ENG 363

Lab #3 Simple Processor

Brian Worts and Chris Jenson

11/7/19



**Table of Contents**

Introduction………………………………………………………………………………….. 2

Procedure/Results…...……………………………………………………………………….. 2

Deliverables……...…...…………………………………………………………………...….. 5

Discussion/Conclusion ……………………………………………………………………….14

Appendix A (*Processor*).......................................................................................................... 14

Appendix B (Datapath)............................................................................................................ 16

Appendix C (*ALU*)................................................................................................................... 19

Appendix D (*Controller*).......................................................................................................... 20

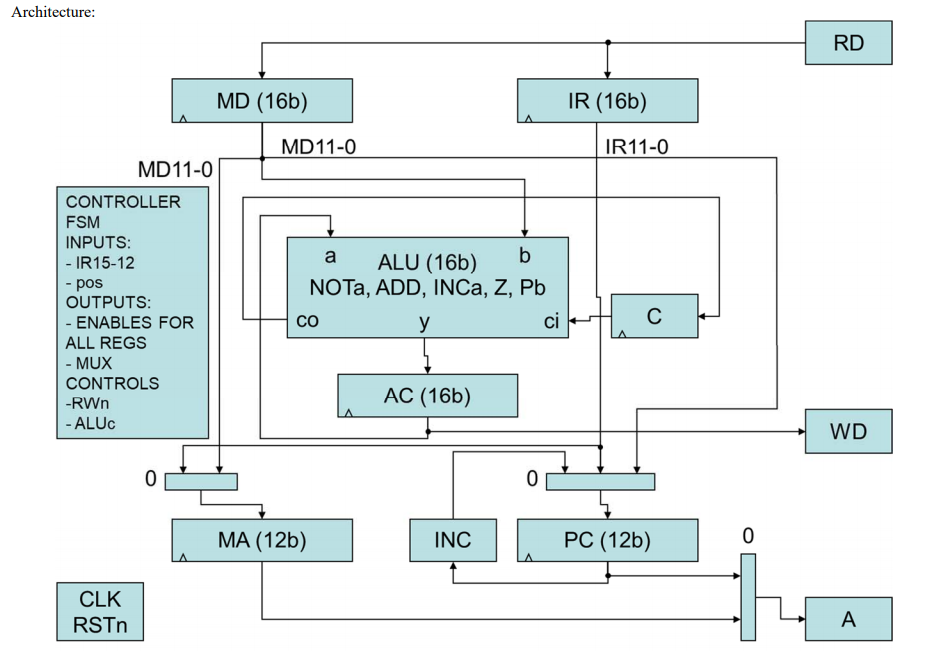
Appendix E (*Test Bench*).......................................................................................................... 24

**Intro:**

Computers are playing an ever increasing role in our society, and the processor is at the core of this innovation. They are becoming increasingly complex and fast while also shrinking in size. However, in order to have a good understanding of a complex processor, one must have a much better grasp on a simple one. Thus, in this lab a 16 bit simple processor was designed using a minimum CPI Von Newman approach.

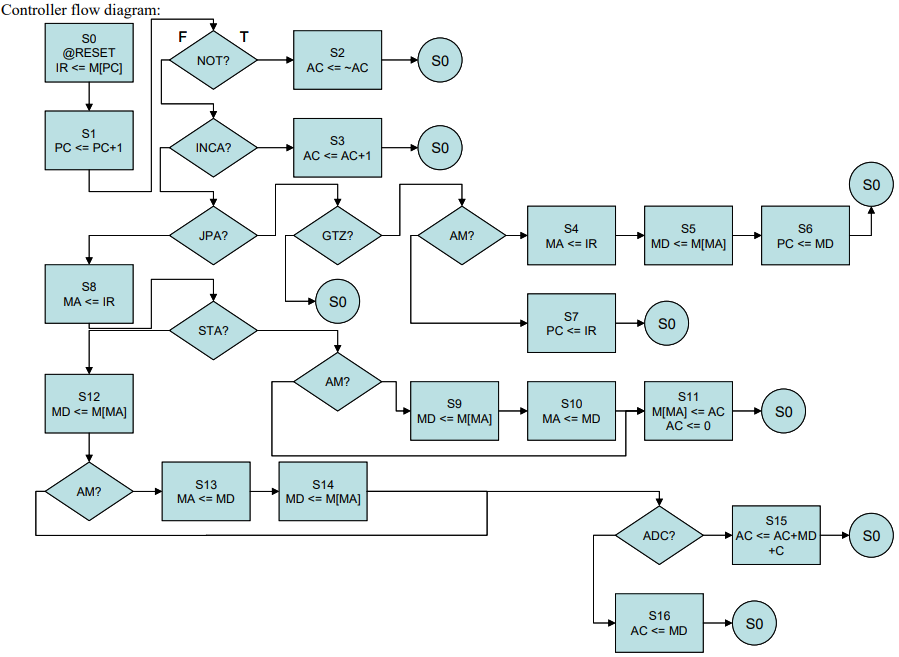
**Procedure and Results:**

The first step of this lab was to understand the supplied architecture of the processor, shown in *Figure 1*. This architecture helped give an understanding of the basic components that make up the processor as well as how the components are connected.



