**Objective**

The objective for this project is to design a microcontroller than can do basic math operations, move, store, and load data to/from registers and other input output registers

**Design methodology: ALU**

My ALU takes in the bus and 4 control signals (A\_in, B\_in, reset, and out\_EN)

If reset is ‘1’ then The value of A and B is set to 16'b0000000000000000.

If reset isn’t one A and B take turns latching their values from the bus.

Next, a select signal determines which operation gets done.

Once done the ALU waits for “out\_EN” to equal ‘1’.

Once it does the result of the operation is driven to the bus.

**Design methodology: Instruction Decoder**

My ID holds all the FSMs, and controls all the other units.

While reset = ‘1’ it sets all the control signals to 0.

Once reset is off, it assigns the current state the value it should be after the previous state.

It drives Opcode, parameterA, and parameterB the corresponding bits from the instruction register.

**Instruction fetch:**

When the Instruction control is set to 1 (which it is off reset) this fsm begins

State 0: signals the program counter to drive the bus and enables the memory address register

State 1: undoes what it did in S1, enables memory, sets read/~write to 1, waits for memory to finish

State 2: disables memory, signals mdr that data is going from memory to bus, allows mdr to output to bus, Enables the instruction register to input from the bus, and increments the program counter

State 3: Disables the 4 things it enabled in the last state.

State 4: IF disables itself and checks the OPcode for what FSM it should turn on.

End

**Register ALU FSM:**

State 0: Checks parameterA (bits 11-6) and enables the corresponding register to drive the bus. Also enables the ALU to read from the bus for the value of A

State 1: disables what it enabled in the last state.

State 2 & 3: samething as 1&2 but for parameter

State 4: tells the ALU which operation to preform

State 5: Enables the receiving register to read from the bus

State 6: undoes what it did in S5

State 7:Disables itself and enables the instruction register.

**Immediate ALU FSM:**

State 0: Checks parameterA (bits 11-6) and enables the corresponding register to drive the bus. Also enables the ALU to read from the bus for the value of A

State 1: disables what it enabled in the last state.

State 2: Enables the ID to drive the bus, sends parameterB (last 6 bits) to the bus, and enables ALU to read from the bus the value for its second operant.

State 3: undoes what it sis in S2

State 4: tells the ALU which operation to do.

State 5: enables the register that is going to store the data to read from bus.

State 6:disables the register that read from the bus.

State 7: Disables itself and enables the instruction register.

**Load FSM:**

State 0: checks which register has the necessary address, and enables it to drive the bus, enables the memory to read the address, enables MAR to push the address to memory

State 1: disables MAR driving to memory, disables the register that was holding the address, and enables memory. Turns off read/write.

State 2: waits for memory to be done.

State 3: signals memory data register that data is coming from memory and going to the bus. Enables MDR to drive the bus. Enables the corresponding register to read from the bus.

State 4: Disables what it did in S3.

State 5: Disables itself and enables the instruction register.

**Store FSM:**

State 0: Enables the register holding the data to drive the bus, tells the MDR that data is coming from the bus and going to memory.

State 1: Undoes what it did in S0, enables memory

State 2: waits for memory to finish, and disables memory

State 3: sends the address for the stored data to the specified register.

State 4: undoes what it did in S3

State 5: Disables itself and enables the instruction register.

**Move FSM:**

State 0: Enables the corresponding register to drive it’s data to bus, also signals the receiving register to read from the bus.

State 1: Undoes what it did in S0.

State 3: Disables itself and enables the instruction register.

**Move Immidate FSM:**

State 0: Enables the corresponding register to read from the bus, enables the ID to drive the bus, drives parameterB (bits 5 to 0) to the bus.

State 1: undoes what it did in S0.

State 2: Disables itself and enables the instruction register.

**Design methodology: Instruction Register**

If reset = ‘1’ drives -> disconnects from Instruction decoder.

If Instruction register gets enabled it drives the data from the bus to the decoder.

Once it gets disabled it stops reading from bus but keeps driving the same instruction until it gets a new one.

**Design methodology: I/O port 0**

If reset = ‘1’ it disconnects from bus

If it gets enabled it latches from the bus.

If IO\_OUT 0 gets set to ‘1’ it drives the bus, if it is ‘0’ it disconnects from the bus.

**Design methodology: I/O port 1**

If reset = ‘1’ it disconnects from bus and sets it’s external output to 16’b00000…

Else it outputs the bus outside the system

If IO\_OUT0 gets enabled it drives the bus, else its disconnected

**Design methodology: Memory address register**

If reset is ‘1’ it disconnects from memory.

When it gets enabled it drives whatever is on the bus to the memory unit.

**Design methodology: Memory data register**

If reset = ‘1’ it sets its data to 0 and disconnects from memory

If the from bus to memory signal gets enabled it drives the bus to the memory unit

If the from memory to bus signal gets enabled it loads the data from memory into itself.

Once it’s output flag gets enabled it drives it’s data to the bus.

If it’s flag is disabled it is disconnected from the bus.

**Design methodology: Program counter**

if reset is ‘1’ it sets the instruction to ‘0’

if the PC\_OUT gets enabled it drives the current instruction to the bus.

At positive edge of increment it increments the instruction by 1.

**Design methodology: registers**

\*All 4 registers are the same.

If reset is ‘1’ the data in register is 16’b0000000000000000

If it’s enable flag gets set to ‘1’ it reads data from the bus.

If it’s output flag gets enabled it drives its data to the bus

If it’s output flag is disabled it disconnects from the bus.

**Design methodology: memory**

If reset = ‘1’ it disconnects from memory data register.

When it gets enabled it checks to see if it should read or write.

If RW = ‘1’ it reads the data from the memory address register and outputs the corresponding data to the memory data register.

If RW = ‘0’ It reads the data from the memory data register.

Once it’s done it sends a flag letting the decoder know.

**Design methodology: Top level design**

top level design links all the lower designs together.

For modules that output to and input from bus it sends “busOUT” to them twice, the modules don’t know that it is the same wire, this allows them to input and output to and from the bus.

Clk cycle is 20 nano seconds

Reset is enabled for 100 nano seconds, then disabled forever.

**Instruction set with machine code assignments**

MOVI R0, #3A 1010 000000 111010

SUBI R0, #9 0100 000000 001001

MOVI P0, #17 1010 000100 010111

AND R0, R2 0110 000000 000010

XOR P0, R0 1000 000100 000000

NOT R0 0101 000000 000000

STORE P0, (R0) 1100 000100 000000

LOAD (R0), R1 1011 000000 000001

**Other assumptions made**

All registers are set to 16’b0000000000000000 at reset

The stored register is 16’b1111111111111111.

**Synthesis process**

I could only synthesis my registers (R0-R3), but no matter what constraints I put on them they wouldn’t show delay. So I decided to not Synthesize my project.

**Simulation method – translate the program into machine code**

I simulated my top level design, which called the lower level designs.

**A picture containing chart

Description automatically generatedResult (simulation waveform screenshots) – need every transaction on bus with clear explanation; zoom in to show delay**

0ns: reset is 1, everything else is set to 0.

100 ns. Reset -> 0, this triggers instruction fetch FSM to start. IF enables Program counter to output to bus. IF enables memory address register to read from bus.

