Laboratory 5: Simple Processor

Bryan Guner and Haley Scott

ELC 363 – L2

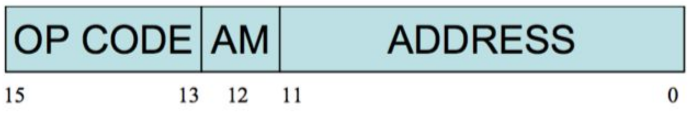
December 4th 2016

Introduction:

The purpose of this laboratory was to implement the architecture of a simple processor using Xilinx' design package for FPGAs and CPLDs. In order to accomplish said goal, considered a simple accumulator based processor, with a 16-bit word length.

Problem Description:

In this experiment, students will be asked to implement the architecture of a simple processor. Students will study the controller flow diagram, as well as the Von Neuman approach. The architecture must utilize machine language to test the program, and use a clock period of 1 microsecond as the timing constraint. The students must follow the guidelines shown below.



If AM = 0, the addressing mode is direct, and if AM = 1, the addressing mode is indirect.

The machine has the following registers only.

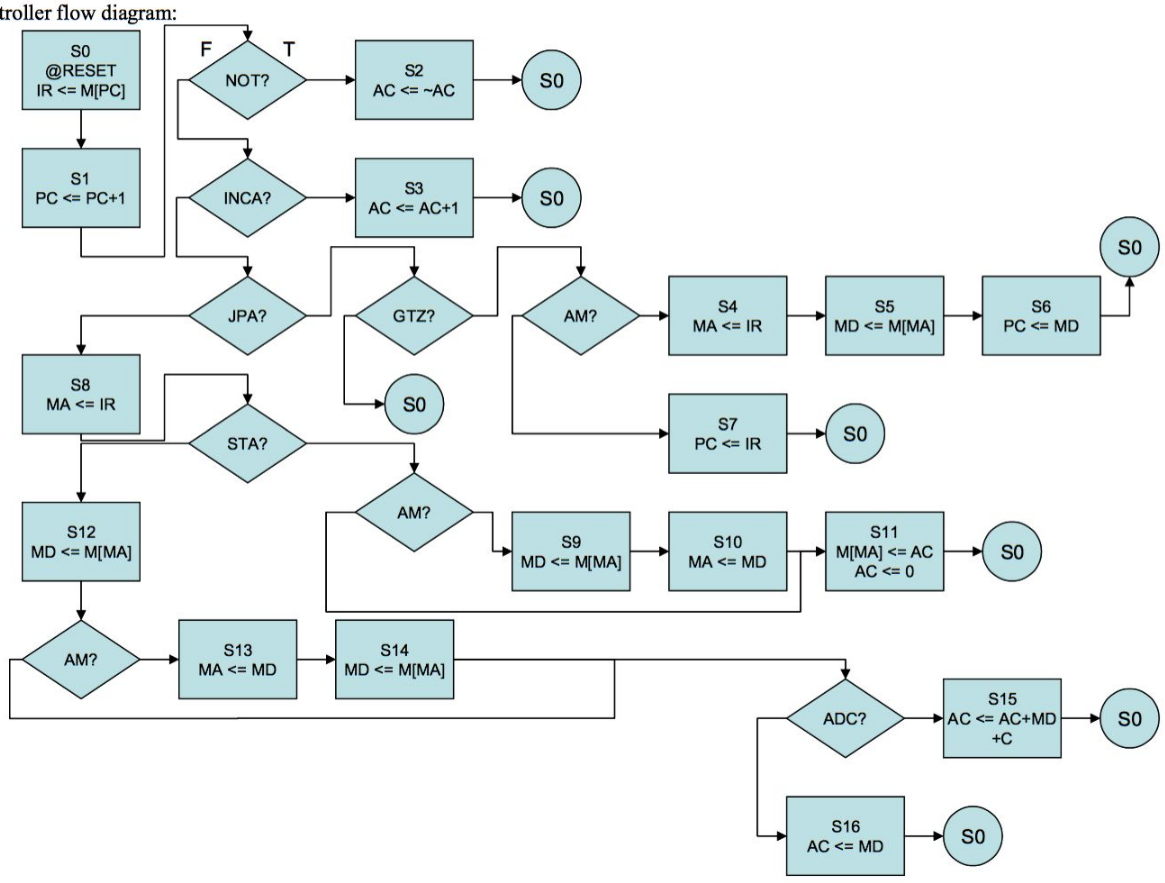
REGISTER NAME	FUNCTION	LENGTH
IR	INSTRUCTION REGISTER	16 bits
MD	MEMORY DATA REGISTER	16 bits
AC	ACCUMULATOR	16 bits
PC	PROGRAM COUNTER	12 bits
MA	MEMORY ADDRESS REGISTER	12 bits

