LAB 9:

Modeling a Sequence Controller

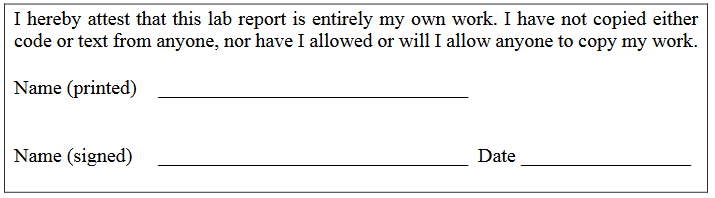
Professor Ronald Mehler

ECE 526L

Spring 2018

Garen Nikoyan

5/10/2018



**Objective:**

The objective of this lab is to create a pair of modules that will later be used as the phase generator and sequence controller for a RISC CPU design. The phase generator will cycle through four instructions, FETCH, DECODE, EXECUTE, and UPDATE. Based on the OPCODE, the sequence controller will complete a certain function.

**Methodology:**

This lab was one of the most challenging designs. One of the things that made this lab

difficult is first trying to understand the process of the RISC-Y Processor and how it was going to function for the design to be built. One of the many interesting things about the RISC-Y processor is that it has 13 different outputs in which it will execute accordingly. The tasks to be executed will be according to the outputs that are active at the time of the simulation of the entire design.

The input also consists of many inputs, in which we call as the OPCODE this along with

the address is what allows what outputs to be active at a certain time. The opcodes can be found in the lab manual. Like mentioned earlier, each instruction takes four clock cycles of which are FETCH, DECODE, EXECUTE and UPDATE. One thing to note is that these phases are all enumerated types.

In the FETCH phase I enabled the instruction register. In the DECODE phase using a case statement I could give the instruction for every opcode along with every branch statement. In these cases, I set up the necessary ports that are going to be needed for that certain opcode.

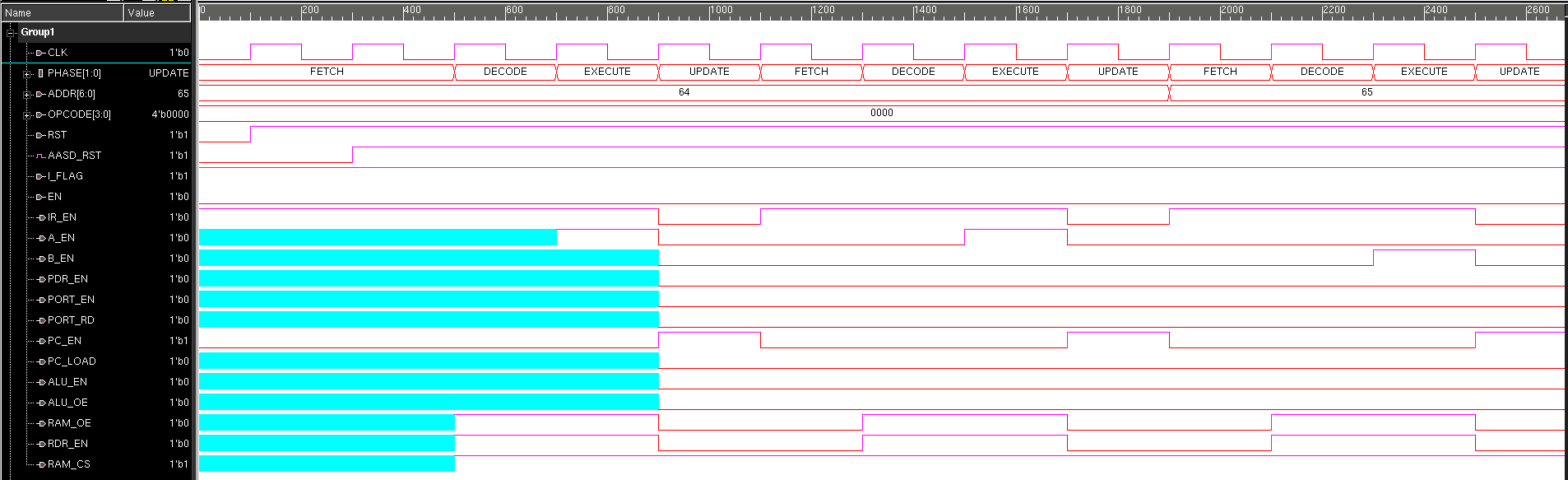
For example, the case for ADD I enabled A\_EN and B\_EN which are which are the inputs for the A and B from the ALU. In the EXECUTE phase I enabled the corresponding elements that allow for the elements enabled in the decode phase. For example, in the decode phase since A\_EN and B\_EN were enabled, now for those to do their respective applications in the execute phase ALU\_EN was enabled. The last cycle/phase out of the four cycles, here all we do is just make sure to set the outputs back to zero to be ready for the next task and set PC\_EN high.

