COMPUTER ORGANIZATION
EECS 2021

LAB "N" REPORT
AUTOMATING THE CONTROL

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

X _____

Camillo John (CJ) D'Alimonte
212754396
Lab Section 2
12/05/14
Professor Aboelaze

Abstract:

This laboratory/experiment was a direct continuation of the previous Lab (Lab M). Once again, I was introduced to some of the more complex applications that can be built through the Verilog language, namely an automated model of a CPU. My CPU will become capable of self-setting the signals it needs. I learned how to distinguish the differences between branching and sequential processing and how to build modules that were to choose between these different operations. One key idea was to automate the generation of our control signals, namely RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite, jump, branch. In a more general sense, I was able to better hone my skills of designing Verilog code from a given circuit diagram. By the end of the lab, I had learned some of the more complex applications (i.e. modelling a CPU) that can be programmed in the Verilog language.

Equipment Used:

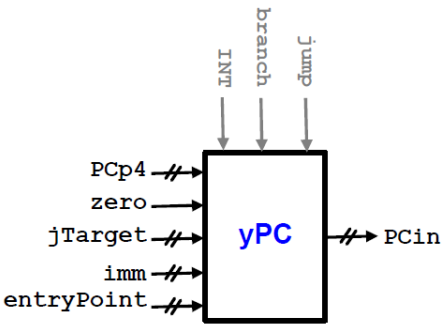
This lab was done through the UNIX environment at the Lassonde Lab. I used the pico editor in UNIX to write my code and I used the built-in Verilog compiler to run and test my programs.

Methods/Procedures:

For LabN1, I needed to recall that my program determined the address of instructions to be executed next based on whether we were jumping, branching, or continuing sequentially. We wanted to alter our program so that this functionality was implemented structurally in the circuit with an output from PCin. But since PCin was an output, I needed to find a mechanism to force the CPU to stop the current program and switch to another. I developed the INT signal (a 1 bit interrupt signal) as well as an entryPoint signal (32 bits containing the address of the switch). Our PCin login was altered to:

if (INT == 1)
PCin = entryPoint;
else
if (*beq* && zero == 1)
PCin = PCp4 + *imm shifted left twice*;
else if (*j*)
PCin = *jTarget shifted left twice*;
else
PCin = PCp4;

I had to fetch the next instruction from address entryPoint thereby affecting a context switch. With this new scheme, PCin is no longer set externally and, hence, can indeed be an output of a circuit. The diagram for the new yPC is below:



To implement yPC, I needed to create three different multiplexers to choose between alternates. The first mux chooses between sequential processing and branching. We branch if there is a beq instruction and its registers are equal. bTarget is computed by multiplying imm by 4 and adding the resut to PCp4. The second mux chooses between the previous mux output and jumping. The jump target is a 32 bit address whose high order 4 bits come from PCp4 and whose lower 26 bits are the input jTarget. The third mux chooses between the previous mux ouput and entryPoint. INT is its control signal.

Pseudo Code
LabN1.v
	entryPoint = 128; INT = 1; #1;
	PCin = 128;
	clk = 1; #1; INT = 0; branch = 0; jump = 0;
	RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;
	if(ins[31:26] == 0)
	RegDst = 1; RegWrite = 1; ALUSrc = 0;
	if(ins[5:0] == 'h20)
	RegDst = 1; RegWrite = 1; ALUSrc = 0;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	else if(ins[5:0] == 'h25)
	op = 3'b001;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	else if(ins[31:26] == 2 || ins[31:26] == 3)
	RegDst = 1; RegWrite = 1; ALUSrc = 1;
	if(ins[31:26] → 'h4)
	ALUSrc = 0; op = 3'b110;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	branch = 1;
	else if(ins[31:26] → 'h23)
	RegDst = 0; RegWrite = 1; ALUSrc = 1;
	MemRead = 1; MemWrite = 0; Mem2Reg = 1;
	else if(ins[31:26] → 'h8)
	RegDst = 0; RegWrite = 1; ALUSrc = 1;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	clk = 0; #1;
	$display(ins, rd1, rd2, z, zero, wb);

For yC1, I needed to set our 8 control signals RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite, jump, branch. We wanted to automate the generation of these eight signals by building structural circuits that output them. In order to accomplish this, I needed to create a circuit that would determine if the instruction is load, store, branch-on-equal, jump, or R-type, and outputs lw, sw, branch, jump, or rtype accordingly:

module yC1(rtype, lw, sw, jump, branch, opCode);
output rtype, lw, sw, jump, branch;
input [5:0] opCode;

The signals sw, branch, and jump can be generated similarly to lw by noting the opCode of the corresponding instructions. The rtype signal is generated by detecting the case of all 6 bits of opCode being 0.