Bus is 16’b0000000000000000

200ns PC\_out and MAR\_EN get disabled. Memory gets enabled, and outputs the first instruction to memory data register

220ns memory function is complete.

300ns MDR gets alerted that data is going from memory to the bus, MDR gets enabled to drive the bus. Instruction register gets enabled to read from the bus. Program counter gets incremented.

400ns IF disables itself and enables MOVI

500ns MOVI FSM drives its data (hexadecimal: #3A). R0\_EN enables register 0 to read from bus.

R0 now holds the value 3A (58)

700 ns. MOVI FSM disables itself and enables instruction fetch FSM to start. IF enables Program counter to drive the bus. IF enables memory address register to read from bus.

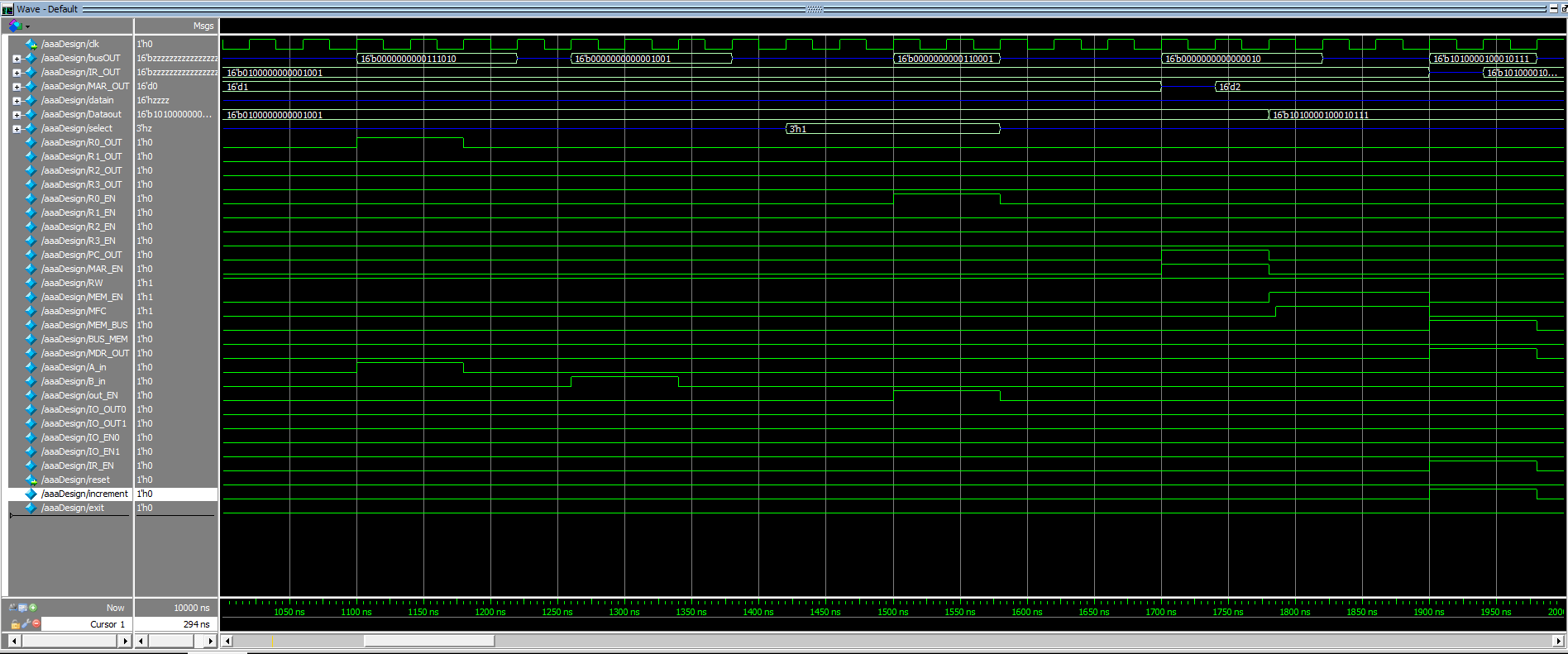
Bus is 16’b0000000000000001

755ns PC\_out and MAR\_EN get disabled. Memory gets enabled, and outputs the second instruction to memory data register

775ns memory function is complete.

900ns MDR gets alerted that data is going from memory to the bus, MDR gets enabled to drive the bus. Instruction register gets enabled to read from the bus. Program counter gets incremented.

975ns IF disables the remaining signals it had enabled, then gives up control to I\_ALU FSM.



1100ns I\_ALU enables R0 to drive the bus and enables the ALU to read it’s A value from the bus.

1180ns I\_ALU disables R0 from driving and disables ALU from reading.

1260ns I\_ALU enables decoder to drive the bus, it drives the value “00000000001001” (9) to bus, it enables ALU to read it’s B value from the bus.

1340ns I\_ALU disables decodser from driving and disables ALU from reading.

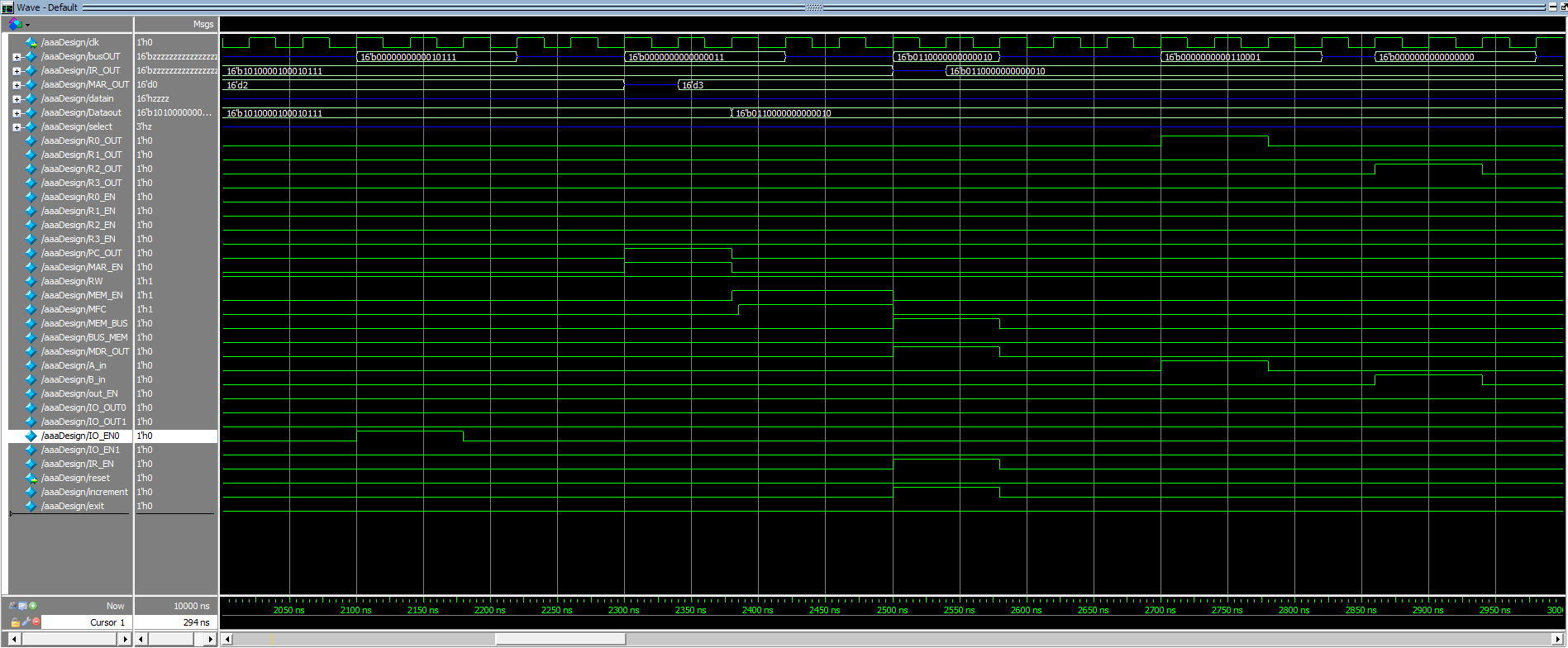
1340 – 1500ns ALU preforms operation

1500ns I\_ALU enables ALU to drive the bus, and enables R0 to read from the bus.