**Analysis:**

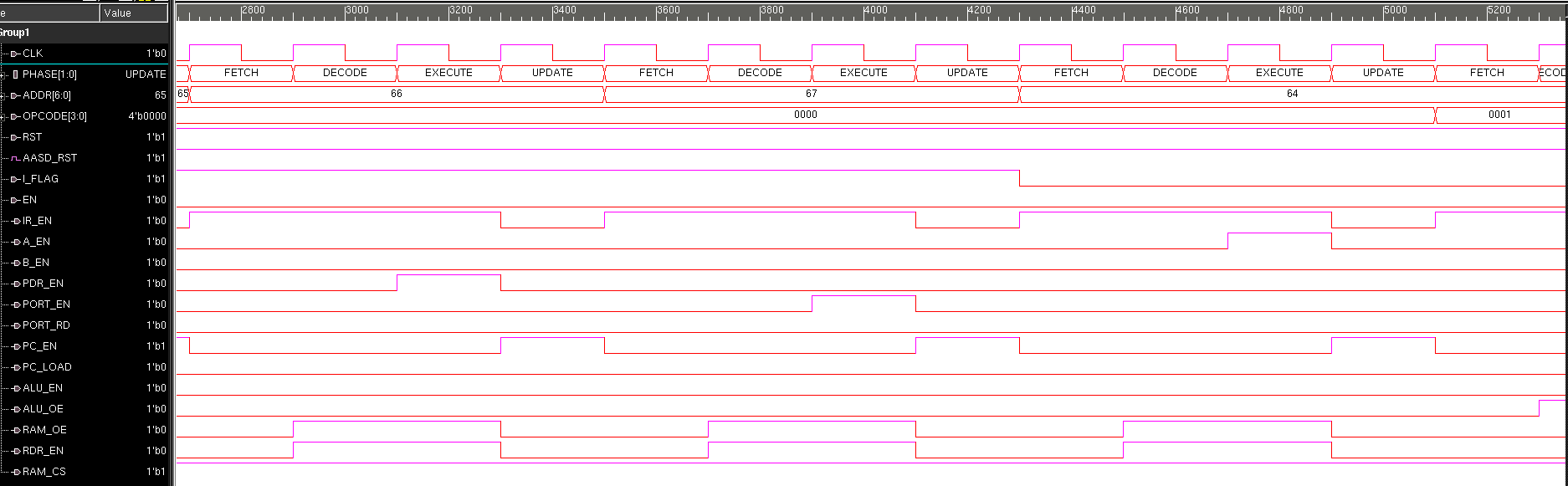
Looking at waveforms and the log, it can be seen that the modules seem to have worked correctly. The sequence controller is difficult to test until it is completely integrated within a processor. The necessary opcodes were ran through to ensure that each output toggled at least once. All of the opcodes were tested, with the exception of the ALU specific ones, where only ADD was tested. This was done to keep a smaller test bench, since if one of those worked correctly, we could safely assume the rest would as well. For STORE and LOAD, different addresses were tested to ensure that outputs were correct.

In Figure 1, the opcode LOAD is tested with address 64d and 65d. In Figure 2, LOAD is tested again with addresses 66d and 67d. IR\_EN, PC\_EN are toggling as expected, as are PDR\_EN, RAM\_OE, and RAM\_CS, since these would be controlling the read and write from RAM. A\_EN and B\_EN are also toggling with their respective addresses.