Pseudo Code LabyC1.v	not (not5, opCode[5]); not (not4, opCode[4]); not (not3, opCode[3]);

	not (not2, opCode[2]); not (not1, opCode[1]); not (not0, opCode[0]); and (rtype, not1, not2, not3, not4, not5, not0); and (lw, opCode[5], not4, not3, not2, opCode[1], opCode[0]); and (sw, opCode[5], not4, opCode[3], not2, opCode[1], opCode[0]); and (jump, not5, not4, not3, not2, opCode[1], not0); and (branch, not5, not4, not3, opCode[2], not1, not0);

For yC2, I needed to implement the module below:

module yC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite,
rtype, lw, sw, branch);
output RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite;
input rtype, lw, sw, branch;

This module represents the second part of the control unit. It takes four of the signals generated by yC1 as input, and generates the six control signals we need. To build this component, I need to implement the logic of the "*Set control signals*" section in the hardware. To that end, I needed to switch from sequential, if-then-else thinking, to parallel, declarative thinking. For ALUSrc, the signal must be 0 for R-types and branches, 1 for loads and stores and addi, and don't-care for jumps. I need to manipulate the don't-care and treat it as 1 so as to end up with a simple rule: 0 for R-types and branches and 1 otherwise.

Pseudo Code
LabyC2.v
	assign RegDst = rtype;
	nor (ALUSrc, rtype, branch);
	nor (RegWrite, sw, branch);
	assign Mem2Reg = lw;
	assign MemWrite = sw;

For LabN2, I needed to write a program that instantiates the two parts of the control unit in addition to the datapath components:

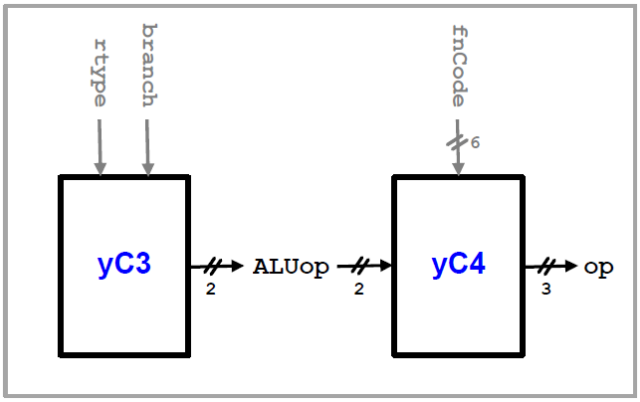
yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEx(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);
assign wd = wb;
yPC myPC(PCin, PCp4,INT,entryPoint,imm,jTarget,zero,branch,jump);
assign opCode = ins[31:26];
yC1 myC1(rtype, lw, sw, jump, branch, opCode);
yC2 myC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite,
rtype, lw, sw, branch);

Except for op, our CPU has become capable of self-setting the signals it needs.

Pseudo Code
LabN2.v
	assign opCode = ins[31:26];
	yC1 myC1(rtype, lw, sw, jump, branch, opCode);
	yC2 myC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead,
	MemWrite, rtype, lw, sw, branch);

```
entryPoint = 128; INT = 1; #1;
PCin = 128;
clk = 1; #1; INT = 0; branch = 0; jump = 0;
RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;
if(ins[31:26] == 0) //R-type
RegDst = 1; RegWrite = 1; ALUSrc = 0;
if(ins[5:0] == 'h20) // add instruction
RegDst = 1; RegWrite = 1; ALUSrc = 0;
MemRead = 0; MemWrite = 0; Mem2Reg = 0;
else if(ins[5:0] == 'h25)
op = 3'b001;
MemRead = 0; MemWrite = 0; Mem2Reg = 0;
else if(ins[31:26] == 2 || ins[31:26] == 3)
RegDst = 1; RegWrite = 1; ALUSrc = 1;
if(ins[31:26] == 'h4) // beq instruction
ALUSrc = 0; op = 3'b110;
MemRead = 0; MemWrite = 0; Mem2Reg = 0;
branch = 1;
else if(ins[31:26] == 'h23) // lw instruction
RegDst = 0; RegWrite = 1; ALUSrc = 1;
MemRead = 1; MemWrite = 0; Mem2Reg = 1;
else if(ins[31:26] == 'h8) // addi instruction
RegDst = 0; RegWrite = 1; ALUSrc = 1;
MemRead = 0; MemWrite = 0; Mem2Reg = 0;
clk = 0; #1;
$display( ins, rd1, rd2, z, zero, wb);
```

For LabyC3, we shifted focus to the op signal. It's the hardest control signal to generate since it depends sensitively on the instruction. We overcome the above difficulty by dividing the problem into two and building two back-to-back circuits: The first, yC3, is responsible for non-R-type instructions and the second, yC4, takes care of R-types. These two circuits interact with each other through a new 2-bit signal ALUop as shown in this block diagram:



The yC3 circuit must generate ALUop as shown in in the table below:

Type	Instruction	Operation	ALUop
I	lw	addition	00
	sw	addition	00
	addi	addition	00
	beq	subtraction	01
J	j	don't-care	xx
R	unknown	unknown	10