The processor implements the following instructions only.

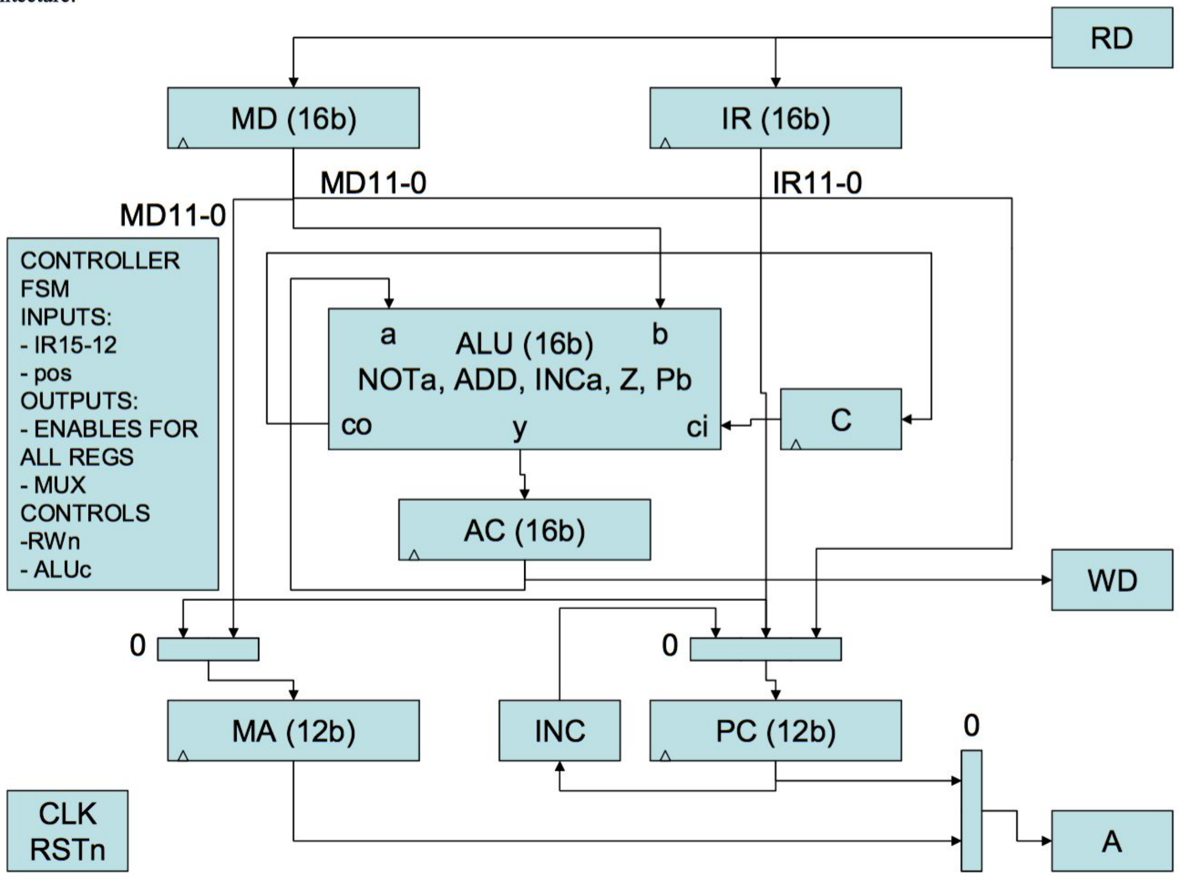
OP CODE	MNEMONIC	DESCRIPTION
000	NOT	INVERT
001	ADC	ADD WITH CARRY
010	JPA	JUMP IF ACC > 0
011	INCA	INCREMENT ACC
100	STA	STORE AND CLEAR ACC
101	LDA	LOAD ACCUMULATOR

Using the above constraints, students must utilize the controller flow diagram shown below. A block diagram of the assigned simple processor architecture is also shown below.