***Figure 1:*** *The processor architecture*

In order to simplify the architecture, the state control flow diagram was also provided, in *Figure 2*. This diagram gives a better idea as to how the processor should function by representing the program as a state machine



***Figure 2:*** *The control flow of the processor*

Next, the processor was designed and tested in Verilog HDL. In order to implement, the code was split up into 4 modules: The processor (*Appendix A,*) the datapath (*Appendix B*), the ALU (*Appendix C,*)and the controller *(Appendix D.)* Furthermore, a testbench was written to allow for the design to be simulated and tested (*Appendix E.*)

Finally, the design was implemented on a Xilinx FPGA board with the clock set at 1MHz. This allowed us to test the design and get data in an actual application of it as opposed to the virtual application in the Vivado tool.

**Deliverables:**

1. **RTL for each instruction and addressing mode combination.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **1** | **2** | **3** | **4** | **5** |
| **(for all)** | **IR <= M[PC]** | **PC <= PC + 1** |  |  |  |
| **NOT** | **6** |  |  |  |  |
| **ADC (direct)** | **MA <= IR** | **MD <= M[MA]** | **AC <= AC + MD + C** |  |  |
| **ADC (indirect)** | **MA <= IR** | **MD <= M[MA]** | **MA <= MD** | **MD <= M[MA]** | **AC <= AC + MD + C** |
| **JPA (direct)** | **PC <= IR** |  |  |  |  |
| **JPA (indirect)** | **MA <= IR** | **MD <= M[MA]** | **PC <= MD** |  |  |
| **INCA** | **AC <= AC + 1** |  |  |  |  |
| **STA (direct)** | **MA <= IR** | **M[MA] <= AC** | **AC <= 0** |  |  |
| **STA (indirect)** | **MA <= IR** | **MD <= M[MA]** | **MA <= MD** | **M[MA] <= AC** | **AC <= 0** |
| **LDA (direct)** | **MA <= IR** | **MD <= M[MA]** | **AC <= MD** |  |  |
| **LDA (indirect)** | **MA <= IR** | **MD <= M[MA]** | **MA <= MD** | **MD <= M[MA]** | **AC <= MD** |

1. **Number of clock cycles for each instruction and addressing mode combination.**

|  |  |  |
| --- | --- | --- |
| **OpCode** | **AM = 0** | **AM = 1** |
| **NOT** | **3** | **3** |
| **ADC** | **5** | **7** |
| **JPA** | **3** | **5** |
| **INCA** | **3** | **3** |
| **STA** | **5** | **7** |
| **LDA** | **5** | **7** |

1. **Your assembly language test program. All instructions and addressing mode combinations need to execute at least once.**