The value in R0 is “0000000000110001” (49)

1580ns I\_ALU disables ALU from driving and disables R0 from reading, disables itself, enables instruction fetch.

1700 – 1980ns IF fetches instruction 3



2100ns MOVI FSM receives control from IF, enables decoder to drive the last 6 bits of the instruction to the bus, and enables P0 to read from the bus.

2180ns MOVI disables the decoder from driving the bus, and disables P0 from reading from the bus. Gives control to Instruction fetch.

The value in P0 is “0000000000010111”

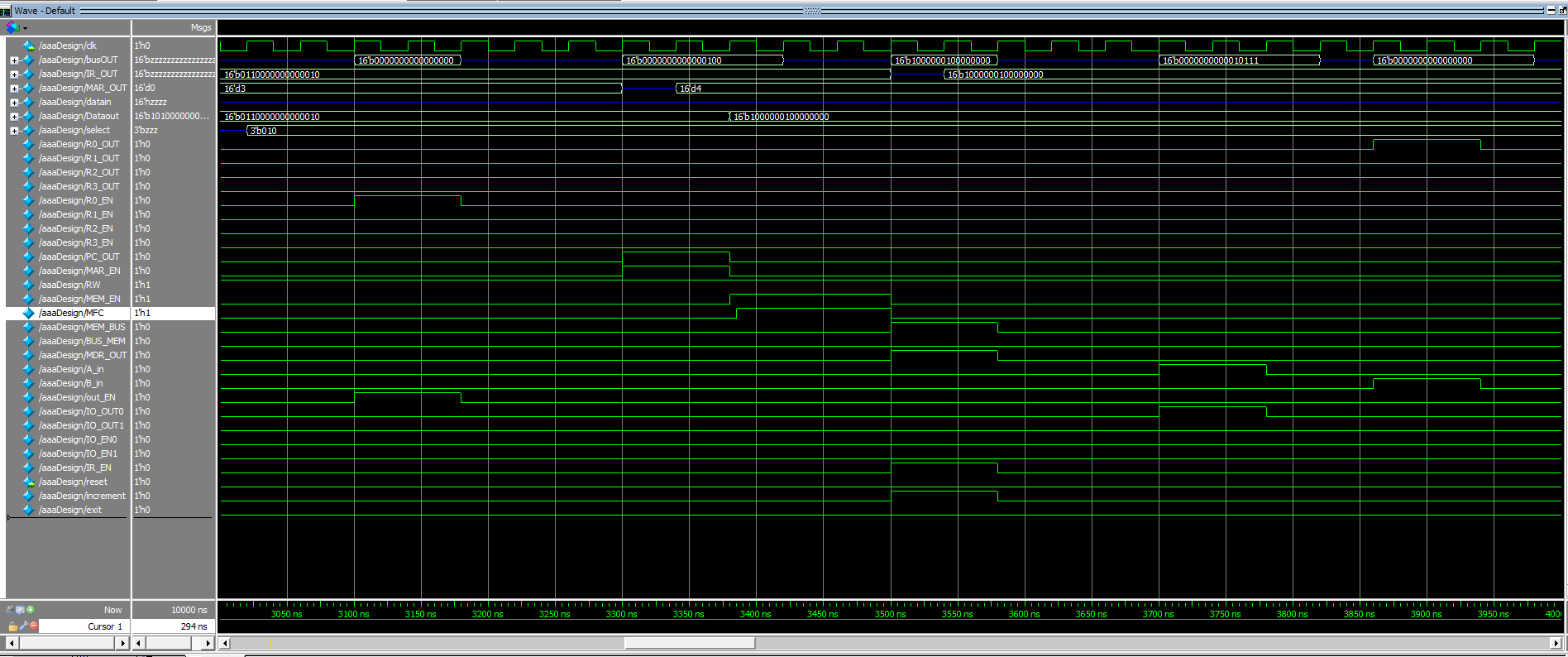
2300-2580ns IF fetches instruction 4

2700ns R\_ALU receives control from IF; Enables R0 to drive the bus, Enables ALU to read from the bus for its 1st value (A).

2780ns R\_ALU disables R0 from driving the bus and disables ALU from reading from the bus.

2860ns R\_ALU Enables R2 to drive the bus, Enables ALU to read from the bus for its 2st value (B).

2940ns R\_ALU disables R2 from driving the bus and disables ALU from reading from the bus.



2940-3100ns ALU preforms operation

3100ns R\_ALU enables ALU to drive it’s result on the bus, enables R0 to read the result from the bus.

3180ns R\_ALU disables ALU from driving the bus and disables R0 from reading from the bus.

The value inside R0 is “0000000000000000”