Pseudo Code LabyC3.v	output [1:0] ALUop;
	input rtype, branch;
	assign ALUop[1] = rtype;
	assign ALUop[0] = branch;

For LabyC4, we now turned our attention to the fourth and last part of our control unit, yC4. This unit sees the fnCode and the ALUop signal generated by yC3 and outputs the 3-bit ALU signal op. Because of this, it is sometimes referred to as the *ALU Control Unit*. Here is the specification of this unit:

ALUop	funCode	Instruction	Operation	op
00	*don't-care*	*don't-care*	addition	010
01	*don't-care*	*don't-care*	subtraction	110
10	100100	and	conjunction	000
10	100101	or	disjunction	001
10	100000	add	addition	010
10	100010	sub	subtraction	110
10	101010	slt	set-on-less-than	111

This unit operates primarily based on ALUop. If this signal is 00 or 01 then yC4 *trusts* and findings of yC3 and generates op accordingly. But if ALUop is 10 then yC4 *knows* that this is an R-type instruction and hence generates op based on the function code.

Pseudo Code LabyC4.v	or left(leftOr, fnCode[0], fnCode[3]);
	or rightUpperOr(op[2], ALUop[0], upperAnd);
	or rightLowerOr(op[1], invertOp1, invertF2);
	and upperAnd(upperAnd, ALUop[1], fnCode[1]);
	and lowerAnd(op[0], ALUop[1], leftOr);
	not invertOp1(invertOp1, ALUop[1]);
	not invertF2(invertF2, fnCode[2]);

For LabN3, I needed to modify my LabN2.v file in the following manner:

yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEx(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);
assign wd = wb;
yPC myPC(PCin, PCp4,INT,entryPoint,imm,jTarget,zero,branch,jump);
assign opCode = ins[31:26];
yC1 myC1(rtype, lw, sw, jump, branch, opCode);
yC2 myC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite,
rtype, lw, sw, branch);
assign fnCode = ins[5:0];
yC3 myC3(ALUop, rtype, branch);
yC4 myC4(op, ALUop, fnCode);

The CPU is now capable of executing the program without any assistance from external modules.

Pseudo Code
LabN3.v
	entryPoint = 128; INT = 1; #1;
	PCin = 128;
	clk = 1; #1; INT = 0; branch = 0; jump = 0;
	RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;
	if(ins[31:26] == 0) //R-type
	RegDst = 1; RegWrite = 1; ALUSrc = 0;
	if(ins[5:0] == 'h20) // add instruction
	RegDst = 1; RegWrite = 1; ALUSrc = 0;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	else if(ins[5:0] == 'h25)
	op = 3'b001;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	else if(ins[31:26] == 2 || ins[31:26] == 3)
	jump = 1;
	RegDst = 1; RegWrite = 1; ALUSrc = 1;
	if(ins[31:26] == 'h4) // beq instruction
	ALUSrc = 0; op = 3'b110;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	branch = 1;
	else if(ins[31:26] == 'h23) // lw instruction
	RegDst = 0; RegWrite = 1; ALUSrc = 1;
	MemRead = 1; MemWrite = 0; Mem2Reg = 1;
	else if(ins[31:26] == 'h8) // addi instruction
	RegDst = 0; RegWrite = 1; ALUSrc = 1;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	clk = 0; #1;
	$display(ins, rd1, rd2, z, zero, wb);

For LabN4, we repackage our components so as to fully separate the concerns. We put all the needed instantiation in one module that represents the CPU chip:

```
module yChip(ins, rd2, wb, entryPoint, INT, clk);
output [31:0] ins, rd2, wb;
input [31:0] entryPoint;
input INT, clk;
```

This module needs not have any output but we have declared ins, wb, and rd2 simply to be able to test it (rd2 helps us test sw). We save LabN3 as LabN4.v and modify it by replacing all the instantiated circuits with an instantiation of yChip:

Pseudo Code
LabN4.v
	reg [31:0] entryPoint;
	reg clk, INT;
	wire [31:0] ins, rd2, wb;
	yChip myChip(ins, rd2, wb, entryPoint, INT, clk);
	assign entryPoint = 32'h80;
	assign INT = 1; #1;
	#1 assign INT = 0;
	repeat(43)
	begin
	clk = 1; #1;
	clk = 0; #1;
	$display(ins, rd2, wb);

Results:

The following is the recorded results for each experiment.