lda &l2

inca

jpa &l0

adc &l0

not

sta &l0

jpa \* &l1

lda \* &l0

adc \* &l0

sta \* &l2

(l0) $0

(l1) $1

(l2) $5

1. **Your machine language test program**

1010000000001100

0110000000000000

0100000000001010

0010000000001010

0000000000000000

1000000000001010

0101000000001011

1011000000001010

0011000000001010

1001000000001100

0000000000000000

0000000000000001

0000000000000101

A00C

6000

400A

200A

0000

800A

500B

B00A

300A

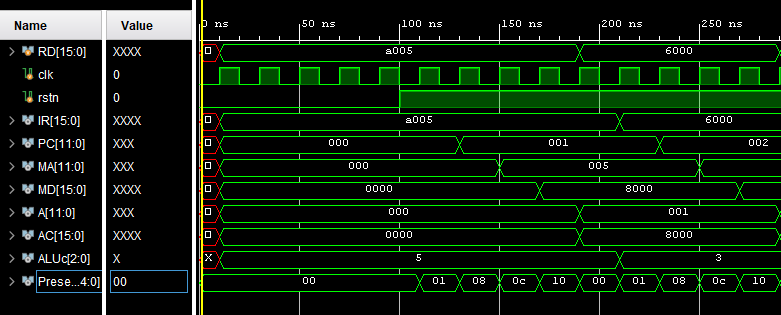
900C

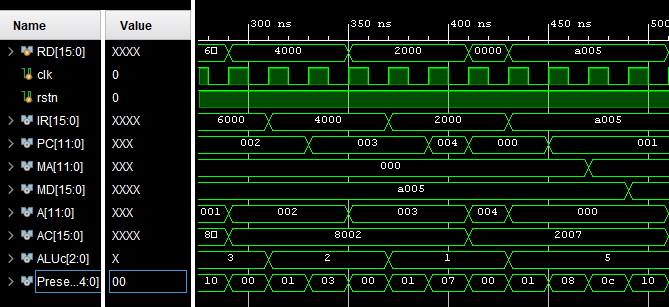
0000

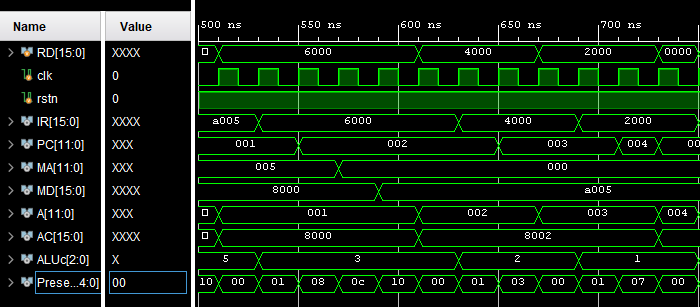
0001

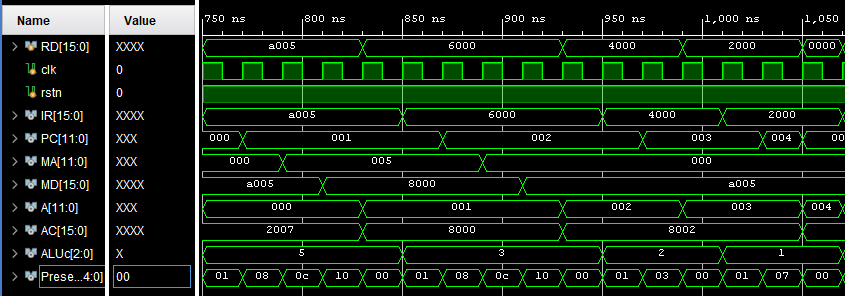
0005

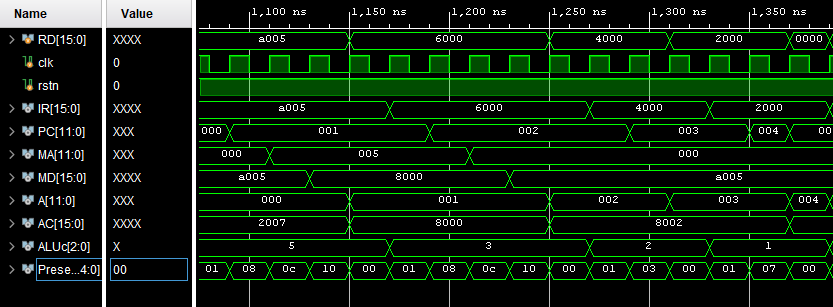
1. **The waveforms resulting from the verification of your design.**

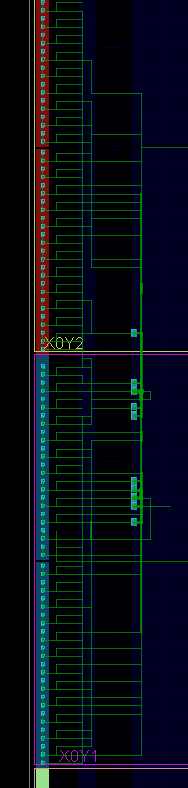
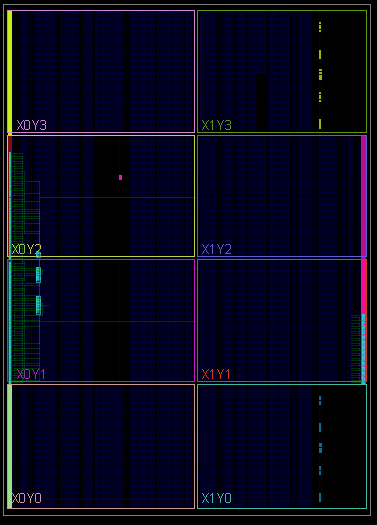
****

