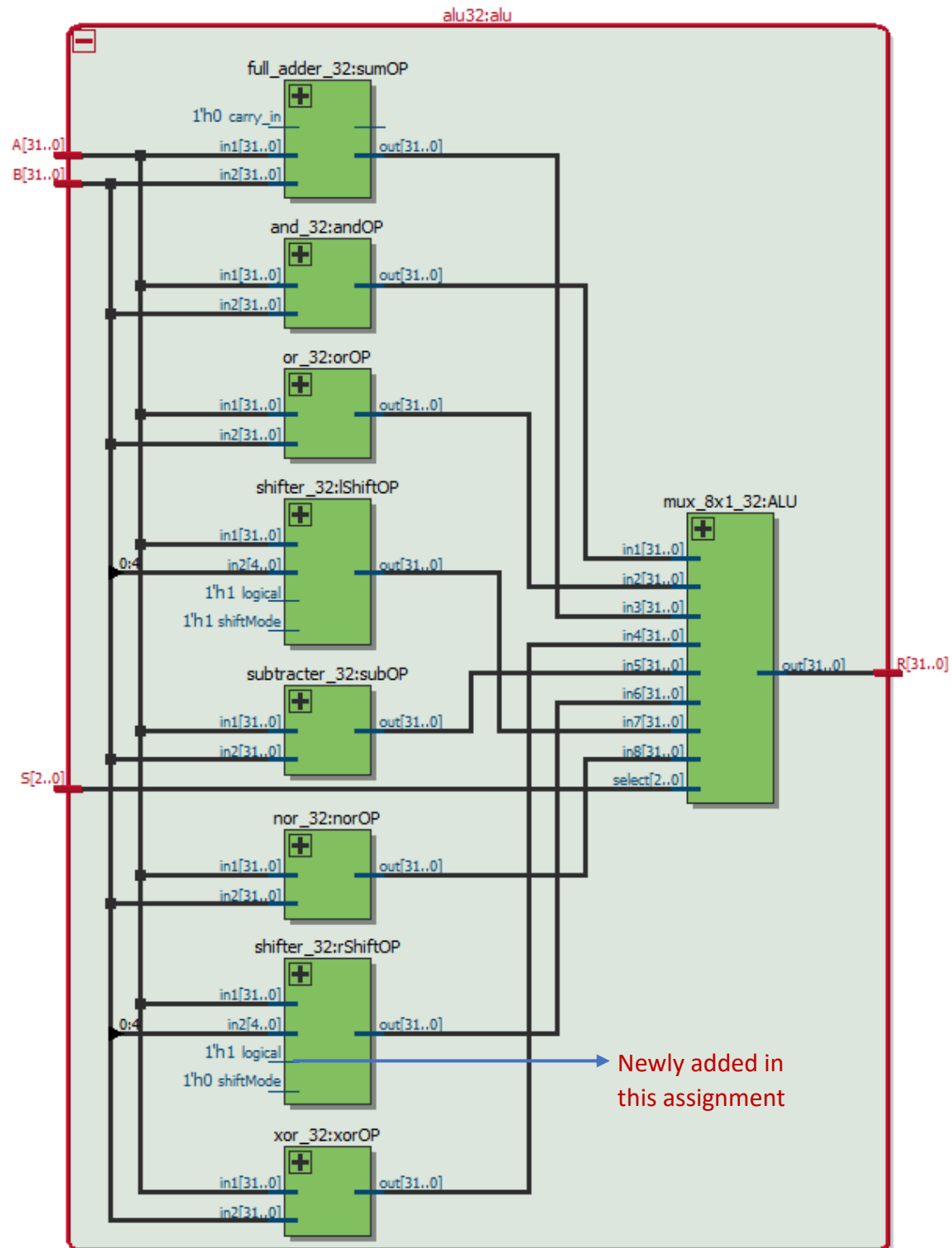
Registers (Mips_register module)

I used an external clock to determine to write or read in a single cycle. In positive edge it always writes and in negative edge it always reads.



Alu (From previous assignment)

I used previous alu unit with slight modifications to shifter to work both in arithmetic and logical way. To do that I added another control input to shifter module in alu.