LabN1.v

```
jun13 302 % iverilog LabN1.v cpu.v
jun13 303 % vvp a.out
00006820: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00008020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00002020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
8da80050: rd1= 0 rd2= x z= 80 zero=0 wb= 1
11000004: rd1= 1 rd2= 0 z=  1 zero=0 wb= 1
02088020: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
00882025: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
21ad0004: rd1= 0 rd2= 0 z=  4 zero=0 wb= 4
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 4 rd2= 1 z= 84 zero=0 wb= 3
11000004: rd1= 3 rd2= 0 z=  3 zero=0 wb= 3
02088020: rd1= 1 rd2= 3 z=  4 zero=0 wb= 4
00882025: rd1= 1 rd2= 3 z=  3 zero=0 wb= 3
21ad0004: rd1= 4 rd2= 4 z=  8 zero=0 wb= 8
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 8 rd2= 3 z= 88 zero=0 wb= 5
11000004: rd1= 5 rd2= 0 z=  5 zero=0 wb= 5
02088020: rd1= 4 rd2= 5 z=  9 zero=0 wb= 9
00882025: rd1= 3 rd2= 5 z=  7 zero=0 wb= 7
21ad0004: rd1= 8 rd2= 8 z= 12 zero=0 wb=12
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=12 rd2= 5 z= 92 zero=0 wb= 7
11000004: rd1= 7 rd2= 0 z=  7 zero=0 wb= 7
02088020: rd1= 9 rd2= 7 z= 16 zero=0 wb=16
00882025: rd1= 7 rd2= 7 z=  7 zero=0 wb= 7
21ad0004: rd1=12 rd2=12 z= 16 zero=0 wb=16
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=16 rd2= 7 z= 96 zero=0 wb= 9
11000004: rd1= 9 rd2= 0 z=  9 zero=0 wb= 9
02088020: rd1=16 rd2= 9 z= 25 zero=0 wb=25
00882025: rd1= 7 rd2= 9 z= 15 zero=0 wb=15
21ad0004: rd1=16 rd2=16 z= 20 zero=0 wb=20
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=20 rd2= 9 z=100 zero=0 wb=11
11000004: rd1=11 rd2= 0 z= 11 zero=0 wb=11
02088020: rd1=25 rd2=11 z= 36 zero=0 wb=36
00882025: rd1=15 rd2=11 z= 15 zero=0 wb=15
21ad0004: rd1=20 rd2=20 z= 24 zero=0 wb=24
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=24 rd2=11 z=104 zero=0 wb= 0
11000004: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
ac100020: rd1= 0 rd2=36 z= 32 zero=0 wb=32
ac040024: rd1= 0 rd2=15 z= 36 zero=0 wb=36
```

LabN2.v

```
jun13 306 % iverilog LabN2.v cpu.v
jun13 307 % vvp a.out
00006820: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00008020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00002020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
8da80050: rd1= 0 rd2= x z= 80 zero=0 wb= 1
11000004: rd1= 1 rd2= 0 z=  1 zero=0 wb= 1
02088020: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
00882025: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
21ad0004: rd1= 0 rd2= 0 z=  4 zero=0 wb= 4
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 4 rd2= 1 z= 84 zero=0 wb= 3
11000004: rd1= 3 rd2= 0 z=  3 zero=0 wb= 3
02088020: rd1= 1 rd2= 3 z=  4 zero=0 wb= 4
00882025: rd1= 1 rd2= 3 z=  3 zero=0 wb= 3
21ad0004: rd1= 4 rd2= 4 z=  8 zero=0 wb= 8
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 8 rd2= 3 z= 88 zero=0 wb= 5
11000004: rd1= 5 rd2= 0 z=  5 zero=0 wb= 5
02088020: rd1= 4 rd2= 5 z=  9 zero=0 wb= 9
00882025: rd1= 3 rd2= 5 z=  7 zero=0 wb= 7
21ad0004: rd1= 8 rd2= 8 z= 12 zero=0 wb=12
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=12 rd2= 5 z= 92 zero=0 wb= 7
11000004: rd1= 7 rd2= 0 z=  7 zero=0 wb= 7
02088020: rd1= 9 rd2= 7 z= 16 zero=0 wb=16
00882025: rd1= 7 rd2= 7 z=  7 zero=0 wb= 7
21ad0004: rd1=12 rd2=12 z= 16 zero=0 wb=16
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=16 rd2= 7 z= 96 zero=0 wb= 9
11000004: rd1= 9 rd2= 0 z=  9 zero=0 wb= 9
02088020: rd1=16 rd2= 9 z= 25 zero=0 wb=25
00882025: rd1= 7 rd2= 9 z= 15 zero=0 wb=15
21ad0004: rd1=16 rd2=16 z= 20 zero=0 wb=20
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=20 rd2= 9 z=100 zero=0 wb=11
11000004: rd1=11 rd2= 0 z= 11 zero=0 wb=11
02088020: rd1=25 rd2=11 z= 36 zero=0 wb=36
00882025: rd1=15 rd2=11 z= 15 zero=0 wb=15
21ad0004: rd1=20 rd2=20 z= 24 zero=0 wb=24
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=24 rd2=11 z=104 zero=0 wb= 0
11000004: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
ac100020: rd1= 0 rd2=36 z= 32 zero=0 wb=32
ac040024: rd1= 0 rd2=15 z= 36 zero=0 wb=36
```