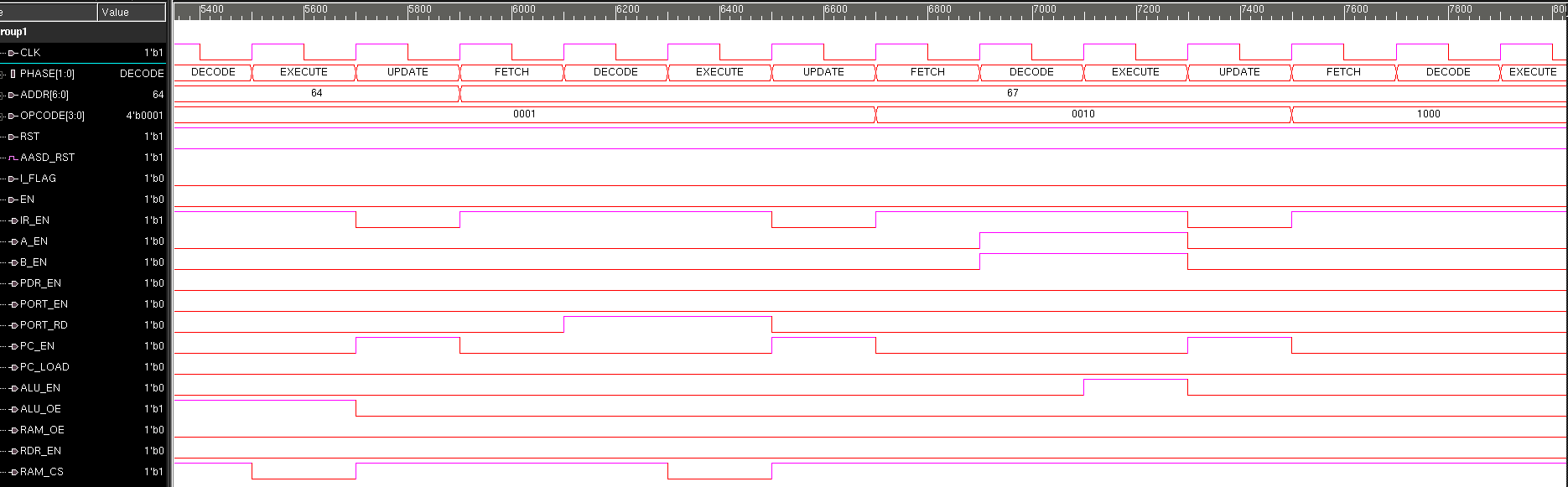
In Figure 3, STORE is tested with address 64, and 67, followed by ADD and B, with address 67. During ADD, it can be seen that A\_EN and B\_EN go high, followed by ALU\_EN, showing the ALU opcode works properly. In Figure 4, the branch opcodes are tested, albeit incorrectly. The flags were not set high after being initialized to 0, to see if PC\_LOAD will properly follow the flags value.