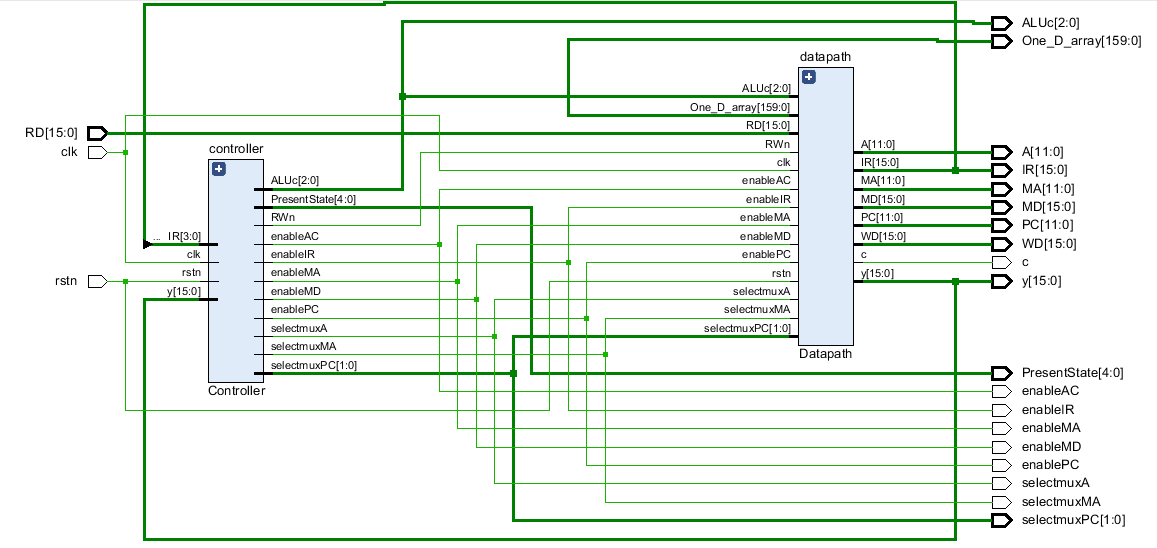
****

****

****

****

1. **The design schematics from the Xilinx synthesis of your design. Do not use any area constraints. Use a clock period of 1 μS as the timing constraint**
2. **Snapshot of the routed design**

****

1. **Post Place and Route timing report**

Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

--------------------------------------------------------------------------------------------------------------------

| Tool Version : Vivado v.2019.1 (win64) Build 2552052 Fri May 24 14:49:42 MDT 2019

| Date : Thu Nov 7 05:01:54 2019

| Host : MSI running 64-bit major release (build 9200)

| Command : report\_timing\_summary -max\_paths 10 -file processor\_timing\_summary\_routed.rpt -pb processor\_timing\_summary\_routed.pb -rpx processor\_timing\_summary\_routed.rpx -warn\_on\_violation

| Design : processor

| Device : 7a100t-csg324

| Speed File : -1 PRODUCTION 1.23 2018-06-13

---------------------------------------------------------------------------------------------------------------------

Timing Summary Report

------------------------------------------------------------------------------------------------

| Timer Settings

------------------------------------------------------------------------------------------------

Enable Multi Corner Analysis : Yes

Enable Pessimism Removal : Yes

Pessimism Removal Resolution : Nearest Common Node

Enable Input Delay Default Clock : No

Enable Preset / Clear Arcs : No

Disable Flight Delays : No

Ignore I/O Paths : No

Timing Early Launch at Borrowing Latches : No

Borrow Time for Max Delay Exceptions : Yes

Corner Analyze Analyze

Name Max Paths Min Paths

------ --------- ---------

Slow Yes Yes

Fast Yes Yes

check\_timing report

Table of Contents

-----------------

1. checking no\_clock

2. checking constant\_clock

3. checking pulse\_width\_clock

4. checking unconstrained\_internal\_endpoints

5. checking no\_input\_delay

6. checking no\_output\_delay

7. checking multiple\_clock

8. checking generated\_clocks

9. checking loops

10. checking partial\_input\_delay

11. checking partial\_output\_delay

12. checking latch\_loops

1. checking no\_clock

--------------------

There are 0 register/latch pins with no clock.

2. checking constant\_clock

-------------------------

There are 0 register/latch pins with constant\_clock.

3. checking pulse\_width\_clock

-----------------------------

There are 0 register/latch pins which need pulse\_width check

4. checking unconstrained\_internal\_endpoints

--------------------------------------------

There are 0 pins that are not constrained for maximum delay.