R\_ALU gives up control to instruction fetch

3300-3580ns IF fetches instruction 5

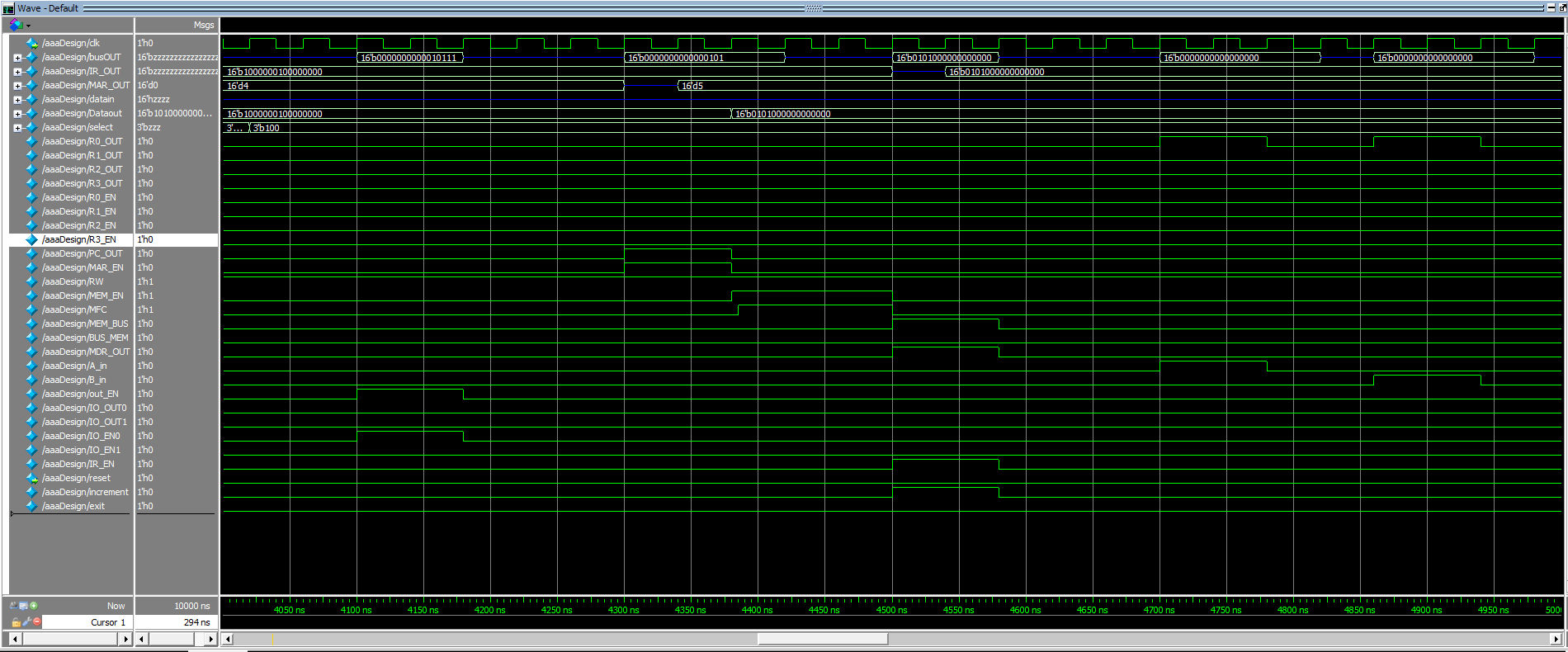
3700ns R\_ALU receives control from IF; Enables P0 to drive the bus, Enables ALU to read from the bus for its 1st value (A).

3780ns R\_ALU disables P0 from driving the bus and disables ALU from reading from the bus.

3860ns R\_ALU Enables R0 to drive the bus, Enables ALU to read from the bus for its 2st value (B).

3940ns R\_ALU disables R0 from driving the bus and disables ALU from reading from the bus.

3940-4100ns ALU preforms operation



4100ns R\_ALU enables ALU to drive the bus and IO\_0 to read from the bus.

4180ns R\_ALU diables ALU from driving the bus and disables IO\_0 from reading from the bus, gives up control to IF.

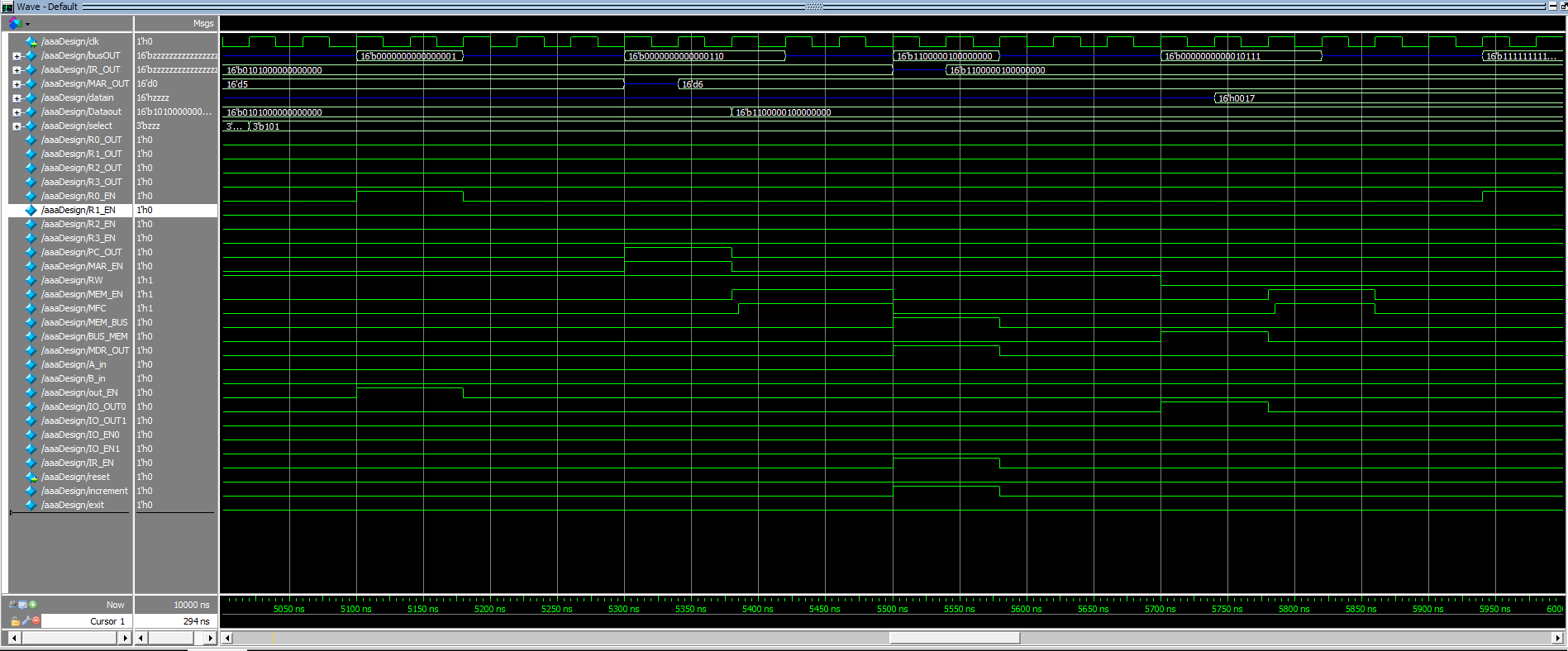
Value in IO\_0 is “0000000000010111”

4300-4580ns IF fetches instruction 6

4700ns R\_ALU receives control from IF; Enables R0 to drive the bus, Enables ALU to read from the bus for its 1st value (A).

4780ns R\_ALU disables R0 from driving the bus and disables ALU from reading from the bus.

4860-4940ns This process repeats for B, but the ALU doesn’t do anything with it because it only needs A.

4940-5100ns ALU preforms operation

5100ns R\_ALU enables ALU to drive the bus and R0 to read from the bus.

5180ns R\_ALU diables ALU from driving the bus and disables R0 from reading from the bus, gives up control to IF.

