Computer Engineering - ELC 363

Lab 5 Report: ARM Implementation

Nick Smith and Brian Dawson

Due: 12/5/19

Introduction and Problem Description:

In this lab, students implemented the ARM architecture in a single cycle format using Verilog. Using an already architecturally structured processor and using a bottoms-up approach, the intention of this lab was to further introduce and develop students understanding regarding processor structure and high level abstract design methodology using a hardware description language. The implementation is able to support memory-reference instructions LDUR and STUR, arithmetic-logical instructions ADD, SUB, AND, and ORR, and control flow instruction CBZ.

Results:

a)

I. Design Code:

```
`timescale 1ns / 1ps
        module CPU(
        input clk,
        input reset,
        input [31:0] instruction_from_testbench,
        input [63:0] datamemreaddata_to_mux3,
        output reg [63:0] pcoutput_to_testbench,
        output reg [63:0] mainalu_to_datamemreaddata,
        output reg [63:0] readdata2_to_datamemwritedata,
        output reg memwrite,memread
        );


        reg [10:0] instruction31_21_to_control;
        reg [10:0] instruction31_21_to_alu;
        reg [31:0] instruction31_0_to_signextend;
```

```
reg [4:0] instruction20_16_to_mux1;
reg [4:0] instruction9_5_to_readregaddr1;
reg [4:0] instruction4_0_to_mux1;
reg [4:0] instruction4_0_to_writeregaddr;

//Control lines
wire reg2loc_to_mux1;
wire [1:0] aluop_to_alucontrol;
wire memtoreg_to_mux3;
wire alusrc_to_mux2;
wire regwrite_to_regfile;
wire memread_to_datamem;
wire memwrite_to_datamem;

// Mux 4 wires
wire [63:0] mux4_to_pcinput;
wire [63:0] pcalu_to_mux4;
wire [63:0] pcoutput_to_shiftalu;
wire shiftalu_to_mux4;
wire cbz_to_mux4;
wire [63:0] signextended_to_shiftalu;
wire branch_to_cbz;
wire andout;
wire mainaluzero_to_cbz;
wire [63:0] mainalu_result;
wire [4:0] mux1_to_readregaddr2;
wire [63:0] readdata1_to_mainalu;
wire [63:0] readdata2_to_mux2;
wire [63:0] signextended_to_mux2;
wire [63:0] mux2_to_mainalu;
wire [3:0] alu_to_mainalu;
```

```
wire [63:0] mainalu_to_mux3;
wire [63:0] mux3_to_writedata;
//Branch AND
assign andout = (branch_to_cbz & mainaluzero_to_cbz);
//Testbench to instruction
always @ (instruction_from_testbench) begin
instruction31_21_to_control = instruction_from_testbench[31:21];
instruction31_21_to_alu = instruction_from_testbench[31:21];
instruction31_0_to_signextend = instruction_from_testbench;
instruction20_16_to_mux1 = instruction_from_testbench[20:16];
instruction9_5_to_readregaddr1 = instruction_from_testbench[9:5];
instruction4_0_to_mux1 = instruction_from_testbench[4:0];
instruction4_0_to_writeregaddr = instruction_from_testbench[4:0];
end
//ALU
Alu alu_inst
(.readdata1(readdata1_to_mainalu),.main_aluinput2(mux2_to_mainalu),.ALU(alu_to_mainalu),.y(mainalu_result),.z(
mainaluzero_to_cbz));
always @ (mainalu_result) begin
mainalu_to_datamemreaddata = mainalu_result;
readdata2_to_datamemwritedata=readdata2_to_mux2;
memwrite= memwrite_to_datamem;
memread=memread_to_datamem;
end
Control control_inst

(.opcode(instruction31_21_to_control),.reg2loc(reg2loc_to_mux1),.branch(branch_to_cbz),.memread(memr
ead_to_datamem),.memtoreg(memtoreg_to_mux3),.aluop(aluop_to_alucontrol),.memwrite(memwrite_to_datamem),.
alusrc(alusrc_to_mux2),.regwrite(regwrite_to_regfile));
mux1 mux1_inst
```

```
(.instruction20_16(instruction20_16_to_mux1),.instruction4_0(instruction4_0_to_mux1),.reg2loc(reg2loc_to_
mux1),.readregaddr2(mux1_to_readregaddr2));
mux2 mux2_inst