There are 0 pins that are not constrained for maximum delay due to constant clock.

5. checking no\_input\_delay

--------------------------

There is 1 input port with no input delay specified. (HIGH)

There are 0 input ports with no input delay but user has a false path constraint.

6. checking no\_output\_delay

---------------------------

There are 56 ports with no output delay specified. (HIGH)

There are 0 ports with no output delay but user has a false path constraint

There are 7 ports with no output delay but with a timing clock defined on it or propagating through it (LOW)

7. checking multiple\_clock

--------------------------

There are 3 register/latch pins with multiple clocks. (HIGH)

8. checking generated\_clocks

----------------------------

There are 0 generated clocks that are not connected to a clock source.

9. checking loops

-----------------

There are 0 combinational loops in the design.

10. checking partial\_input\_delay

--------------------------------

There are 0 input ports with partial input delay specified.

11. checking partial\_output\_delay

---------------------------------

There are 0 ports with partial output delay specified.

12. checking latch\_loops

------------------------

There are 0 combinational latch loops in the design through latch input

------------------------------------------------------------------------------------------------

| Design Timing Summary

------------------------------------------------------------------------------------------------

WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WHS(ns) THS(ns) THS Failing Endpoints THS Total Endpoints WPWS(ns) TPWS(ns) TPWS Failing Endpoints TPWS Total Endpoints

------- ------- --------------------- ------------------- ------- ------- ---------------------

0.000 0.000 0 86 0.013 0.000 0 86 499.500 0.000 0 47

All user specified timing constraints are met.

------------------------------------------------------------------------------------------------

| Clock Summary

------------------------------------------------------------------------------------------------

Clock Waveform(ns) Period(ns) Frequency(MHz)

----- ------------ ---------- --------------

clk {0.000 500.000} 1000.000 1.000

controller/Q[0] {0.000 500.000} 1000.000 1.000

controller/Q[1] {0.000 500.000} 1000.000 1.000

controller/Q[2] {0.000 500.000} 1000.000 1.000

clock {0.000 500.000} 1000.000 1.000

------------------------------------------------------------------------------------------------

| Intra Clock Table

------------------------------------------------------------------------------------------------

Clock WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WHS(ns) THS(ns) THS Failing Endpoints THS Total Endpoints WPWS(ns) TPWS(ns) TPWS Failing Endpoints TPWS Total Endpoints

--------------------- ------------------- -------- -------- ---------------------- --------------------

clk 997.493 0.000 0 40 0.201 0.000 0 40 499.500 0.000 0 44

controller/Q[0] 0.000 0.000 0 1 0.528 0.000 0 1 499.500 0.000 0 3

controller/Q[1] 0.000 0.000 0 1 0.487 0.000 0 1 499.500 0.000 0 3

controller/Q[2] 0.000 0.000 0 1 0.031 0.000 0 1 499.500 0.000 0 3

------------------------------------------------------------------------------------------------

| Inter Clock Table

------------------------------------------------------------------------------------------------

From Clock To Clock WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WHS(ns) THS(ns) THS Failing Endpoints THS Total Endpoints

---------- -------- ------- ------- --------------------- ------------------- -------

controller/Q[0] clk 497.551 0.000 0 27 0.330 0.000 0 27

controller/Q[1] clk 497.349 0.000 0 27 0.379 0.000 0 27

controller/Q[2] clk 496.862 0.000 0 43 0.355 0.000 0 43

controller/Q[1] controller/Q[0] 0.000 0.000 0 1 0.279 0.000 0 1

controller/Q[2] controller/Q[0] 0.000 0.000 0 1 0.077 0.000 0 1

controller/Q[0] controller/Q[1] 0.000 0.000 0 1 0.193 0.000 0 1

controller/Q[2] controller/Q[1] 0.000 0.000 0 1 0.013 0.000 0 1

controller/Q[0] controller/Q[2] 0.000 0.000 0 1 0.101 0.000 0 1

controller/Q[1] controller/Q[2] 0.000 0.000 0 1 0.123 0.000 0 1

------------------------------------------------------------------------------------------------

| Other Path Groups Table

------------------------------------------------------------------------------------------------

Path Group From Clock To Clock WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WHS(ns) THS(ns) THS Failing Endpoints THS Total Endpoints

------------------------------------------------------------------------------------------------

| Timing Details

---------------------------------------------------------------------------------------------------

From Clock: clk

To Clock: clk

Setup : 0 Failing Endpoints, Worst Slack 997.493ns, Total Violation 0.000ns

Hold : 0 Failing Endpoints, Worst Slack 0.201ns, Total Violation 0.000ns

PW : 0 Failing Endpoints, Worst Slack 499.500ns, Total Violation 0.000ns

**Discussion/Conclusion**

Throughout this project, the group encountered many challenges. The first of these challenges was the flow of information in the simulation, so the team developed a very efficient debugging method to deal with this. One team member would read the state flow, and the other would pause them when an error was present. The team could then follow the control and datapath at that state to find where there may have been an error. In the end, although a waveform was generated, there were still some imperfections within it, such as instructions repeating themselves.