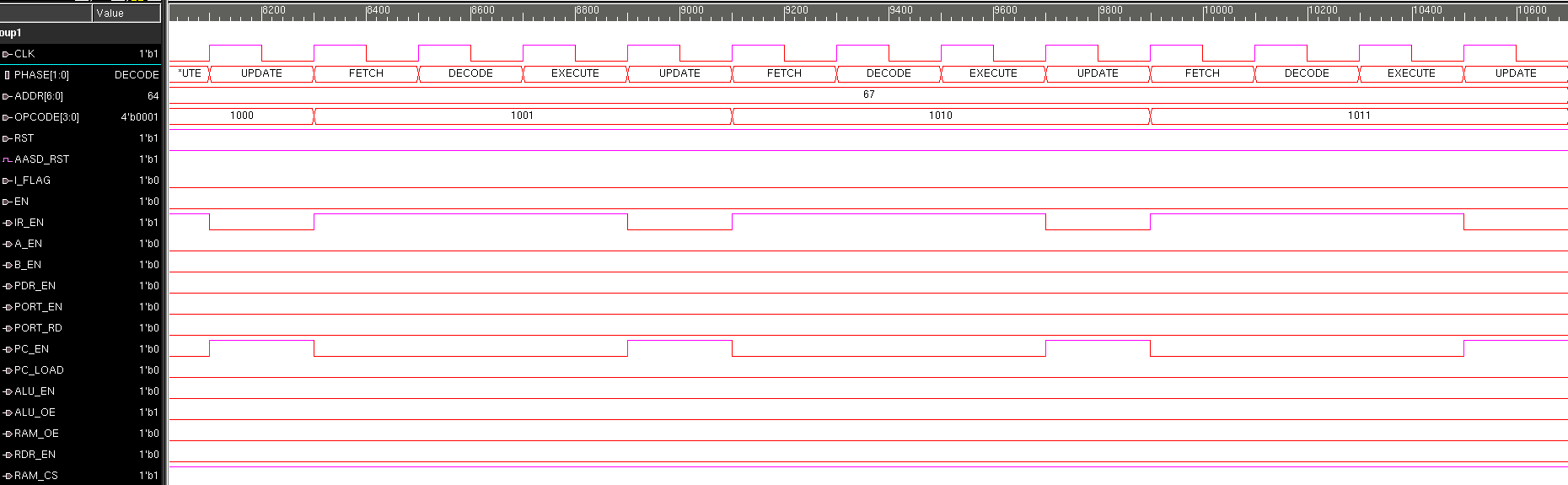
*Figure 1: LOAD at addr 64 and 65*



*Figure 2: LOAD at addr 66 and 67.*



*Figure 3: STORE at addr 64 and 67, followed by ADD and B at addr 67.*



*Figure 4: BZ, BN,and BV at addr 67.*

**Modules:**

**toplevel.sv**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* ECE526L Experiment #9 Garen Nikoyan, Spring 2018

\*\*\* Modeling a Sequence Controller

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: toplevel.sv Created by: Garen Nikoyan, 4/26/2018 \*\*\*

\*\*\* -Revision History

\*\*\* 4/26/2018: First draft

\*\*\* 5/8/2018: Changed branching in sequence\_controller

\*\*\* 5/10/2018: Needs proper testing, error in test vectors, branch flags

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* This module instantiates a sequence\_controller, AASD, and phase\_generator

\*\*\* It is meant to fit into a larger design, such as processor.

\*\*\* This module will provide the proper enable lines for certian opcodes

\*\*\* throughout FETCH, DECODE, EXECUTE, and UPDATE cycles

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

`timescale 1ns/100ps

`default\_nettype none

package cycle\_package;

typedef enum [1:0] {FETCH =0, DECODE = 1, EXECUTE = 2, UPDATE = 3} CYCLE;

endpackage

module toplevel(ADDR,OPCODE,I\_FLAG,ZF,NF,OF,CF,IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS,CLK,RST,EN);

import cycle\_package::\*; //imports package

input [6:0] ADDR;

input [3:0] OPCODE;

input I\_FLAG,ZF,NF,OF,CF,RST,CLK,EN;

output reg IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS;

CYCLE PHASE;

wire AASD\_RST;

sequence\_controller seq\_cont(ADDR,OPCODE,PHASE,I\_FLAG,ZF,NF,OF,CF,IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS);

AASD aasd\_seq(AASD\_RST,CLK,RST);

phaser phase\_gen(CLK,AASD\_RST,EN,PHASE);

endmodule

**sequence\_controller.sv**

`timescale 1ns/100ps

`default\_nettype none

package cycle\_package;

typedef enum [1:0] {FETCH =0, DECODE = 1, EXECUTE = 2, UPDATE = 3} CYCLE;

endpackage

module sequence\_controller(ADDR,OPCODE,PHASE,I\_FLAG,ZF,NF,OF,CF,IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS);

import cycle\_package::\*;

input [6:0] ADDR;

input [3:0] OPCODE;

input CYCLE PHASE;

input I\_FLAG,ZF,NF,OF,CF;

output reg IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS;

reg branches;

// Instruction set

localparam LOAD = 0,

STORE = 1,

ADD = 2,

SUB = 3,

AND = 4,

OR = 5,

XOR = 6,

NOT = 7,

B = 8,

BZ = 9,

BN = 10,

BV = 11,

BC = 12;

always @(PHASE,I\_FLAG,ZF,NF,OF,CF,ADDR,OPCODE)

case (PHASE)