Controller flow diagram:



Architecture:



While utilizing the above constraints, students will implement the simple processor architecture. Students must also provide waveform outputs for the architecture.

Results:

Our team's results display the functionality of our simple processor implementation. We began the coding process by creating various modules to incorporate each component in the simple processor. All code from all modules are shown below.

Alu module:

```
module alu(
input[15:0]a,b,
input[2:0] alu_control,
input ci,
output reg[15:0]y,
output reg co
        );
         always@(*)begin
case(alu_control)
000:y=~a; //not
001:{co,y}=a+b; //adc
011:y=a+1; //inca
100:y=0; //sta
101:y=b; //lda
endcase
end
endmodule
```

Controller Module:

```
module controller(

input [3:0] opcode,
input clk,reset,
input[15:0] AC,
output reg enable_IR,enable_MD,enable_AC,enable_MA,enable_PC,mux1,mux3,RWn,
output [2:0] alu_control,
output reg[1:0] mux2,
reg[4:0]  presentstate,nextstate
);
initial begin
presentstate=5'b00000;
end

always@(posedge clk) begin
presentstate=nextstate;
end
assign
 alu_control= opcode[3:1];

always @ (presentstate) begin
        case(presentstate)
                        5'b00000: begin //s0
                                enable_IR=1;
                                enable_MD=0;
                                enable_AC=0;
                                enable_MA=0;
                                enable_PC=1;
                                mux2=3;
                                mux1=0;
                                mux3=0;
                                RWn=1;
                                nextstate=5'b00001;
```

```
end

5'b00001: begin //s1
        enable_IR=0;
        enable_MD=0;
        enable_AC=0;
        enable_MA=0;
        enable_PC=1;
        mux2=0;
        mux1=0; //do not care
        mux3=0;
        RWn=0;
        if(opcode[3:1]==3'b000)begin
                nextstate=5'b00010; //s2
        end
        else if(opcode[3:1]==3'b011)begin
                nextstate=5'b00011;
        end
        else if(opcode[3:1]==3'b010)begin
                if(AC>0)begin
                        if(opcode[0]==1)begin //AM is lsb of opcode
                                nextstate=5'b00100; //s4
                        end
                        else
                                nextstate=5'b00111;
                end
                else
                        nextstate=5'b00000;
        end
        else
                nextstate=5'b01000;
end

5'b00010:begin //s2
enable_IR=0;
enable_MD=0;
enable_AC=1;
enable_MA=0;
enable_PC=0;
mux2=0;// do not care
mux1=0;//do not care
mux3=0;//do not care
RWn=0; //tells if writing out or reading in (used if reading from RD)
nextstate=5'b00000;
end

5'b00011:begin //s3
enable_IR=0;
enable_MD=0;
enable_AC=1;
```

```
enable_MA=0;
enable_PC=0;
mux2=0;// do not care
mux1=0;//do not care
mux3=0;//do not care
RWn=0;
nextstate=5'b00000;
end

5'b00100:begin //s4
enable_IR=1;
enable_MD=0;
enable_AC=0;
enable_MA=1;
enable_PC=0;
mux2=1;// do not care
mux1=0;//do not care
mux3=1;//do not care
RWn=0;
nextstate=5'b00101;
end

5'b00101:begin //s5 CONCERN
enable_IR=0;
enable_MD=1;
enable_AC=0;
enable_MA=1;
enable_PC=0;
mux2=2;
mux1=1;
mux3=1;
RWn=1; //Whenever state has M[something]
nextstate=5'b00110;
end


5'b00110:begin //s6
enable_IR=0;
enable_MD=1;
enable_AC=0;
enable_MA=0;
enable_PC=1;