Another challenge occurred was when developing the implementation. Errors were first received as a result of the use of multiple always loops that changed the same values. Thus, the code had to be restructured to combine all the values into one loop. Once completed, a new error occurred, that was the result of using too many I/O ports. The team had to optimize their design so that it was within the limits of Vivado, and eventually were successful in this regard.

This lab was extremely successful in immersing the team members in the Vivado software and allowed them to get a significant knowledge of the software and language.

**Appendix A - Processor**

`timescale 1ns / 1ps

module processor(RD,clk,rstn,WD,A,IR,MD,y,PC,MA,PresentState,enableIR,enableMD,enableAC,enablePC,enableMA,selectmuxMA,selectmuxPC,selectmuxA,ALUc,c, One\_D\_array);

input clk;

input rstn;

input [15:0] RD;

output [11:0] A;

output [15:0] IR;

output [11:0] PC;

output [11:0] MA;

output [15:0] MD;

output [15:0] WD;

output [15:0] y;

output enableAC;

output enableIR;

output enableMA;

output enableMD;

output enablePC;

output [2:0] ALUc;

output selectmuxA;

output selectmuxMA;

output [1:0] selectmuxPC;

output [4:0] PresentState;

output c;

output wire [159:0]One\_D\_array;

wire [15:0] IR;

wire [11:0] PC;

wire [11:0] MA;

wire [15:0] MD;

wire [11:0] A;

wire [15:0] y;

wire clk;

wire rstn;

wire [2:0] ALUc;

wire enableAC;

wire enableIR;

wire enableMA;

wire enableMD;

wire enablePC;

wire RWn;

wire selectmuxA;

wire selectmuxMA;

wire [1:0] selectmuxPC;

Datapath datapath(RD,clk,rstn,enableMD,enableAC,enablePC,enableIR,enableMA,selectmuxMA,selectmuxPC,selectmuxA,ALUc,WD,A,RWn,PC,MA,y,IR,MD,c, One\_D\_array);

Controller controller(IR[15:12],clk,rstn,y,enableMD,enableAC,enablePC,enableIR,enableMA,ALUc,RWn,selectmuxMA,selectmuxPC,selectmuxA,PresentState);

endmodule

**Appendix B - Datapath**

`timescale 1ns / 1ps

module Datapath(RD,clk,rstn,enableMD,enableAC,enablePC,enableIR,enableMA,selectmuxMA,selectmuxPC,selectmuxA,ALUc,WD,A,RWn,PC,MA,y,IR,MD,c, One\_D\_array);

input clk,rstn;

input [15:0] RD;

input enableAC,enableIR,enableMA,enableMD,enablePC,selectmuxA,selectmuxMA;

input [2:0] ALUc;

input [1:0] selectmuxPC;

input RWn;

input [159:0]One\_D\_array;

output[11:0] A;

output[15:0] y;

output[15:0] IR;

output[11:0] MA;

output[15:0] MD;

output[11:0] PC;

output[15:0] WD;

output c;

reg [11:0] A;

reg [15:0] IR;

reg [11:0] PC;

reg [15:0] y; //Accumulator

reg [11:0] MA;

reg [15:0] MD;

reg [15:0] WD;

wire c; //Corresponding to carry in state 15 //make a wire for synthesis

wire enableAC,enableIR,enableMA,enableMD,enablePC,selectmuxA,selectmuxMA;

wire [1:0] selectmuxPC;

wire [2:0] ALUc;

wire [15:0] z; //ALU store output

//issue?

always @(posedge clk) begin

if (rstn == 0) begin

A = 0;

y = 0; //y is ac

//c = 0; //comment out for synthesis

PC = 0;

MD = 0;

MA = 0;

IR = 0;

end

end

alu ALU(y,MD,ALUc,z,c,c);

always @ (posedge clk) begin

if (enableAC == 1) begin

y = z; //Load y with ALU store output

end

WD = y;

end

always @ (posedge clk) begin

case(RWn)