(.readdata2(readdata2_to_mux2),.extended(signextended_to_mux2),.alusrc(alusrc_to_mux2),.main_aluinpu
t2(mux2_to_mainalu));
mux3 mux3_inst

(.datamem_readdata(datamemreaddata_to_mux3),.y(mainalu_result),.memtoreg(memtoreg_to_mux3),.write
data(mux3_to_writedata));
mux4 mux4_inst
(.pc_output(pcalu_to_mux4),.extended(signextended_to_mux2),.cbz(andout),.pc(mux4_to_pcinput));
always@(*)begin
pcoutput_to_testbench=mux4_to_pcinput;
end
pc pc_inst (.pc_input(mux4_to_pcinput),.pc_output(pcalu_to_mux4),.clk(clk));
registerfile reg_inst

(.readregaddr1(instruction9_5_to_readregaddr1),.readregaddr2(mux1_to_readregaddr2),.writeregaddr(instru
ction4_0_to_writeregaddr),.writedata(mux3_to_writedata),.readdata1(readdata1_to_mainalu),.readdata2(readdata2_t
o_mux2),.reset(reset),.clk(clk),.regwrite(regwrite_to_regfile));
signextend extend_inst (.unextended(instruction31_0_to_signextend),.extended(signextended_to_mux2));
alu_control alucontrol_inst
(.aluop(aluop_to_alucontrol),.instruction31_21(instruction31_21_to_alu),.alu_controlline(alu_to_mainalu));

endmodule
module Alu(
input [63:0] readdata1,
input [63:0] main_aluinput2,
input [3:0] ALU,
```

```
output reg [63:0] y,
output reg z
);
always@(readdata1 or main_aluinput2 or ALU)
begin
        case(ALU)
        4'b0000: assign y = readdata1 & main_aluinput2;
        4'b0001: assign y = readdata1 | main_aluinput2;
        4'b0010: assign y = readdata1 + main_aluinput2;
        4'b0110: assign y = readdata1 - main_aluinput2;
        4'b0111: assign y = main_aluinput2 ;
        4'b1100: assign y = ~(readdata1 | main_aluinput2);
        endcase

if (y == 0)
        begin
        z = 1;
        end

else

        begin
        z = 0;
        end
end
endmodule
module Control(
input [10:0] opcode,
output reg [1:0] aluop,
output reg reg2loc,
output reg alusrc,
```

```
output reg memtoreg,
output reg regwrite,
output reg memread,
output reg memwrite,
output reg branch
        );
        always@(*) begin
        if (opcode[10:3]==8'b10110100) begin
        reg2loc=1'b1;
        alusrc=1'b0;
        regwrite=1'b0;
        memread=1'b0;
        memwrite=1'b0;
        branch=1'b1;
        aluop=2'b01;

        end

        else
        case(opcode)

        11'b10001011000: begin //ADD
        reg2loc=1'b0;
        alusrc=1'b0;
        memtoreg=1'b0;
        regwrite=1'b1;
        memread=1'b0;
        memwrite=1'b0;
        branch=1'b0;
        aluop=2'b10;
        end
```

```
11'b11001011000: begin //SUB
reg2loc=1'b0;
alusrc=1'b0;
memtoreg=1'b0;
regwrite=1'b1;
memread=1'b0;
memwrite=1'b0;
branch=1'b0;
aluop=2'b10;
end

11'b10001010000: begin//AND
reg2loc=1'b0;
alusrc=1'b0;
memtoreg=1'b0;
regwrite=1'b1;
memread=1'b0;
memwrite=1'b0;
branch=1'b0;
aluop=2'b10;
end

11'b10101010000: begin //ORR
reg2loc=1'b0;
alusrc=1'b0;
memtoreg=1'b0;
regwrite=1'b1;
memread=1'b0;
memwrite=1'b0;
branch=1'b0;
```

```
            aluop=2'b10;
            end

            11'b11111000010: begin //LDUR
            alusrc=1'b1;
            memtoreg=1'b1;
            regwrite=1'b1;
            memread=1'b1;
            memwrite=1'b0;
            branch=1'b0;
            aluop=2'b00;
            end

            11'b11111000000:begin //STUR
            reg2loc=1'b1;
            alusrc=1'b1;
            regwrite=1'b0;
            memread=1'b0;
            memwrite=1'b1;
            branch=1'b0;
            aluop=2'b00;
            end
            endcase
            end
endmodule
module mux1(
input [4:0] instruction20_16,
input [4:0] instruction4_0,
input reg2loc,
output reg [4:0] readregaddr2
);
```