LabN3.v

```
jun13 308 % iverilog LabN3.v cpu.v
jun13 309 % vvp a.out
00006820: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00008020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
```

```
00002020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
8da80050: rd1= 0 rd2= x z= 80 zero=0 wb= 1
11000004: rd1= 1 rd2= 0 z=  1 zero=0 wb= 1
02088020: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
00882025: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
21ad0004: rd1= 0 rd2= 0 z=  4 zero=0 wb= 4
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 4 rd2= 1 z= 84 zero=0 wb= 3
11000004: rd1= 3 rd2= 0 z=  3 zero=0 wb= 3
02088020: rd1= 1 rd2= 3 z=  4 zero=0 wb= 4
00882025: rd1= 1 rd2= 3 z=  3 zero=0 wb= 3
21ad0004: rd1= 4 rd2= 4 z=  8 zero=0 wb= 8
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1= 8 rd2= 3 z= 88 zero=0 wb= 5
11000004: rd1= 5 rd2= 0 z=  5 zero=0 wb= 5
02088020: rd1= 4 rd2= 5 z=  9 zero=0 wb= 9
00882025: rd1= 3 rd2= 5 z=  7 zero=0 wb= 7
21ad0004: rd1= 8 rd2= 8 z= 12 zero=0 wb=12
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=12 rd2= 5 z= 92 zero=0 wb= 7
11000004: rd1= 7 rd2= 0 z=  7 zero=0 wb= 7
02088020: rd1= 9 rd2= 7 z= 16 zero=0 wb=16
00882025: rd1= 7 rd2= 7 z=  7 zero=0 wb= 7
21ad0004: rd1=12 rd2=12 z= 16 zero=0 wb=16
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=16 rd2= 7 z= 96 zero=0 wb= 9
11000004: rd1= 9 rd2= 0 z=  9 zero=0 wb= 9
02088020: rd1=16 rd2= 9 z= 25 zero=0 wb=25
00882025: rd1= 7 rd2= 9 z= 15 zero=0 wb=15
21ad0004: rd1=16 rd2=16 z= 20 zero=0 wb=20
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=20 rd2= 9 z=100 zero=0 wb=11
11000004: rd1=11 rd2= 0 z= 11 zero=0 wb=11
02088020: rd1=25 rd2=11 z= 36 zero=0 wb=36
00882025: rd1=15 rd2=11 z= 15 zero=0 wb=15
21ad0004: rd1=20 rd2=20 z= 24 zero=0 wb=24
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
8da80050: rd1=24 rd2=11 z=104 zero=0 wb= 0
11000004: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
ac100020: rd1= 0 rd2=36 z= 32 zero=0 wb=32
ac040024: rd1= 0 rd2=15 z= 36 zero=0 wb=36
```

LabN4.v

```
jun13 312 % iverilog LabN4.v cpu.v
jun13 313 % vvp a.out
00006820: rd2= 0 wb= 0
00008020: rd2= 0 wb= 0
00002020: rd2= 0 wb= 0
8da80050: rd2= x wb= 1
11000004: rd2= 0 wb= 1
02088020: rd2= 1 wb= 1
00882025: rd2= 1 wb= 1
```

```
21ad0004: rd2= 0 wb= 4
08000023: rd2= 0 wb=35
8da80050: rd2= 1 wb= 3
11000004: rd2= 0 wb= 3
02088020: rd2= 3 wb= 4
00882025: rd2= 3 wb= 3
21ad0004: rd2= 4 wb= 8
08000023: rd2= 0 wb=35
8da80050: rd2= 3 wb= 5
11000004: rd2= 0 wb= 5
02088020: rd2= 5 wb= 9
00882025: rd2= 5 wb= 7
21ad0004: rd2= 8 wb=12
08000023: rd2= 0 wb=35
8da80050: rd2= 5 wb= 7
11000004: rd2= 0 wb= 7
02088020: rd2= 7 wb=16
00882025: rd2= 7 wb= 7
21ad0004: rd2=12 wb=16
08000023: rd2= 0 wb=35
8da80050: rd2= 7 wb= 9
11000004: rd2= 0 wb= 9
02088020: rd2= 9 wb=25
00882025: rd2= 9 wb=15
21ad0004: rd2=16 wb=20
08000023: rd2= 0 wb=35
8da80050: rd2= 9 wb=11
11000004: rd2= 0 wb=11
02088020: rd2=11 wb=36
00882025: rd2=11 wb=15
21ad0004: rd2=20 wb=24
08000023: rd2= 0 wb=35
8da80050: rd2=11 wb= 0
11000004: rd2= 0 wb= 0
ac100020: rd2=36 wb=32
ac040024: rd2=15 wb=36
```

Discussion:

The following are the answers to the questions listed in the lab manual.

Question 12) I ran into difficulty trying to correctly place the bits of the jump instruction into jumpA, but other than that, the program ran as expected.