// FETCH

FETCH:

begin

IR\_EN=1'b1;

PC\_EN=1'b0;

end

// DECODE

DECODE:

begin

case (OPCODE)

LOAD:

begin

RAM\_OE <= 1'b1;

RAM\_CS <= 1'b1;

RDR\_EN <= 1'b1;

end

STORE:

begin

if(ADDR==67) PORT\_RD <= 1'b1;

else ALU\_OE <= 1'b1;

end

ADD:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

SUB:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

AND:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

OR:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

XOR:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

NOT:

begin

A\_EN = 1'b1;

B\_EN = 1'b1;

end

endcase

end

// EXECUTE

EXECUTE:

begin

case (OPCODE)

LOAD:

begin

case (ADDR)

64: A\_EN = 1'b1;

65: B\_EN =1'b1;

66: PDR\_EN <=1'b1;

67: PORT\_EN <=1'b1;

endcase

end

STORE:

begin

RAM\_CS <= 1'b0;

end

ADD:

begin

ALU\_EN <=1'b1;

end

SUB:

begin

ALU\_EN <=1'b1;

end

AND:

begin

ALU\_EN <=1'b1;

end

OR:

begin

ALU\_EN <=1'b1;

end

XOR:

begin

ALU\_EN <=1'b1;

end

NOT:

begin

ALU\_EN <=1'b1;

end

endcase

end

// UPDATE

UPDATE: begin

case(OPCODE)

B: PC\_LOAD=1'b1;

BZ: PC\_LOAD=ZF;

BN: PC\_LOAD=NF;

BV: PC\_LOAD=OF;

BC: PC\_LOAD=CF;

default: PC\_LOAD=1'b0;

endcase

IR\_EN <= 1'b0;

A\_EN = 1'b0;

B\_EN = 1'b0;

PDR\_EN <= 1'b0;

PORT\_EN <= 1'b0;

PORT\_RD <= 1'b0;

ALU\_EN <= 1'b0;

ALU\_OE <= 1'b0;

RAM\_OE <= 1'b0;

RDR\_EN <= 1'b0;

PC\_EN=1'b1;

PC\_LOAD=1'b0;

RAM\_CS <= 1'b1;

end

// DEFAULT

default: begin

IR\_EN <= 1'b0;

A\_EN = 1'b0;

B\_EN = 1'b0;

PDR\_EN <= 1'b0;

PORT\_EN <= 1'b0;

PORT\_RD <= 1'b0;

ALU\_EN <= 1'b0;

ALU\_OE <= 1'b0;

RAM\_OE <= 1'b0;

RDR\_EN <= 1'b0;

PC\_EN=1'b0;

PC\_LOAD=1'b0;

RAM\_CS <= 1'b1;

end

endcase

endmodule

**phaser.sv**

`timescale 1ns/100ps

`default\_nettype none

package cycle\_package;

typedef enum [1:0] {FETCH =0, DECODE = 1, EXECUTE = 2, UPDATE = 3} CYCLE;

endpackage

module phaser(CLOCK, RESET, EN, PHASE);

import cycle\_package::\*;

input CLOCK, RESET, EN;

output CYCLE PHASE;

always @(posedge CLOCK, negedge RESET) begin

if(!RESET)

PHASE <= PHASE.first();

else if(!EN)

PHASE <= PHASE.next();

else

PHASE <= PHASE;

end

endmodule

**AASD.sv**

`timescale 1ns/100ps

`default\_nettype none

package cycle\_package;

typedef enum [1:0] {FETCH =0, DECODE = 1, EXECUTE = 2, UPDATE = 3} CYCLE;

endpackage

module AASD(AASD\_RST,CLOCK,RESET);

output reg AASD\_RST;

input CLOCK,RESET;

reg Out1; //this is the wire for the output of the first FF

always @ (posedge CLOCK or negedge RESET) //allows 4 synchronous/asynchronous

if(!RESET)begin//asynchronous reset

Out1 <= 1'b0;

AASD\_RST <= 1'b0;

end

else begin

Out1 <= 1'b1;