mux2=3;                    // MUXES?
mux1=1;
mux3=0;
RWn=0;
nextstate=5'b00000;
end

5'b00111:begin //s7
```

```
enable_IR=1;
enable_MD=0;
enable_AC=0;
enable_MA=0;
enable_PC=1;
mux2=1;
mux1=0;
mux3=0;
RWn=0;
nextstate=5'b00000;
end

5'b01000:begin //s8
enable_IR=1;
enable_MD=0;
enable_AC=0;
enable_MA=1;
enable_PC=0;
mux2=1;
mux1=0;
mux3=1;
RWn=0;
        if(opcode[3:1]==3'b100)begin
                if(opcode[0]==1)begin
                nextstate=5'b01001;
                end
                else
                nextstate=5'b01011;
        end
        else
        nextstate=5'b01100;
end

5'b01001:begin // s9
enable_IR=0;
enable_MD=1;
enable_AC=0;
enable_MA=1;
enable_PC=0;
mux2=2;
mux1=1;
mux3=1;
RWn=1;
nextstate=5'b01010;
end

5'b01010:begin //s10
enable_IR=0;
enable_MD=1;
enable_AC=0;
```

```
        enable_MA=1;
        enable_PC=0;
        mux2=2;
        mux1=1;
        mux3=1;
        RWn=0;
        nextstate=5'b01011;
        end

        5'b01011:begin //s11
        enable_IR=0;
        enable_MD=0;
        enable_AC=1;
        enable_MA=1;
        enable_PC=0;
        mux2=0;//do not care
        mux1=0;//do not care
        mux3=1;
        RWn=1;
        nextstate=5'b01010;
        end


        5'b01100:begin //s12
        enable_IR=0;
        enable_MD=1;
        enable_AC=0;
        enable_MA=1;
        enable_PC=0;
        mux2=2;
        mux1=1;
        mux3=1;
        RWn=1;
        if(opcode[0]==1)begin
                nextstate=5'b01101;
        end
        else begin
                if(opcode[3:1]==001)
                        nextstate=5'b01111;
                else
                        nextstate=5'b10000;
end
        end

        5'b01101:begin //s13
        enable_IR=0;
        enable_MD=1;
        enable_AC=0;
        enable_MA=1;
        enable_PC=0;
```

```
                        mux2=2;
                        mux1=1;
                        mux3=1;
                        RWn=0;
  nextstate=5'b01110;
                        end

                        5'b01110:begin //s14
                        enable_IR=0;
                        enable_MD=1;
                        enable_AC=0;
                        enable_MA=1;
                        enable_PC=0;
                        mux2=2;
                        mux1=1;
                        mux3=1;
                        RWn=1;
                        if(opcode[3:1]==001)
                             nextstate=5'b01111;
                        else
                             nextstate=5'b10000;
                        end



                        5'b01111:begin //s15
                        enable_IR=0;
                        enable_MD=1;
                        enable_AC=1;
                        enable_MA=0;
                        enable_PC=0;
                        mux2=2;
                        mux1=1;
                        mux3=0; //do not care
                        RWn=0;
                        nextstate=5'b00000;
                        end

                        5'b10000:begin //s16
                        enable_IR=0;
                        enable_MD=1;
                        enable_AC=1;
                        enable_MA=0;
                        enable_PC=0;
                        mux2=2;
                        mux1=1;
                        mux3=1;
                        RWn=0;
                        end
endcase
```

```
end
endmodule
```