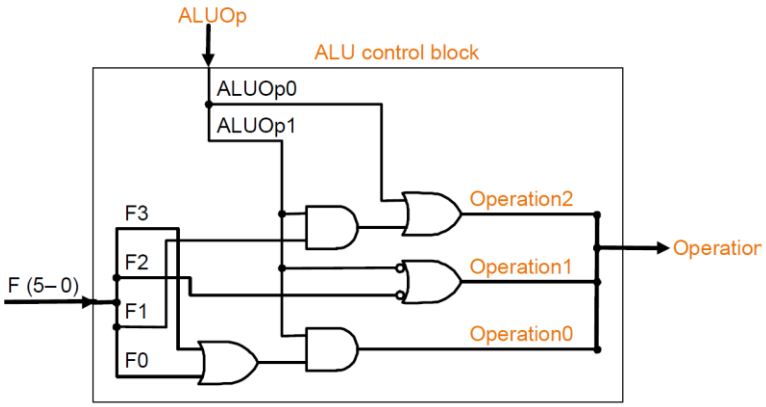
Question 27) After minor debugging, and fixing syntax errors in the circuit, the program ran as expected.

Question 35) The table is correct for the following reasons:

- If ALUop = 00
  - op[2] = 0 by the top AND gate (0 & X = 0) and the top OR gate (0 | 0 = 0)
  - op[1] = 1 by the middle OR gate (~0 | X = 1 | X = 1)

- op[0] = 0 by the bottom AND gate (0 | X = 0)

- If ALUop = 01
  - op[2] = 1 by the top OR gate (1 | X = 1)
  - op[1] = 1 by the top OR gate (~0 | X = 1 | X = 1)
  - op[0] = 0 by the bottom AND gate (0 & X = 0)
- If ALUop = 10
  - and: 10**0100**
    - op[2] = (1 & 0) | 0 = 0 | 0 = 0
    - op[1] = (~1 | ~1) = (0 | 0) = 0
    - op[0] = (0 | 0) & 1 = 0 & 1 = 0
  - or: 10**0101**
    - op[2= (1 & 0) | 0 = 0 | 0 = 0
    - op[1] = (~1 | ~1) = (0 | 0) = 0
    - op[0] = (1 | 0) & 1= 1 & 1 = 1
  - add: 10**0000**
    - op[2] = = (1 & 0) | 0 = 0 | 0 = 0
    - op[1] = (~1 | ~0) = (0 | 1) = 1
    - op[0] = (0 | 0) & 1= 0 & 1 = 0
  - sub: 10**0010**
    - op[2] = (1 & 1) | 1 = 1 | 1= 1
    - op[1] = (~1 | ~0) = (0 | 1) = 1
    - op[0] = (0 | 0) & 1= 0 & 1 = 0
  - slt: 10**1010**
    - op[2] = (1 & 1) | 1 = 1 | 1= 1
    - op[1] = (~1 | ~0) = (0 | 1) = 1
    - op[0] = (0 | 1) & 1 = 1 & 1 = 1



Question 39) After fixing a small syntax error, the circuit ran as expected.

Question 43) The circuit ran as expected, with no issues.

Conclusion:

This lab taught me how to build a variety of components that make up the automation of a CPU. My CPU built in this Lab communicates with the outside world through four channels:

- The clock signal (input)
- The interrupt signal (input)
- The entry point signal (input)
- The BIU (Bus Interface Unit in yIF and yDM) (input and output)

The clock rate of the CPU is determined based on the longest path that an instruction takes. As well as learning how to build different modules to create my CPU's automation, this lab continued to teach me how to use the concept of structural modeling.

References:

No external references were used for this lab other than the lab material provided to us at the beginning of the lab.

Appendix:

The source codes for all the experiments as well as the auxiliary programs are included below.

LabN1.v:

```
module labN1;
reg [31:0] PCin;
reg RegDst, RegWrite, clk, ALUSrc, INT, MemRead, MemWrite, Mem2Reg;
reg [2:0] op;
wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;
wire [25:0] jTarget;
wire zero;
wire [15:0] zeros, ones;

assign zeros = 16'h0000;
assign ones =  16'hffff;

yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);
yPC myPC(PCin, PCp4, INT, entryPoint, imm, jTarget, zero, branch, jump);
assign wd = wb;

initial
begin
  entryPoint = 128; INT = 1; #1;
        PCin = 128;
        repeat(43)
        begin
                clk = 1; #1; INT = 0; branch = 0; jump = 0;
                RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;

                if(ins[31:26] == 0) //R-type
                begin
```

```
                RegDst = 1; RegWrite = 1; ALUSrc = 0;
                if(ins[5:0] == 'h20) // add instruction
                begin
                        RegDst = 1; RegWrite = 1; ALUSrc = 0;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end

                else if(ins[5:0] == 'h25)
                begin
                        op = 3'b001;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end
        end // J-type

        else if(ins[31:26] == 2 || ins[31:26] == 3)
        begin
                jump = 1;
        end

        else // I-type
        begin
                RegDst = 1; RegWrite = 1; ALUSrc = 1;
                if(ins[31:26] == 'h4) // beq instruction
                begin
                        ALUSrc = 0; op = 3'b110;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                        branch = 1;
                end

                else if(ins[31:26] == 'h23) // lw instruction
                begin
                        RegDst = 0; RegWrite = 1; ALUSrc = 1;
                        MemRead = 1; MemWrite = 0; Mem2Reg = 1;
                end

                else if(ins[31:26] == 'h8) // addi instruction
                begin
                        RegDst = 0; RegWrite = 1; ALUSrc = 1;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end

        end

        clk = 0; #1;
    $display("%h: rd1=%2d rd2=%2d z=%3d zero=%b wb=%2d", ins, rd1, rd2, z, zero, wb);

    $finish;
end
endmodule
```