0 : begin //direct////////////////////////////////////////////////////////

if (enableAC == 1)

WD = y; //store output of alu into wd

// if (enableAC == 1 && enableMA == 1 && enablePC == 1) begin

// One\_D\_array[MA\*16+:16] = y;

// end //For store, but was not able to get this working

end

1 : begin //////////////////////////////////////////////////////////////

if (enableMD == 1)

MD = One\_D\_array[MA\*16+:16];

else if (enableIR == 1)

IR = One\_D\_array[PC\*16+:16];

end

endcase

case(selectmuxMA)

0 : begin

if (enableMA == 1 && enableIR == 1)

MA <= IR[11:0];

end

1 : begin

if (enableMA == 1 && enableMD == 1)

MA <= MD[11:0];

end

endcase

case(selectmuxPC)

0 : begin

if (enablePC == 1)

PC <= PC+1;

end

1 : begin

if (enablePC == 1)

PC <= IR[11:0];

end

2 : begin

if (enablePC == 1)

PC <= MD[11:0];

end

endcase

end

always @ (posedge clk) begin

case(selectmuxA)

0 : begin

if (enablePC == 1)

A = PC;

end

1 : begin

if (enableMA == 1)

A = MA;

end

endcase

end

endmodule

**Appendix C -ALU**

`timescale 1ns / 1ps

module alu(a,b,ALUc,z,cout,cin);

input [15:0] a;

input [15:0] b;

input [2:0] ALUc;

input cin;

output [15:0] z;

output cout;

reg c;

reg [16:0] y; //17 bits for carry (c)

always@(ALUc or b)

begin

case(ALUc)

0 : begin // NOT

y = ~a;

//c = cin;

end

1 : begin // ADC

y = a + b;// + cin;

//c = y[16];

end

3 : begin // INCREMENT ACC

y = y + 1;

c = y[16];

end

4 : begin // Store & Clear

y = 0;

//c = cin;

end

5 : begin // Pass b

y = b;

// c = cin;

end

endcase

end

assign z = y;

assign cout = c;

endmodule

**Appendix D - Controller**

`timescale 1ns / 1ps

module Controller(IR,clk,rstn,y,enableMD,enableAC,enablePC,enableIR,enableMA,ALUc,RWn,selectmuxMA,selectmuxPC,selectmuxA,PresentState);

input [3:0] IR; //instruction

input clk;

input rstn;

input [15:0] y; //Accumulator

output enableAC,enableIR,enableMA,enableMD,enablePC,selectmuxA,selectmuxMA,RWn;

output [1:0] selectmuxPC;

output [2:0] ALUc;

output [4:0] PresentState;

reg c,enableAC,enableIR,enableMA,enableMD,enablePC,selectmuxA,selectmuxMA;

reg [1:0] selectmuxPC;

reg RWn;

reg [4:0] PresentState, NextState;

parameter S0 = 5'd0, S1 = 5'd1, S2 = 5'd2, S3 = 5'd3,

S4 = 5'd4, S5 = 5'd5, S6 = 5'd6, S7 = 5'd7,

S8 = 5'd8, S9 = 5'd9, S10 = 5'd10,

S11 = 5'd11, S12 = 5'd12, S13 = 5'd13,

S14 = 5'd14, S15 = 5'd15, S16 = 5'd16;

assign ALUc = IR[3:1];

always @ (posedge clk or negedge rstn) begin

if (rstn == 1'b0)

PresentState = S0;

else

PresentState = NextState;

end

always @(PresentState) begin

enableAC = 0;

enableMD = 0;

enablePC = 0;

enableIR = 0;

enableMA = 0;

selectmuxMA = 0;

selectmuxPC = 2'b00;

selectmuxA = 0; //Reset enables for every state

RWn = 0;

case(PresentState)

S0: begin

RWn = 1;

enablePC = 1;

enableIR = 1;

selectmuxPC = 2'b11;

NextState = S1;

end

S1: begin

enablePC = 1;

if (IR[3:1] == 3'b000) begin //not

NextState = S2;

end

else if (IR[3:1] == 3'b011) begin //inca

NextState = S3;

end

else if (IR[3:1] == 3'b010) begin //jpa

if (y > 0) begin

if (IR[0] == 1'b1)

NextState = S4;

else

NextState = S7;

end

else begin

NextState = S0;

end

end

else begin

NextState = S8;

end

end

S2: begin

enableAC = 1;

enablePC = 1;

selectmuxPC = 2'b11;

NextState = S0; //AC

end

S3: begin

enableAC = 1;

enablePC = 1;

selectmuxPC = 2'b11;