Value in R0 is “0000000000000001”

5300-5580ns IF fetches instruction 6

5580ns IF gives up control to STORE FSM

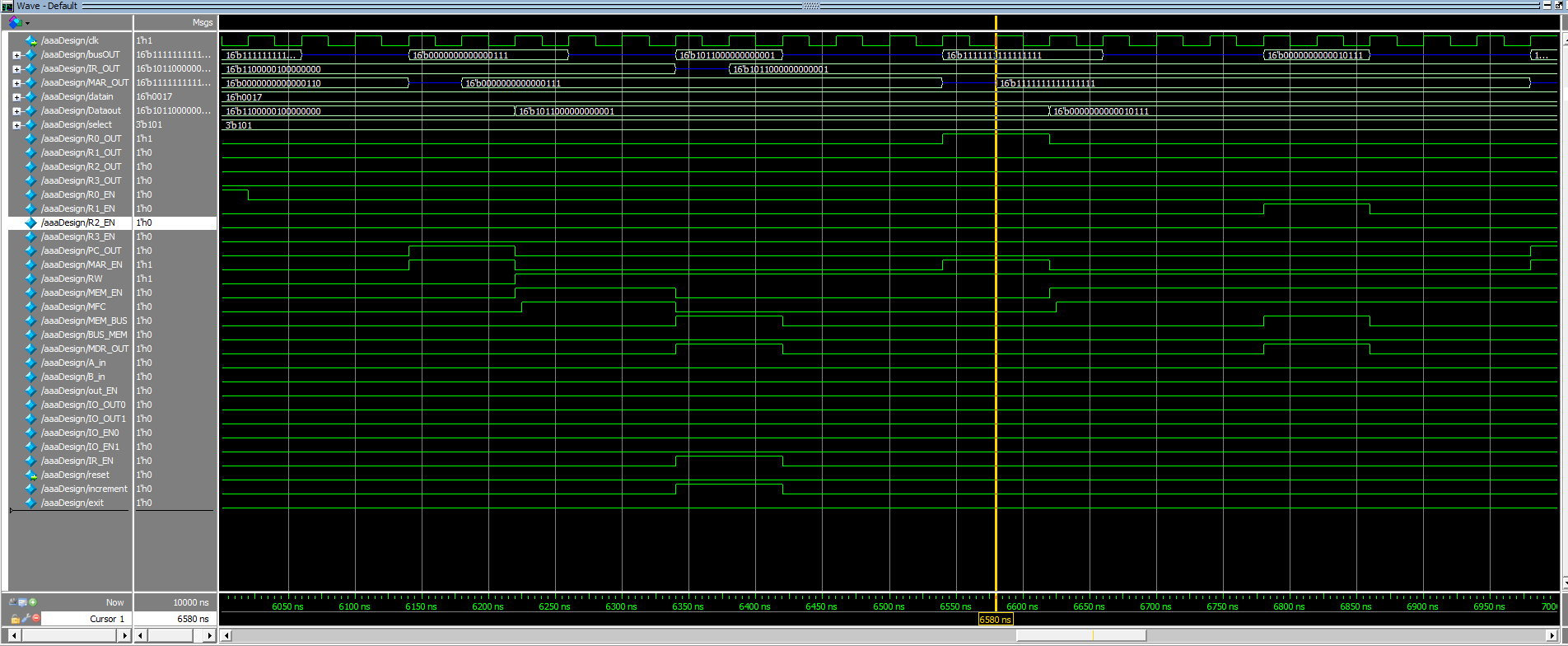
5700ns STORE sets R/W to 0, signals MDR that data is moving from bus to memory, and enables P0 to drive the bus.

5780ns bus to memory flag gets disabled, P0 gets disabled from driving the bus. Memory gets enabled.

5785ns memory function is complete.

5860ns memory is disabled

5940ns STORE outputs “1111111111111111” to bus (address for memorycell).

 STORE enables R0 to read from bus.

6020ns STORE disables R0 from reading from bus, gives up control to Instruction fetch.

The value in R0 is “1111111111111111”

6140-6420ns IF fetches instruction 7

6540ns IF gives up control to LOAD, LOAD enables R0 to drive the bus, LOAD enables memory address register to read from the bus.

6620ns LOAD disables R0 from driving the bus, LOAD disables memory address register from reading from the bus. LOAD enables memory.

6625ns Memory function is complete.

6780ns LOAD signals MDR that data is coming from the memory to the bus, LOAD enables MDR to output to bus, LOAD enables R1 to read from the bus

6860ns LOAD disables what it previously enabled, gives up control to IF.

R1 has a value of “0000000000010111”

6980ns IF begins fetching another instruction, but it doesn’t exist.

**Conclusion**

This project went over pretty good, I wish other classes weren’t a factor so I could’ve figured out the synthesis part. Program executes all instructions as expected.

**Source Code**

module aaaDesign(reset,clk);

wire[15:0]IR\_OUT,busOUT,MAR\_OUT,datain,Dataout;

wire[2:0] select;

wire R0\_OUT,R1\_OUT,R2\_OUT,R3\_OUT,R0\_EN, R1\_EN, R2\_EN, R3\_EN,PC\_OUT,MAR\_EN,RW,MEM\_EN,MFC,MEM\_BUS,BUS\_MEM,MDR\_OUT,A\_in,B\_in,out\_EN,IO\_OUT0,IO\_OUT1,IO\_EN0,IO\_EN1,IR\_EN;

output reg clk, reset;

ID topID(IR\_OUT,reset,clk,MFC,R0\_OUT,R1\_OUT,R2\_OUT,R3\_OUT,R0\_EN, R1\_EN, R2\_EN, R3\_EN,PC\_OUT,MAR\_EN,RW,MEM\_EN,MEM\_BUS,BUS\_MEM,MDR\_OUT,A\_in,B\_in,out\_EN,IO\_OUT0,IO\_OUT1,IO\_EN0,IO\_EN1,IR\_EN,increment,select,busOUT);

ALU topALU(out\_EN,reset,A\_in,B\_in,busOUT,select,busOUT);

MAR topMAR(busOUT,MAR\_EN,reset,clk,MAR\_OUT);

MDR topMDR(busOUT,Dataout,MEM\_BUS,BUS\_MEM, reset,MDR\_OUT,clk,datain,busOUT);

PC topPC(reset,PC\_OUT,clk,increment,busOUT);

IR topIR(busOUT,IR\_EN,reset,clk,IR\_OUT);

R0 topR0(busOUT,R0\_EN,reset,R0\_OUT,clk,busOUT);

R1 topR1(busOUT,R1\_EN,reset,R1\_OUT,clk,busOUT);

R2 topR2(busOUT,R2\_EN,reset,R2\_OUT,clk,busOUT);

R3 topR3(busOUT,R3\_EN,reset,R3\_OUT,clk,busOUT);

IO\_0 topIO\_0(busOUT,IO\_EN0,reset,IO\_OUT0,clk,busOUT);

IO\_1 topIO\_1(busOUT,IO\_EN1,reset,IO\_OUT1,busOUT,exit);

mainmemory topMEM(Dataout, MFC, MEM\_EN, MAR\_OUT,datain, RW,reset);

initial begin

reset = 1; clk = 0;

#100;

reset = 0;

end

always

#20 clk = ~clk;

Endmodule

module R0(

input [15:0] busIN,

input R0\_EN,reset,R0\_OUT,clk,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge R0\_EN or posedge reset or posedge clk)

begin

if(reset == 1'b1)

Q = 16'b0000000000000000;

else if (R0\_EN == 1)

Q = busIN;

end

always @ (posedge clk or posedge R0\_OUT)

if (R0\_OUT == 1)begin

busOUT = Q;

end

else if (R0\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module R1(

input [15:0] busIN,

input R1\_EN,reset,R1\_OUT,clk,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge R1\_EN or posedge reset or posedge clk)

begin

if(reset == 1'b1)

Q <= 16'b0000000000000000;

else if (R1\_EN)

Q <= busIN;

end

always @ (posedge clk or posedge R1\_OUT)

if (R1\_OUT == 1)begin

busOUT = Q;

end

else if (R1\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module R2(

input [15:0] busIN,

input R2\_EN,reset,R2\_OUT,clk,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge R2\_EN or posedge reset or posedge clk)

begin

if(reset == 1'b1)