LabyPC.v

```
module yPC(PCin, PCp4, INT, entryPoint, imm, jTarget, zero, branch, jump);

output [31:0] PCin;
```

```
// entryPoint, 32-bit containing the addres to swich to
input [31:0] PCp4, entryPoint, imm;
input [25:0] jTarget;
// INT, 1bit signal, stop the current program
input INT, zero, branch, jump;

wire [31:0] immX4, bTarget, choiceA, choiceB;
wire doBranch, zf;                              // zf is a dummy

assign immX4[31:2] = imm[29:0];
assign immX4[1:0] = 2'b00;
assign jTargetX4[31:28] = PCp4[31:28];
assign jTargetX4[27:2] = jTarget[25:0];
assign jTargetX4[1:0] = 2'b00;

yAlu myALU(bTarget, zf, PCp4, immX4, 3'b010);
and (doBranch, branch, zero);
yMux #(32) mux1(choiceA, PCp4, bTarget, doBranch);
yMux #(32) mux2(choiceB, choiceA, {PCp4[31:28], jTarget[25:0], 2'b00}, jump);
yMux #(32) mux3(PCin, choiceB, entryPoint, INT);

endmodule
```

LabyC1.v

```
module yC1(rtype, lw, sw, jump, branch, opCode);

output rtype, lw, sw, jump, branch;
input [5:0] opCode;
wire not4, not3, not2;

not (not5, opCode[5]);
not (not4, opCode[4]);
not (not3, opCode[3]);
not (not2, opCode[2]);
not (not1, opCode[1]);
not (not0, opCode[0]);
and (rtype, not1, not2, not3, not4, not5, not0);
and (lw, opCode[5], not4, not3, not2, opCode[1], opCode[0]);          // lw, 100011
and (sw, opCode[5], not4, opCode[3], not2, opCode[1], opCode[0]);     // sw, 101011
and (jump, not5, not4, not3, not2, opCode[1], not0);                 // jump, 000010
and (branch, not5, not4, not3, opCode[2], not1, not0);               // branch, 000100
endmodule
```

LabyC2.v

```
module yC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite, rtype, lw, sw, branch);

output RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite;
input rtype, lw, sw, branch;

assign RegDst = rtype;
nor (ALUSrc, rtype, branch);
nor (RegWrite, sw, branch);
assign Mem2Reg = lw;
assign MemWrite = sw;
```

endmodule

LabN2.v

```
module labN2;
reg [31:0] PCin;
reg RegDst, RegWrite, clk, ALUSrc, INT, MemRead, MemWrite, Mem2Reg;
reg [2:0] op;
wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;
wire [25:0] jTarget;
wire zero;
wire [15:0] zeros, ones;

assign zeros = 16'h0000;
assign ones =  16'hffff;

yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);
yPC myPC(PCin, PCp4, INT, entryPoint, imm, jTarget, zero, branch, jump);
assign wd = wb;

assign opCode = ins[31:26];
yC1 myC1(rtype, lw, sw, jump, branch, opCode);
yC2 myC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite,
        rtype, lw, sw, branch);
initial
begin
  entryPoint = 128; INT = 1; #1;
        PCin = 128;
        repeat(43)
        begin
                clk = 1; #1; INT = 0; branch = 0; jump = 0;
                RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;

                if(ins[31:26] == 0) //R-type
                begin
                        RegDst = 1; RegWrite = 1; ALUSrc = 0;
                        if(ins[5:0] == 'h20) // add instruction
                        begin
                                RegDst = 1; RegWrite = 1; ALUSrc = 0;
                                MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                        end

                        else if(ins[5:0] == 'h25)
                        begin
                                op = 3'b001;
                                MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                        end
                end // J-type

                else if(ins[31:26] == 2 || ins[31:26] == 3)
                begin
```

```
                jump = 1;
            end

            else // I-type
            begin
                RegDst = 1; RegWrite = 1; ALUSrc = 1;
                if(ins[31:26] == 'h4) // beq instruction
                begin
                    ALUSrc = 0; op = 3'b110;
                    MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                    branch = 1;
                end

                else if(ins[31:26] == 'h23) // lw instruction
                begin
                    RegDst = 0; RegWrite = 1; ALUSrc = 1;
                    MemRead = 1; MemWrite = 0; Mem2Reg = 1;
                end

                else if(ins[31:26] == 'h8) // addi instruction
                begin
                    RegDst = 0; RegWrite = 1; ALUSrc = 1;
                    MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end

            end

            clk = 0; #1;
        $display("%h: rd1=%2d rd2=%2d z=%3d zero=%b wb=%2d", ins, rd1, rd2, z, zero, wb);

        $finish;
end
endmodule
```