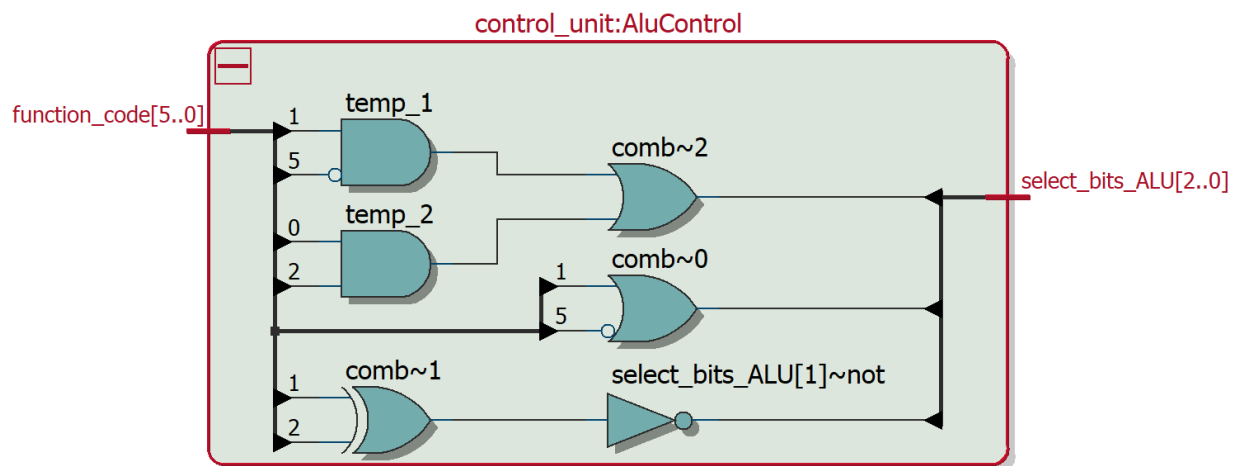


Alu Control Unit

I created a truth table for given function what to expect from Alu select bits. And after that used simplification and Karnaugh map methods to create simplest logic out of it.

	Inst[5-0]	ALU OP	ALU OP Module
	F5 F4 F3 F2 F1 F0	S2 S1 S0	
add	1 0 0 0 0 0	0 1 0	
addu	1 0 0 0 0 1	0 1 0	$S2 = \overline{F5} + F1$
and	1 0 0 1 0 0	0 0 0	
nor	1 0 0 1 1 1	1 1 1	$S1 = \overline{F2F1} + F2F1$ (XNOR)
or	1 0 0 1 0 1	0 0 1	
sltu	1 0 1 0 1 1	1 0 0	$S0 = \overline{F3F1} + F2F0$
sll	0 0 0 0 0 0	1 1 0	
srl	0 0 0 0 1 0	1 0 1	
sub	1 0 0 0 1 0	1 0 0	
subu	1 0 0 0 1 1	1 0 0	

Note: I used an online Karnaugh map solver for given input and outputs to take the most efficient logic.



All modules

work	Library
alu32	Module
and_32	Module
control_unit	Module
full_adder	Module
full_adder_32	Module
half_adder	Module
mips32	Module
mips32_testbenc...	Module
mips_registers	Module
mux_2x1	Module
mux_2x1_32	Module
mux_4x1	Module
mux_4x1_32	Module
mux_8x1_32	Module
nor_32	Module
not_32	Module
or_32	Module
shifter_32	Module
subtractor_32	Module
xor_32	Module

Simulation Results

```
# rs = 10000, rt = 10001, rd = 10010, func = 100000, shamt = 00000, result = 00000000000000000000000000000000100001
# rs = 10010, rt = 10001, rd = 10011, func = 000000, shamt = 00001, result = 000000000000000000000000000000001000010
# rs = 10010, rt = 10001, rd = 10100, func = 000010, shamt = 00001, result = 0000000000000000000000000000000010000
# rs = 10000, rt = 10001, rd = 10101, func = 100001, shamt = 00000, result = 00000000000000000000000000000000100001
# rs = 10000, rt = 10001, rd = 00000, func = 100100, shamt = 00000, result = 0000000000000000000000000000000010000
# rs = 10000, rt = 10001, rd = 00001, func = 100111, shamt = 00000, result = 1111111111111111111111111111101110
# rs = 10000, rt = 10001, rd = 00010, func = 100101, shamt = 00000, result = 0000000000000000000000000000000010001
# rs = 10000, rt = 10001, rd = 00011, func = 101011, shamt = 00000, result = 0000000000000000000000000000000000001
# rs = 10001, rt = 10000, rd = 10100, func = 100010, shamt = 00000, result = 0000000000000000000000000000000000001
# rs = 10001, rt = 10000, rd = 10101, func = 100011, shamt = 00000, result = 0000000000000000000000000000000000001
```