Q <= 16'b0000000000000000;

else if (R2\_EN)

Q <= busIN;

end

always @ (posedge clk or posedge R2\_OUT)

if (R2\_OUT == 1)begin

busOUT = Q;

end

else if (R2\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module R3(

input [15:0] busIN,

input R3\_EN,reset,R3\_OUT,clk,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge R3\_EN or posedge reset or posedge clk)

begin

if(reset == 1'b1)

Q <= 16'b0000000000000000;

else if (R3\_EN)

Q <= busIN;

end

always @ (posedge clk or posedge R3\_OUT)

if (R3\_OUT == 1)begin

busOUT = Q;

end

else if (R3\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module PC(

input reset,PC\_OUT,clk,increment,

output reg [15:0] busOUT

);

reg [15:0] instruction = 16'bzzzzzzzzzzzzzzzz;

always @(posedge reset)

begin

if (reset == 1)begin

instruction = 16'b0000000000000000;

end

end

always @(posedge increment)begin

instruction = instruction + 1'b1;

end

always @ (posedge PC\_OUT or posedge clk)

if (PC\_OUT == 1)begin

busOUT = instruction;

end

else if (PC\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module mainmemory(Dataout, MFC, MEM\_EN, MAR\_OUT,

datain, RW,reset);

input MEM\_EN, RW,reset;

input[15:0] MAR\_OUT, datain;

output [15:0] Dataout;

output MFC;

reg [15:0] Dataout, memorycell;

reg MFC;

always@(posedge reset)

begin

Dataout = 16'bZZZZZZZZZZZZZZZZ;

end

always@(posedge MEM\_EN)

begin

if(RW==1) begin

case(MAR\_OUT)

16'b0000000000000000: Dataout = 16'b1010000000111010;

16'b0000000000000001: Dataout = 16'b0100000000001001;

16'b0000000000000010: Dataout = 16'b1010000100010111;

16'b0000000000000011: Dataout = 16'b0110000000000010;

16'b0000000000000100: Dataout = 16'b1000000100000000;

16'b0000000000000101: Dataout = 16'b0101000000000000;

16'b0000000000000110: Dataout = 16'b1100000100000000;

16'b0000000000000111: Dataout = 16'b1011000000000001;

16'b1111111111111111: Dataout = memorycell;

default: Dataout = 16'bzzzzzzzzzzzzzzzz;

endcase

#5 MFC = 1;

end

else

begin

memorycell = datain;

#5 MFC = 1;

end

end

always@(negedge MEM\_EN)

MFC = 0;

Endmodule

module MDR(

input [15:0] busIN, Dataout,

input MEM\_BUS,BUS\_MEM, reset,MDR\_OUT,clk,

output reg [15:0] datain,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge BUS\_MEM or posedge reset or posedge clk)

begin

if(reset == 1'b1)

datain <= 16'bzzzzzzzzzzzzzzzz;

else if(BUS\_MEM == 1)

datain <= busIN;

end

always @(posedge MEM\_BUS or posedge reset)

begin

if(reset == 1'b1)

Q <= 16'b0000000000000000;

else

Q <= Dataout;

end

always @ (\*)

if (MDR\_OUT == 1) begin

busOUT = Q;

end

else if (MDR\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module MAR(

input [15:0] busOUT,

input MAR\_EN,reset,clk,

output reg [15:0] MAR\_OUT

);

always @(posedge reset)

begin

if(reset == 1'b1)

MAR\_OUT <= 16'bzzzzzzzzzzzzzzzz;

end

always @(posedge MAR\_EN or posedge clk)

begin

if(MAR\_EN == 1)

MAR\_OUT <= busOUT;

end

endmodule

module IO\_1(

input [15:0] busIN,

input IO\_EN1,reset,IO\_OUT1,

output reg [15:0] busOUT,

output reg [15:0] exit

);

reg [15:0] Q;

always @(posedge IO\_EN1 or posedge reset)

begin

if(reset == 1'b1)begin

Q <= 16'b0000000000000000;

exit <= 16'b0000000000000000;

end

else

begin

Q <= busIN;

exit <= busIN;

end

end

always @ (\*)

if (IO\_OUT1 == 1)begin

busOUT = Q;

end

else if (IO\_OUT1 == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module IO\_0(

input [15:0] busIN,

input IO\_EN0,reset,IO\_OUT0,clk,

output reg [15:0] busOUT

);

reg [15:0] Q;

always @(posedge IO\_EN0 or posedge reset or posedge clk)

begin

if(reset == 1'b1)

Q <= 16'bzzzzzzzzzzzzzzzz;

else if(IO\_EN0 == 1)

Q <= busIN;

end

always @ (posedge IO\_OUT0 or posedge clk)

if (IO\_OUT0 == 1)begin

busOUT <= Q;

end

else if (IO\_OUT0 == 0) begin

busOUT <= 16'bZZZZZZZZZZZZZZZZ;

end

endmodule

module IR(

input [15:0] busOUT,

input IR\_EN,reset,clk,

output reg [15:0] IR\_OUT

);

always @(posedge reset)

begin

if(reset == 1'b1)

IR\_OUT <= 16'bzzzzzzzzzzzzzzzz;

end

always @(posedge IR\_EN or posedge clk)

begin

if (IR\_EN == 1)

IR\_OUT <= busOUT;

end

endmodule

module ID(

input [15:0] IR\_OUT,

input reset,clk,MFC,

output reg R0\_OUT,R1\_OUT,R2\_OUT,R3\_OUT,R0\_EN, R1\_EN, R2\_EN, R3\_EN,PC\_OUT,MAR\_EN,RW,MEM\_EN,MEM\_BUS,BUS\_MEM,MDR\_OUT,A\_in,B\_in,out\_EN,IO\_OUT0,IO\_OUT1,IO\_EN0,IO\_EN1,IR\_EN,increment,

output reg [2:0] select,

output reg [15:0] busOUT

);

reg [3:0] OPcode;

reg [5:0] parameterA, parameterB;

reg IF\_EN,R\_ALU\_EN,I\_ALU\_EN,LOAD\_EN,STORE\_EN,MOV\_EN,MOVI\_EN,ID\_OUT;

reg [2:0] state, nextstate;

reg [15:0] data;

parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 = 3'b100, S5 = 3'b101, S6 = 3'b110, S7 = 3'b111;

always @(posedge clk or posedge reset) begin

if (reset)

begin

OPcode <= 4'b0000;

parameterA <= 6'b000000;

parameterB <= 6'b000000;

state <= S0;

IF\_EN <= 1;

R0\_OUT <= 0;

R1\_OUT <= 0;

R2\_OUT <= 0;

R3\_OUT <= 0;

R0\_EN <= 0;

R1\_EN <= 0;

R2\_EN <= 0;

R3\_EN <= 0;

PC\_OUT <= 0;

MAR\_EN <= 0;

RW <= 0;

MEM\_EN <= 0;

MEM\_BUS <= 0;

BUS\_MEM <= 0;

MDR\_OUT <= 0;

IR\_EN <= 0;

A\_in <= 0;

B\_in <= 0;

out\_EN <= 0;

IO\_OUT0 <= 0;

IO\_EN0 <= 0;