LabyC3.v

```
module yC3(ALUop, rtype, branch);

output [1:0] ALUop;
input rtype, branch;

// assign ALUop = {rtype, branch};
assign ALUop[1] = rtype;
assign ALUop[0] = branch;

endmodule
```

LabyC4.v

```
module yC4(op, ALUop, fnCode);

output [2:0] op;
input [5:0] fnCode;
input [1:0] ALUop;
wire upperAnd, invertF2, invertOp1, leftOr;
```

```
or left(leftOr, fnCode[0], fnCode[3]);
or rightUpperOr(op[2], ALUop[0], upperAnd);
or rightLowerOr(op[1], invertOp1, invertF2);
and upperAnd(upperAnd, ALUop[1], fnCode[1]);
and lowerAnd(op[0], ALUop[1], leftOr);
not invertOp1(invertOp1, ALUop[1]);
not invertF2(invertF2, fnCode[2]);

endmodule
```

LabN3.v

```
module labN3;
reg [31:0] PCin;
reg RegDst, RegWrite, clk, ALUSrc, INT, MemRead, MemWrite, Mem2Reg;
reg [2:0] op;
wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;
wire [25:0] jTarget;
wire zero;
wire [15:0] zeros, ones;

assign zeros = 16'h0000;
assign ones =  16'hffff;

yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);
yPC myPC(PCin, PCp4, INT, entryPoint, imm, jTarget, zero, branch, jump);
assign wd = wb;

assign opCode = ins[31:26];
yC1 myC1(rtype, lw, sw, jump, branch, opCode);
yC2 myC2(RegDst, ALUSrc, RegWrite, Mem2Reg, MemRead, MemWrite,
        rtype, lw, sw, branch);

assign fnCode = ins[5:0];
yC3 myC3(ALUop, rtype, branch);
yC4 myC4(op, ALUop, fnCode);

initial
begin
  entryPoint = 128; INT = 1; #1;
        PCin = 128;
        repeat(43)
        begin
                clk = 1; #1; INT = 0; branch = 0; jump = 0;
                RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;

                if(ins[31:26] == 0) //R-type
                begin
                        RegDst = 1; RegWrite = 1; ALUSrc = 0;
                        if(ins[5:0] == 'h20) // add instruction
                        begin
                                RegDst = 1; RegWrite = 1; ALUSrc = 0;
```

```
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end

                else if(ins[5:0] == 'h25)
                begin
                        op = 3'b001;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end
        end // J-type

        else if(ins[31:26] == 2 || ins[31:26] == 3)
        begin
                jump = 1;
        end

        else // I-type
        begin
                RegDst = 1; RegWrite = 1; ALUSrc = 1;
                if(ins[31:26] == 'h4) // beq instruction
                begin
                        ALUSrc = 0; op = 3'b110;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                        branch = 1;
                end

                else if(ins[31:26] == 'h23) // lw instruction
                begin
                        RegDst = 0; RegWrite = 1; ALUSrc = 1;
                        MemRead = 1; MemWrite = 0; Mem2Reg = 1;
                end

                else if(ins[31:26] == 'h8) // addi instruction
                begin
                        RegDst = 0; RegWrite = 1; ALUSrc = 1;
                        MemRead = 0; MemWrite = 0; Mem2Reg = 0;
                end

        end

        clk = 0; #1;
$display("%h: rd1=%2d rd2=%2d z=%3d zero=%b wb=%2d", ins, rd1, rd2, z, zero, wb);

$finish;
end
endmodule
```

LabN4.v

```
module labN4;

reg [31:0] entryPoint;
reg clk, INT;
wire [31:0] ins, rd2, wb;
yChip myChip(ins, rd2, wb, entryPoint, INT, clk);

initial
```

```
begin
	// --------------------entry point
	assign entryPoint = 32'h80;
	assign INT = 1; #1;
	#1 assign INT = 0;                                          // unset interrupt signal
	// --------------------run program
	repeat(43)
	begin
		// ----------------fetch an ins
		clk = 1; #1;
		// ----------------set control signals
		// ----------------execute the ins
		clk = 0; #1;
		// ----------------view results
		$display("%h: rd2=%2d wb=%2d", ins, rd2, wb);
		// ----------------to prepare for the next instruction
	end
	$finish;
end

endmodule
```