Data path module:

```
module Datapath(
input mux1,mux3,enable_IR,enable_MD,enable_AC,enable_MA,enable_PC,RWn,clk,reset,
input[2:0]alu_control,
input[1:0]mux2,
input[15:0]RD,
output reg[11:0] A,
output reg [15:0] WD,AC, IR
  );
reg[15:0] MD;
reg[11:0] MA,PC;
reg c ;//both ci and co
reg[15:0] alu_output;


initial begin
PC=0;
c=0;
A=0;
AC=0;
end
alu m1 (.a(AC),.b(MD),.y(alu_output),.ci(c),.co(c),.alu_control(alu_control));

always@(*)begin
if(enable_AC==1)
AC=alu_output;

end

always@(posedge clk)begin
case(RWn)
0:begin
if(enable_AC==1)
WD=AC;
end
1: begin
if(enable_IR==1)
IR=RD;
else if(enable_MD==1)
MD=RD;
end
endcase
//mux1
case(mux1)

0:begin
```

```
if(enable_MA == 1 && enable_IR==1)
 MA=IR;
end

1:begin
if(enable_MA == 1 && enable_MD==1)
 MA=MD;
end

endcase

case(mux2)

0:begin
if(enable_PC == 1)
 PC=PC+1;
end

1:begin
if(enable_PC == 1 && enable_IR==1)
 PC=IR;
end

2:begin
if(enable_PC == 1 && enable_MD==1)
 PC=MD;
 end

endcase

case(mux3)

0:begin
if(enable_PC == 1)
A=PC;
end

1:begin
if(enable_MA == 1)
A=MA;
end

endcase
end
endmodule
```

CPU Module:

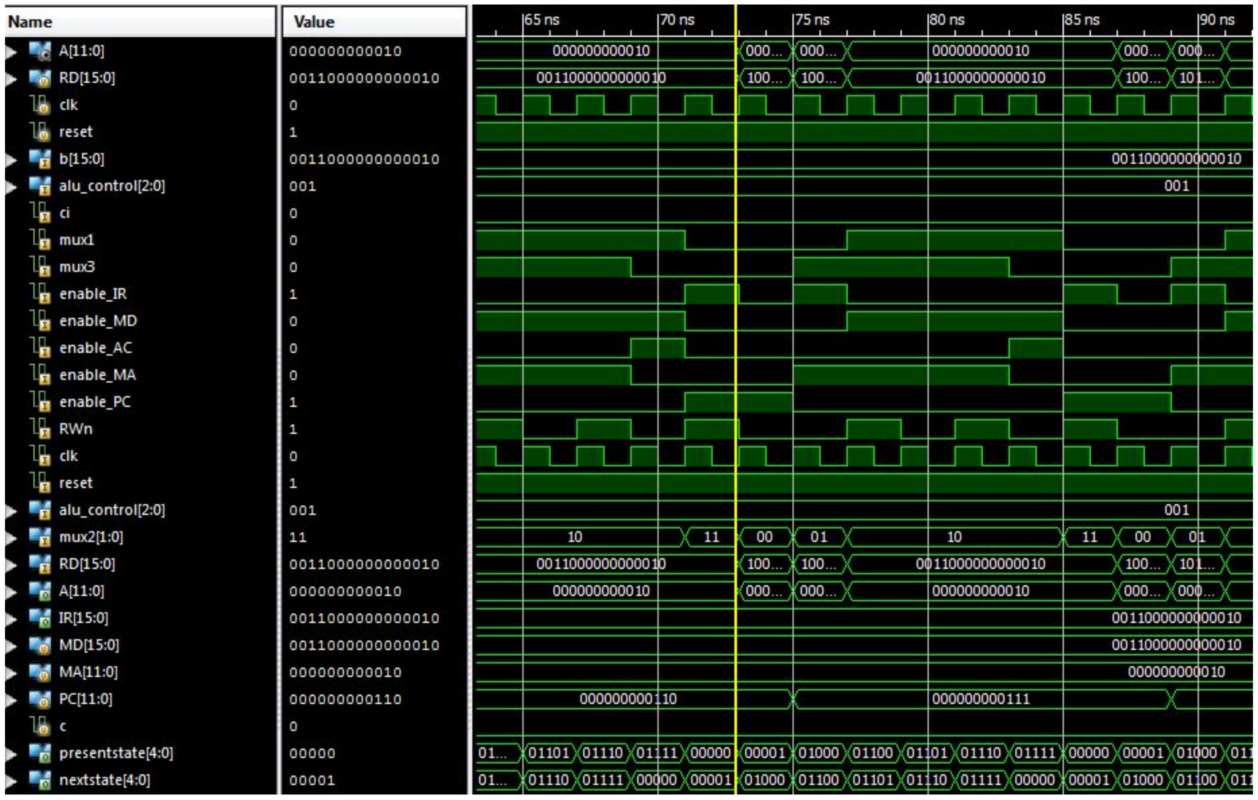
```
module CPU(
input[15:0] RD,
input clk, reset,
output[15:0]WD,
output[11:0] A
            );


wire mux1,mux3,enable_IR,enable_MD,enable_AC,enable_MA,enable_PC,RWn,clk,reset,RWn;
wire [1:0] mux2;
wire[15:0]AC;
wire[2:0]alu_control;
wire[15:0] IR;


Datapath m2
(.mux1(mux1),.mux2(mux2),.mux3(mux3),.enable_IR(enable_IR),.enable_MD(enable_MD),.enable_AC(enable_AC),.enable_M
A(enable_MA),.enable_PC(enable_PC),.RWn(RWn),.alu_control(alu_control),.clk(clk),.reset(reset),.RD(RD),.A(A),.WD(WD),.
AC(AC), .IR(IR));
controller m3
(.opcode(IR[15:12]),.clk(clk),.reset(reset),.AC(AC),.enable_IR(enable_IR),.enable_MD(enable_MD),.enable_AC(enable_AC),.e
nable_MA(enable_MA),.enable_PC(enable_PC),.mux1(mux1),.mux2(mux2),.mux3(mux3),.RWn(RWn),.alu_control(alu_control
));
Endmodule
```