IO\_OUT1 <= 0;

IO\_EN1 <= 0;

IR\_EN <= 0;

increment <= 0;

busOUT <= 16'bZZZZZZZZZZZZZZZZ;

select <= 3'bzzz;

end

else begin

state <= nextstate;

OPcode <= IR\_OUT [15:12];

parameterA <= IR\_OUT [11:6];

parameterB <= IR\_OUT [5:0];

end

end

always @(posedge clk or posedge reset)

if (reset)

state <= S0;

else if (IF\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

PC\_OUT <= 1;

MAR\_EN <= 1;

end

S1: begin

nextstate <= S2;

MAR\_EN <= 0;

PC\_OUT <= 0;

RW <= 1;

MEM\_EN <= 1;

if (MFC == 1)

nextstate <= S2;

else

nextstate <= S1;

end

S2: begin

nextstate <= S3;

MEM\_EN <= 0;

MEM\_BUS <= 1;

MDR\_OUT <= 1;

IR\_EN <= 1;

increment <= 1;

end

S3: begin

nextstate <= S4;

IR\_EN <= 0;

MDR\_OUT <= 0;

MEM\_BUS <= 0;

increment <= 0;

end

S4: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 0;

if(OPcode == 4'b0000)

MOV\_EN <= 1;

else if(OPcode == 4'b0001)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b0010)

I\_ALU\_EN <= 1;

else if(OPcode == 4'b0011)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b0100)

I\_ALU\_EN <= 1;

else if(OPcode == 4'b0101)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b0110)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b0111)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b1000)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b1001)

R\_ALU\_EN <= 1;

else if(OPcode == 4'b1010)

MOVI\_EN <= 1;

else if(OPcode == 4'b1011)

LOAD\_EN <= 1;

else if(OPcode == 4'b1100)

STORE\_EN <= 1;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (R\_ALU\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

if (parameterA == 6'b000000)

R0\_OUT <= 1;

if (parameterA == 6'b000001)

R1\_OUT <= 1;

if (parameterA == 6'b000010)

R2\_OUT <= 1;

if (parameterA == 6'b000011)

R3\_OUT <= 1;

if (parameterA == 6'b000100)

IO\_OUT0 <= 1;

A\_in <= 1;

end

S1: begin

nextstate <= S2;

if (parameterA == 6'b000000)

R0\_OUT <= 0;

if (parameterA == 6'b000001)

R1\_OUT <= 0;

if (parameterA == 6'b000010)

R2\_OUT <= 0;

if (parameterA == 6'b000011)

R3\_OUT <= 0;

if (parameterA == 6'b000100)

IO\_OUT0 <= 0;

A\_in <= 0;

end

S2: begin

nextstate <= S3;

if (parameterB == 6'b000000)

R0\_OUT <= 1;

if (parameterB == 6'b000001)

R1\_OUT <= 1;

if (parameterB == 6'b000010)

R2\_OUT <= 1;

if (parameterB == 6'b000011)

R3\_OUT <= 1;

if (parameterB == 6'b000100)

IO\_OUT0 <= 1;

B\_in <= 1;

end

S3: begin

nextstate <= S4;

if (parameterB == 6'b000000)

R0\_OUT <= 0;

if (parameterB == 6'b000001)

R1\_OUT <= 0;

if (parameterB == 6'b000010)

R2\_OUT <= 0;

if (parameterB == 6'b000011)

R3\_OUT <= 0;

if (parameterB == 6'b000100)

IO\_OUT0 <= 0;

B\_in <= 0;

end

S4: begin

nextstate <= S5;

if (OPcode == 4'b0001)

select <= 3'b000;

if (OPcode == 4'b0011)

select <= 3'b001;

if (OPcode == 4'b0110)

select <= 3'b010;

if (OPcode == 4'b0111)

select <= 3'b011;

if (OPcode == 4'b1000)

select <= 3'b100;

if (OPcode == 4'b1001)

select <= 3'b110;

if (OPcode == 4'b0101)

select <= 3'b101;

end

S5: begin

nextstate <= S6;

out\_EN <= 1;

if (parameterA == 6'b000000)

R0\_EN <= 1;

if (parameterA == 6'b000001)

R1\_EN <= 1;

if (parameterA == 6'b000010)

R2\_EN <= 1;

if (parameterA == 6'b000011)

R3\_EN <= 1;

if (parameterA == 6'b000100)

IO\_EN0 <= 1;

end

S6: begin

nextstate <= S7;

out\_EN <= 0;

if (parameterA == 6'b000000)

R0\_EN <= 0;

if (parameterA == 6'b000001)

R1\_EN <= 0;

if (parameterA == 6'b000010)

R2\_EN <= 0;

if (parameterA == 6'b000011)

R3\_EN <= 0;

if (parameterA == 6'b000100)

IO\_EN0 <= 0;

end

S7: begin

state <= S0;

nextstate <= S0;

IF\_EN <= 1;

R\_ALU\_EN <= 0;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (I\_ALU\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

if (parameterA == 6'b000000)

R0\_OUT <= 1;

if (parameterA == 6'b000001)

R1\_OUT <= 1;

if (parameterA == 6'b000010)

R2\_OUT <= 1;

if (parameterA == 6'b000011)

R3\_OUT <= 1;

A\_in <= 1;

end

S1: begin

nextstate <= S2;

if (parameterA == 6'b000000)

R0\_OUT <= 0;

if (parameterA == 6'b000001)

R1\_OUT <= 0;

if (parameterA == 6'b000010)

R2\_OUT <= 0;

if (parameterA == 6'b000011)

R3\_OUT <= 0;

A\_in <= 0;

end

S2: begin

nextstate <= S3;

ID\_OUT <= 1;

data <= parameterB;

B\_in <= 1;

end

S3: begin

nextstate <= S4;

B\_in <= 0;

ID\_OUT <= 0;

end

S4: begin

nextstate <= S5;

if (OPcode == 4'b0010)

select <= 3'b000;

if (OPcode == 4'b0100)

select <= 3'b001;

end

S5: begin

nextstate <= S6;

out\_EN <= 1;

if (parameterA == 6'b000000)

R0\_EN <= 1;

if (parameterA == 6'b000001)

R1\_EN <= 1;

if (parameterA == 6'b000010)

R2\_EN <= 1;

if (parameterA == 6'b000011)

R3\_EN <= 1;

if (parameterA == 6'b000100)

IO\_EN0 <= 1;

end

S6: begin

if (parameterA == 6'b000000)

R0\_EN <= 0;

if (parameterA == 6'b000001)

R1\_EN <= 0;

if (parameterA == 6'b000010)

R2\_EN <= 0;

if (parameterA == 6'b000011)

R3\_EN <= 0;

if (parameterA == 6'b000100)

IO\_EN0 <= 0;

out\_EN <= 0;

select <= 3'bzzz;

nextstate <= S7;

end

S7: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 1;

I\_ALU\_EN <= 0;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (LOAD\_EN)

begin

case(state)

S0: begin

RW <= 1;

MAR\_EN <= 1;

nextstate <= S1;

if (parameterA == 6'b000000)

R0\_OUT <= 1;

if (parameterA == 6'b000001)

R1\_OUT <= 1;

if (parameterA == 6'b000010)

R2\_OUT <= 1;

if (parameterA == 6'b000011)