NextState = S0; //AC

end

S4: begin

enableIR = 1;

enableMA = 1;

selectmuxA = 1'b1;

NextState = S5;

end

S5: begin

enableMA = 1;

enableMD = 1;

selectmuxA = 1'b1;

RWn = 1'b1;

NextState = S6;

end

S6: begin

enableAC = 1;

enableMD = 1;

enablePC = 1;

selectmuxPC = 2'b10;

NextState = S0;

end

S7: begin

enableAC = 1;

enableIR = 1;

enablePC = 1;

selectmuxPC = 2'b01;

NextState = S0;

end

S8: begin

enableIR = 1;

enableMA = 1;

if (IR[3:1] == 3'b100 && IR[0]==1)

NextState = S9; //STA & AM

else if (IR[3:1] == 3'b100 && IR[0]==0)

NextState = S11; //STA & ~AM

else

NextState = S12;

end

S9: begin

enableMA = 1;

enableMD = 1;

selectmuxA = 1'b1;

RWn = 1'b1;

NextState = S10;

end

S10: begin

enableMA = 1;

enableMD = 1;

selectmuxMA = 1'b1;

NextState = S11;

end

S11: begin

enableAC = 1;

enableMA = 1;

enablePC = 1;

selectmuxPC = 2'b11;

NextState = S0;

end

S12: begin

enableMA = 1;

enableMD = 1;

RWn = 1;

if ((IR[0]==1) && (IR[3:1] != 3'b001))

NextState = S13; //AM

else begin

if (IR[3:1] == 3'b001)

NextState = S15; //ADC

else

NextState = S16;

end

end

S13: begin

enableMA = 1;

enableMD = 1;

selectmuxMA = 1;

NextState = S14;

end

S14: begin

enableMA = 1;

enableMD = 1;

selectmuxA = 1;

RWn = 1;

if (IR[3:1] == 3'b001)

NextState = S15; //ADC

else

NextState = S16;

end

S15: begin

enableAC = 1;

enableMD = 1;

NextState = S0;

end

S16: begin

enableAC = 1;

enableMD = 1;

enablePC = 1;

selectmuxPC = 2'b11;

NextState = S0;

end

endcase

end

endmodule

**Appendix E - Testbench**

`timescale 1ns / 1ps

module testbench;

reg [15:0] RD;

reg clk;

reg rstn;

wire [15:0] IR;

wire [11:0] PC;

wire [11:0] MA;

wire [15:0] MD;

wire [15:0] WD;

wire [11:0] A;

wire [15:0] AC;

wire enableAC;

wire enableIR;

wire enableMA;

wire enableMD;

wire enablePC;

wire [2:0] ALUc;

wire c;

wire selectmuxA;

wire selectmuxMA;

wire [1:0] selectmuxPC;

wire [4:0] PresentState;

reg [15:0] M[9:0];

wire [159:0]One\_D\_array;

// Instantiate the Unit Under Test (UUT)

processor uut (

.RD(RD),

.clk(clk),

.rstn(rstn),

.WD(WD),

.A(A),

.IR(IR),

.MD(MD),

.y(AC),

.PC(PC),

.MA(MA),

.PresentState(PresentState),

.enableIR(enableIR),

.enableMD(enableMD),

.enableAC(enableAC),

.enablePC(enablePC),

.enableMA(enableMA),

.selectmuxMA(selectmuxMA),

.selectmuxPC(selectmuxPC),

.selectmuxA(selectmuxA),

.ALUc(ALUc),

.c(c),

.One\_D\_array(One\_D\_array)

);

initial begin

clk = 0;

rstn = 0;

//Memory to set every every op code and adressing

M[0] <= 16'b1010000000000101;

M[1] <= 16'b0110000000000000;

M[2] <= 16'b0100000000000000;

M[3] <= 16'b0010000000000000;

M[4] <= 16'b0000000000000000;

M[5] <= 16'b1000000000000000;

M[6] <= 16'b0101000000000001;

M[7] <= 16'b1011000000000000;

M[8] <= 16'b0011000000000000;

M[9] <= 16'b1001000000000101;

#100;

rstn = 1;

RD <= M[A];

end

genvar i;

for (i=0; i<10; i=i+1) begin

assign One\_D\_array[16\*i+15:16\*i] = M[i];

end

always @(clk) begin

#10

clk <= ~clk;

end

always @ (posedge clk or negedge rstn or A) begin

RD <= M[A];

end

always @ (WD) begin

if (WD == 0 && enableAC == 1)

M[A] <= WD;

end

endmodule