Name: Joseph Hany **ID:900182870**

Architecture Lab4 Report

2) A technical summary of experiments conducted in the lab (steps, results, components, code functionality, etc.)

- **Experiment 1:**
  This experiment aims to implement A N-bit ALU with zero output flag that has a selection line in order to select one of these four operations: addition, subtraction, ANDing, and ORing
- In order to implement such a circuit, we should do the following:
  1- Create five module: ALU, Full_Adder, nbitMux, RCA4, TwoByOneMux
  2- Full_adder module is the basic unit the adds two simple signals that is used to build the RCA4
  3- RCA4 module instantiated Full_Adder module n times in order to construct the n bit carry adder
  4- TwoByOneMux module is the building block of the nbitMux module which selects which n-bit input to be used in the add/subtract operations

- **Experiment 2:**
  This experiment aims to implement a Register File with Reset
- In order to implement such a circuit, we should do the following:
  1- Create one new module: RegFile, and use 3 old modules: DFlipFlop, MUX_2_to_1, n_bit_register
  2- RegFile module simply generates n (n bit) registers which is the register file, reads the the values inside the two specified source registers r1, r2 and assign their values to rd1 and rd2, then it writes the value wd to the register with number “rw” if RegWrite pin was enabled.

  For example:

  - This block generates the n_bit_register in the register file:

```
genvar i;
generate
        for(i=0;i<32;i=i+1)
        begin:RegisterFile
                n_bit_register #(.n(n))r0(load[i],wd,clk,rst,Q[i]);
        end
endgenerate
```

  - This assigns the values that we read from rs1=r1 and rs2=r2

```
assign rd1=Q[r1];
assign rd2=Q[r2];
```

- This block writes to the register file whenever RegWrite is enabled

```
always@(RegWrite) begin
 for(j=0;j<32;j=j+1) begin
 if(j==rw) load[j]=RegWrite;
 else load[j]=0;
 end
end
```

- **<u>Experiment 3:</u>**
  This experiment aims to implement a circuit which acts as a control unit in our cpu
- In order to implement such a circuit, we should do the following:
  1- Create one module: A ControlUnit module
  2- This control module simply acts as a truth table that has four main cases 01100 (i.e. R-Format), 00000 (i.e. LW), 01000 (i.e. SW), 11000 (i.e. BEQ), upon seeing these opcodes it sets the control signal accordingly to the type of the instruction and its functionality.

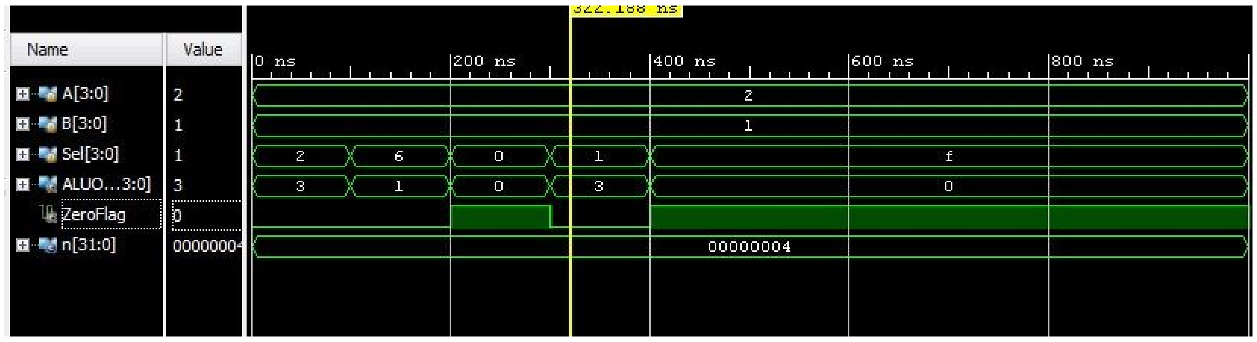
- **<u>Experiment 4:</u>**
  This experiment aims to implement a circuit which represents the ALU control unit
- In order to implement such a circuit, we should do the following:
  1- Create one module: an ALU_Control_Unit module.
  2- This module simply acts as a truth table that outputs the correct select line for the ALU circuit.

3) Verilog code for all modules of each experiment.
- **Verilog code of each experiment is attached in the zip file**

4) Verilog code for the testbench module of each experiment's top-level module.
- **Verilog code for the testbench is attached in the zip file**

5) Snapshot of simulation output:

- **EXP1 Simulation Snapshot:**

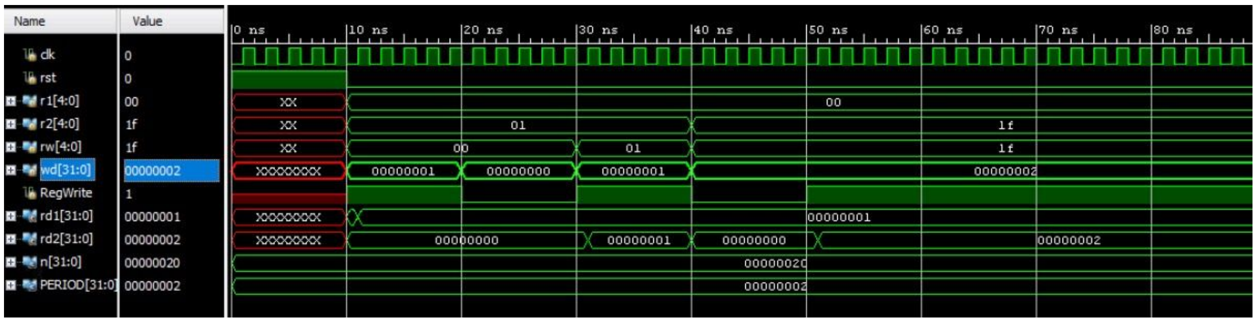


Here, we can see that when:

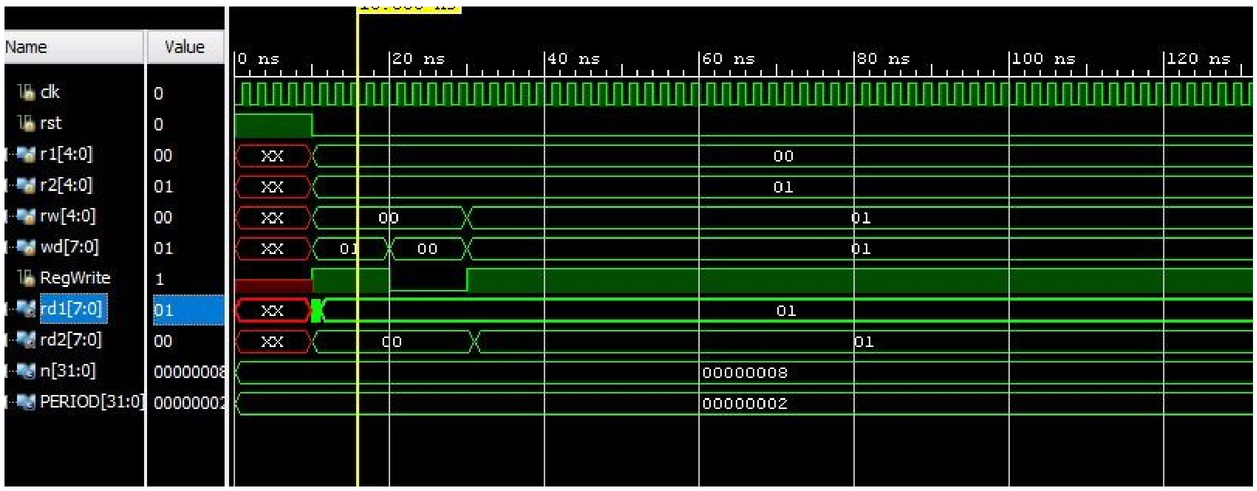
- The selection line to the ALU was Sel=4'b0010 (i.e. ADD operation) and the two inputs A=4'b0010 (i.e. 2 in decimal representation) and B=4'b0001(i.e. 1 in decimal representation), the output was 1 + 2 = 3.
- The selection line to the ALU was Sel=4'b0110 (i.e. SUB operation) and the two inputs A=4'b0010 (i.e. 2 in decimal representation) and B=4'b0001(i.e. 1 in decimal representation), the output was 2 - 1 = 1.
- The selection line to the ALU was Sel=4'b0000 (i.e. AND operation) and the two inputs A=4'b0010 (i.e. 2 in decimal representation) and B=4'b0001(i.e. 1 in decimal representation), the output was 2 & 1 = 0.
- The selection line to the ALU was Sel=4'b0001 (i.e. OR operation) and the two inputs A=4'b0010 (i.e. 2 in decimal representation) and B=4'b0001(i.e. 1 in decimal representation), the output was 2 or 1 = 3.
- The selection line to the ALU was Sel=4'b1111 (i.e. undefined operation) and the two inputs A=4'b0010 (i.e. 2 in decimal representation) and B=4'b0001(i.e. 1 in decimal representation), the output was 0 because the operation is not defined.