CPU testbench:

```
module CPU_test_hope;
            // Inputs
            reg [15:0] RD;
            reg clk;
            reg reset;

            // Outputs
            wire [11:0] A;
            wire [15:0] WD;


reg[15:0] instructionmem[63:0];
            // Instantiate the Unit Under Test (UUT)
            CPU uut (
                        .RD(RD),
                        .clk(clk),
                        .reset(reset),
                        .A(A),
                        .WD(WD)
            );

            initial begin
instructionmem[0]=16'b0000000000000001;//Not
instructionmem[1]=16'b0010000000000010;//ADC
instructionmem[2]=16'b0011000000000010;//ADC w AM=1
```

```
instructionmem[3]=16'b0100000000000011;//JPA
instructionmem[4]=16'b0101000000000011;//JPA w AM=1
instructionmem[5]=16'b0110000000000100;//INCA
instructionmem[6]=16'b1000000000000101;//STA
instructionmem[7]=16'b1001000000000101;//STA w AM=1
instructionmem[8]=16'b1010000000000111;//LDA
instructionmem[9]=16'b1011000000000111;//LDA w AM=1



                              RD = 0;
                              clk = 0;
                              reset = 1;

                              // Wait 100 ns for global reset to finish
                              #100;

                              // Add stimulus here

          end
          always @ (clk) begin
                    #1;
                    clk <= ~clk;
                                        end

          always @(A) begin
          RD=instructionmem[A];


          end
always@(WD)
instructionmem[A]=WD;

endmodule
```

After utilizing the test bench to test our processor, our team captured the output waveform from the processor. The input is shown above in the Test Bench Module section of our code, and the waveform is shown below, in Figure 1.



Name	Value
A[11:0]	000000000010
RD[15:0]	0011000000000010
clk	0
reset	1
b[15:0]	0011000000000010
alu_control[2:0]	001
ci	0
mux1	0
mux3	0
enable_IR	1
enable_MD	0
enable_AC	0
enable_MA	0
enable_PC	1
RWn	1
clk	0
reset	1
alu_control[2:0]	001
mux2[1:0]	11
RD[15:0]	0011000000000010
A[11:0]	000000000010
IR[15:0]	0011000000000010
MD[15:0]	0011000000000010
MA[11:0]	000000000010
PC[11:0]	000000000110
c	0
presentstate[4:0]	00000
nextstate[4:0]	00001

Figure 1: Waveform Output for Simple Processor

Conclusions:

Overall, our team successfully implemented a simple processor architecture using Verilog. We implemented the proper code for each module, as well as initialized a testbench that takes machine code in as an input. We implemented the main components as modules, including an ALU module, a Controller module, a Data Path module, and a CPU module. The main takeaways from this lab are to understand the concept of a Von Neuman architecture, and understand the implementation of a simple processor in Verilog.