AASD\_RST <= Out1;

end

endmodule

**Testbench:**

**toplevel\_tb.sv**

`timescale 1ns/100ps

`default\_nettype none

module toplevel\_tb();

reg [6:0] ADDR;

reg I\_FLAG,ZF,NF,OF,CF,RST,CLK,EN;

wire IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS;

enum reg[3:0] {LOAD, STORE, ADD, SUB, AND, OR, XOR, NOT, B, BZ, BN, BV, BC} OPCODE;

toplevel UUT(ADDR,OPCODE,I\_FLAG,ZF,NF,OF,CF,IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS,CLK,RST,EN);

initial //makes the clock

begin

CLK=1'b0;

forever #10 CLK=~CLK;

end

initial

begin

$monitor("OUTPUTS: %d ns\tIR\_EN=%d A\_EN=%d B\_EN=%d PDR\_EN=%d PORT\_EN=%d PORT\_RD=%d PC\_EN=%d PC\_LOAD=%d ALU\_EN=%d ALU\_OE=%d RAM\_OE=%d RDR\_EN=%d RAM\_CS=%d",$time,IR\_EN,A\_EN,B\_EN,PDR\_EN,PORT\_EN,PORT\_RD,PC\_EN,PC\_LOAD,ALU\_EN,ALU\_OE,RAM\_OE,RDR\_EN,RAM\_CS);

$vcdpluson;

end

initial begin

$monitoroff;

RST = 1'b0; ADDR = 64; CF = 0; OF = 0; NF = 0; ZF = 0; I\_FLAG = 1; OPCODE = 0; EN=1'b0; //initial states

#10 RST = 1'b1;

#110 $monitoron;

#70 ADDR = 65;

#80 ADDR = 66;

#80 ADDR = 67;

#80 I\_FLAG = 0; ADDR = 64;

#80 OPCODE = 1; ADDR = 64;

#80 OPCODE = 1; ADDR = 67;

#80 OPCODE = 2;

#80 OPCODE = 8; CF = 0; OF = 0; NF = 0; ZF = 0;

#80 OPCODE = 9;

#80 OPCODE = 10;

#80 OPCODE = 11;

#80 OPCODE = 12;

$finish;

end

endmodule

**Log:**

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version M-2017.03-SP1\_Full64; Runtime version M-2017.03-SP1\_Full64; May 8 21:12 2018

VCD+ Writer M-2017.03-SP1\_Full64 Copyright (c) 1991-2017 by Synopsys Inc.

OUTPUTS: 120 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 130 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 150 ns IR\_EN=1 A\_EN=1 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 170 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 190 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 210 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 230 ns IR\_EN=1 A\_EN=0 B\_EN=1 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 250 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 270 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 290 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 310 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=1 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 330 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 350 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 370 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 390 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=1 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 410 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 430 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 450 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 470 ns IR\_EN=1 A\_EN=1 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=1 RDR\_EN=1 RAM\_CS=1

OUTPUTS: 490 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 510 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 530 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=1 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 550 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=1 RAM\_OE=0 RDR\_EN=0 RAM\_CS=0

OUTPUTS: 570 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 590 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 610 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=1 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 630 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=1 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=0

OUTPUTS: 650 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 670 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 690 ns IR\_EN=1 A\_EN=1 B\_EN=1 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 710 ns IR\_EN=1 A\_EN=1 B\_EN=1 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=1 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 730 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 750 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 810 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 830 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 890 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 910 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 970 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 990 ns IR\_EN=1 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=0 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

OUTPUTS: 1050 ns IR\_EN=0 A\_EN=0 B\_EN=0 PDR\_EN=0 PORT\_EN=0 PORT\_RD=0 PC\_EN=1 PC\_LOAD=0 ALU\_EN=0 ALU\_OE=0 RAM\_OE=0 RDR\_EN=0 RAM\_CS=1

$finish called from file "SeqConTB.sv", line 44.

$finish at simulation time 10700

V C S S i m u l a t i o n R e p o r t

Time: 1070000 ps

CPU Time: 0.210 seconds; Data structure size: 0.0Mb

Tue May 8 21:12:41 2018

**Conclusion:**

Everything in this lab seems to have worked as expected, except for the branches opcodes which was tested improperly. The modules created for this lab will be truly put to the test in Lab 10, when they are integrated within a processor.