- **EXP2 Simulation Snapshot:**

When N = 32



When N = 8

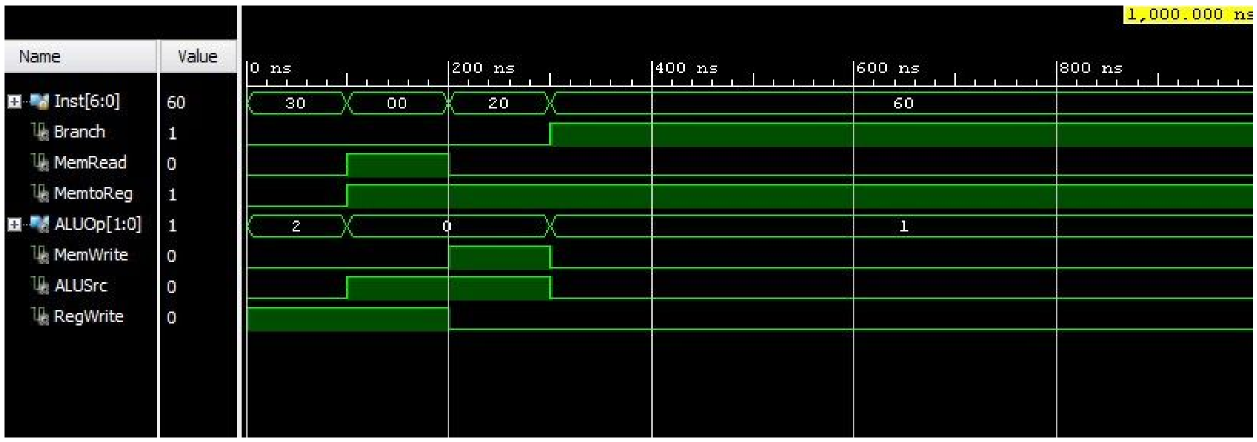


Here, we can see that when:

Since rst → reset pin which resets all the data in the register file
RegWrite → write data to the register file enabled
wd → the data which we will write the register
Rw → the number of the register we will write to
R1 → the number of first register we will read from
R2 → the number of second register we will read from

- Setting rst=0, RegWrite=1, wd=1, rw=0, r1=0, r2=1, we will get rd1 = 1 and rd2=0 since the writing is enabled and we wrote 1 to register 0 which we read again and put its value in rd1. However, rd2 is still zero because we did not write any value other than zero in it.
- Setting RegWrite=0, wd=0, we will have no change at all in the register file registers values. Thus, we will get the same outputs in rd and rd2.
- Setting wd=1, rw=1, RegWrite=1, we will get rd1 = 1 and rd2=0 since the writing is enabled and we wrote 1 to register 1 which we read back again. And rd1 is the same as before as we did not change the value inside it from the last clock cycle.

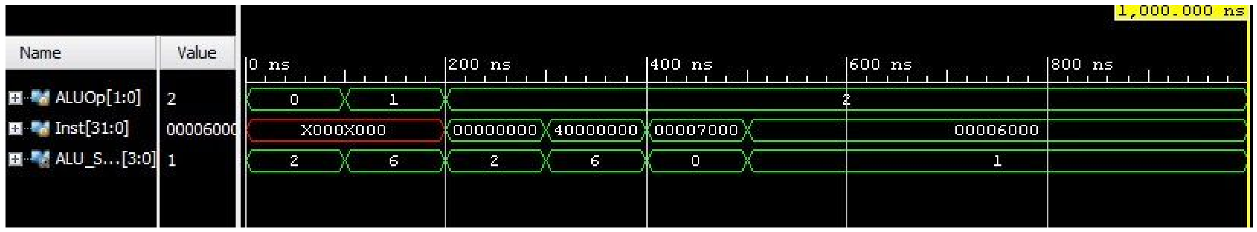
- **EXP3 Simulation Snapshot:**



Here, we can see that when:

- Setting Inst=7'b0110000, after parsing the Inst[6-2] we find it to be 01100 which is the R-Format. Hence, the signal values of the R-format found in the truth table in the lab manual has been set correctly.
- Setting Inst=7'b0000000, after parsing the Inst[6-2] we find it to be 00000 which is the Lw instruction. Hence, the signal values of the R-format found in the truth table in the lab manual has been set correctly.
- Setting Inst=7'b0100000, after parsing the Inst[6-2] we find it to be 01000 which is the Sw instruction. Hence, the signal values of the R-format found in the truth table in the lab manual has been set correctly.
- Setting Inst=7'b1100000, after parsing the Inst[6-2] we find it to be 11000 which is the Beq instruction. Hence, the signal values of the R-format found in the truth table in the lab manual has been set correctly.

- **EXP4 Simulation Snapshot:**



Here, we can see that when:

- Inst=0, ALUOp=2'b00,Inst[14:12]=3'bxxx (DON't CARES), Inst[30]=1'bx → the output is 0010 (ADD)
- ALUOp=2'b01, Inst[14:12]=3'bxxx (DON't CARES), Inst[30]=1'bx → the output is 0110 (SUB)
- ALUOp=2'b10, Inst[14:12]=3'b000, Inst[30]=1'b0 → the output is 0010 (ADD)
- ALUOp=2'b10, Inst[14:12]=3'b000, Inst[30]=1'b1 → the output is 0110 (SUB)
- ALUOp=2'b10, Inst[14:12]=3'b111, Inst[30]=1'b0 → the output is 0000 (AND)
- ALUOp=2'b10, Inst[14:12]=3'b110, Inst[30]=1'b0 → the output is 0001 (OR)