R3\_OUT <= 1;

end

S1: begin

nextstate <= S2;

MAR\_EN <= 0;

if (parameterA == 6'b000000)

R0\_OUT <= 0;

if (parameterA == 6'b000001)

R1\_OUT <= 0;

if (parameterA == 6'b000010)

R2\_OUT <= 0;

if (parameterA == 6'b000011)

R3\_OUT <= 0;

MEM\_EN <= 1;

if (OPcode == 1011) begin

RW <= 0;

end

end

S2: begin

if (MFC == 1)

nextstate <= S3;

else

nextstate <= S2;

end

S3: begin

nextstate <= S4;

MDR\_OUT <= 1;

MEM\_BUS <= 1;

if (parameterB == 6'b000000)

R0\_EN <= 1;

if (parameterB == 6'b000001)

R1\_EN <= 1;

if (parameterB == 6'b000010)

R2\_EN <= 1;

if (parameterB == 6'b000011)

R3\_EN <= 1;

end

S4: begin

nextstate <= S5;

MDR\_OUT <= 0;

MEM\_BUS <= 0;

if (parameterB == 6'b000000)

R0\_EN <= 0;

if (parameterB == 6'b000001)

R1\_EN <= 0;

if (parameterB == 6'b000010)

R2\_EN <= 0;

if (parameterB == 6'b000011)

R3\_EN <= 0;

end

S5: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 1;

LOAD\_EN <= 0;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (STORE\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

if (parameterA == 6'b000000)

R0\_OUT <= 1;

if (parameterA == 6'b000001)

R1\_OUT <= 1;

if (parameterA == 6'b000010)

R2\_OUT <= 1;

if (parameterA == 6'b000011)

R3\_OUT <= 1;

if (parameterA == 6'b000100)

IO\_OUT0 <= 1;

BUS\_MEM <= 1;

RW <= 0;

end

S1: begin

nextstate <= S2;

if (parameterA == 6'b000000)

R0\_OUT <= 0;

if (parameterA == 6'b000001)

R1\_OUT <= 0;

if (parameterA == 6'b000010)

R2\_OUT <= 0;

if (parameterA == 6'b000011)

R3\_OUT <= 0;

if (parameterA == 6'b000100)

IO\_OUT0 <= 0;

BUS\_MEM <= 0;

MEM\_EN <= 1;

end

S2: begin

if (MFC == 1)begin

nextstate <= S3;

MEM\_EN <= 0;

end

else begin

nextstate <= S2;

end

end

S3: begin

nextstate <= S4;

data = 16'b1111111111111111;

ID\_OUT <= 1;

if (parameterB == 6'b000000)

R0\_EN <= 1;

if (parameterB == 6'b000001)

R1\_EN <= 1;

if (parameterB == 6'b000010)

R2\_EN <= 1;

if (parameterB == 6'b000011)

R3\_EN <= 1;

if (parameterB == 6'b000100)

IO\_EN0 <= 1;

end

S4: begin

nextstate <= S5;

ID\_OUT <= 0;

if (parameterB == 6'b000000)

R0\_EN <= 0;

if (parameterB == 6'b000001)

R1\_EN <= 0;

if (parameterB == 6'b000010)

R2\_EN <= 0;

if (parameterB == 6'b000011)

R3\_EN <= 0;

if (parameterB == 6'b000100)

IO\_EN0 <= 0;

end

S5: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 1;

STORE\_EN <= 0;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (MOV\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

if (parameterB == 6'b000000)

R0\_OUT <= 1;

if (parameterB == 6'b000001)

R1\_OUT <= 1;

if (parameterB == 6'b000010)

R2\_OUT <= 1;

if (parameterB == 6'b000011)

R3\_OUT <= 1;

if (parameterA == 6'b000000)

R0\_EN <= 1;

if (parameterA == 6'b000001)

R1\_EN <= 1;

if (parameterA == 6'b000010)

R2\_EN <= 1;

if (parameterA == 6'b000011)

R3\_EN <= 1;

end

S1: begin

nextstate <= S2;

if (parameterB == 6'b000000)

R0\_OUT <= 0;

if (parameterB == 6'b000001)

R1\_OUT <= 0;

if (parameterB == 6'b000010)

R2\_OUT <= 0;

if (parameterB == 6'b000011)

R3\_OUT <= 0;

if (parameterA == 6'b000000)

R0\_EN <= 0;

if (parameterA == 6'b000001)

R1\_EN <= 0;

if (parameterA == 6'b000010)

R2\_EN <= 0;

if (parameterA == 6'b000011)

R3\_EN <= 0;

end

S2: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 1;

MOV\_EN <= 0;

end

default:

begin

end

endcase

end

always @(posedge clk or posedge reset)

if (reset)begin

state <= S0;

nextstate <= S0;

end

else if (MOVI\_EN)

begin

case(state)

S0: begin

nextstate <= S1;

if (parameterA == 6'b000000)

R0\_EN <= 1;

if (parameterA == 6'b000001)

R1\_EN <= 1;

if (parameterA == 6'b000010)

R2\_EN <= 1;

if (parameterA == 6'b000011)

R3\_EN <= 1;

if (parameterA == 6'b000100)

IO\_EN0 <= 1;

data <= parameterB;

ID\_OUT <= 1;

end

S1: begin

nextstate <= S2;

ID\_OUT <= 0;

if (parameterA == 6'b000000)

R0\_EN <= 0;

if (parameterA == 6'b000001)

R1\_EN <= 0;

if (parameterA == 6'b000010)

R2\_EN <= 0;

if (parameterA == 6'b000011)

R3\_EN <= 0;

if (parameterA == 6'b000100)

IO\_EN0 <= 0;

end

S2: begin

nextstate <= S0;

state <= S0;

IF\_EN <= 1;

MOVI\_EN <= 0;

end

default:

begin

end

endcase

end

always @ (posedge ID\_OUT or posedge clk)

begin

if (ID\_OUT == 1)begin

busOUT = data;

end

else if (ID\_OUT == 0) begin

busOUT = 16'bZZZZZZZZZZZZZZZZ;

end

end

endmodule

module ALU(

input out\_EN,reset,A\_in,B\_in,

input [15:0] busIN, //Bus Input

input [2:0] select, //Selection

output reg [15:0] busOUT //Output to Bus

);

reg [15:0] ALU\_Result, A, B;

always @(\*)

begin

if (reset == 1)

A <= 16'b0000000000000000;

else if (A\_in)

A <= busIN;

end

always @(\*)

begin

if (reset == 1)

B <= 16'b0000000000000000;

else if (B\_in)

B <= busIN;

end

always @(\*)

begin

case(select)

3'b000: ALU\_Result = A + B; //Add

3'b001: ALU\_Result = A - B; //Subtract

3'b010: ALU\_Result = A & B; //AND

3'b011: ALU\_Result = A | B; //OR

3'b100: ALU\_Result = A ^ B; //XOR

3'b101: ALU\_Result = !A; //NOT

3'b110: ALU\_Result = ~(A ^ B); //XNOR

default: ALU\_Result = 16'b0000000000000000;

endcase

if (out\_EN == 1) begin

busOUT <= ALU\_Result;

end

else if (out\_EN == 0) begin

busOUT <= 16'bZZZZZZZZZZZZZZZZ;

end

end

endmodule

‘