```
always @(*) begin
case(reg2loc)
        1'b0:readregaddr2=instruction20_16;
        1'b1:readregaddr2=instruction4_0;
endcase
end
endmodule
module mux2(
input [63:0] readdata2,
input [63:0] extended,
input alusrc,
output reg [63:0] main_aluinput2
);
always @(*) begin
case(alusrc)
        1'b0:main_aluinput2=readdata2;
        1'b1:main_aluinput2=extended;
endcase
end
endmodule
module mux3(
input memtoreg,
input [63:0] y,
input [63:0] datamem_readdata,
output reg [63:0] writedata
);
always @(*) begin
case(memtoreg)
1'b1: writedata=datamem_readdata;
1'b0: writedata=y;
endcase
```

```
end
endmodule
module mux4(
input cbz,
input [63:0] pc_output,
input [63:0] extended,
output reg [63:0] pc
);
initial begin
pc=0;
end

always @(*) begin
case(cbz)
        1'b1: pc = (extended<<2)+pc_output;
        1'b0: pc = pc_output;
endcase

end
endmodule
module pc(
input clk,
input [63:0] pc_input,
output reg [63:0] pc_output
);

initial begin
        pc_output = 0;
end

always @(posedge clk)begin
```

```
        pc_output=pc_input+4;
end
endmodule
module registerfile (
input reset,
input clk,
input regwrite,
input [4:0] readregaddr1,readregaddr2,
input [63:0] writedata,
input [4:0] writeregaddr,
output reg [63:0] readdata1,readdata2
        );
reg [63:0] regs[31:0];//Creates the 32,64-bit registers
integer i;

initial begin
for(i=0;i<64;i=i+1) begin
regs[i] <= i;
end
end
always@(*) begin //for read
readdata1= regs[readregaddr1];
readdata2= regs[readregaddr2];
end
always@(posedge clk)begin
if(regwrite == 1'b1)begin
case(writeregaddr) // Decoder for part b. of register file
        5'b00000: regs[0] = writedata;
        5'b00001: regs[1] = writedata;
        5'b00010: regs[2] = writedata;
        5'b00011: regs[3] = writedata;
```

```
            5'b00100: regs[4] = writedata;
            5'b00101: regs[5] = writedata;
            5'b00110: regs[6] = writedata;
            5'b00111: regs[7] = writedata;
            5'b01000: regs[8] = writedata;
            5'b01001: regs[9] = writedata;
            5'b01010: regs[10] = writedata;
            5'b01011: regs[11] = writedata;
            5'b01100: regs[12] = writedata;
            5'b01101: regs[13] = writedata;
            5'b01110: regs[14] = writedata;
            5'b01111: regs[15] = writedata;
            5'b10000: regs[16] = writedata;
            5'b10001: regs[17] = writedata;
            5'b10010: regs[18] = writedata;
            5'b10011: regs[19] = writedata;
            5'b10100: regs[20] = writedata;
            5'b10101: regs[21] = writedata;
            5'b10110: regs[22] = writedata;
            5'b10111: regs[23] = writedata;
            5'b11000: regs[24] = writedata;
            5'b11001: regs[25] = writedata;
            5'b11010: regs[26] = writedata;
            5'b11011: regs[27] = writedata;
            5'b11100: regs[28] = writedata;
            5'b11101: regs[29] = writedata;
            5'b11110: regs[30] = writedata;
            5'b11111: regs[31] = writedata;
    endcase
    end
    end
```

```
endmodule
module signextend(
input [31:0] unextended,
output reg [63:0] extended
        );
        reg signed [51:0] temp;
always@(*) begin
if (unextended[31:24]==8'b10110100) begin
        extended [63:19]= {45{unextended[23]}};
        extended [18:0]= unextended[23:5];

        end

else begin

        extended[63:9]={55{unextended[20]}};
        extended[8:0]=unextended[20:12];

        end
end
endmodule
module alu_control(
input[1:0] aluop,
input [10:0] instruction31_21,
output reg [3:0] alu_controlline
        );

always@(*)begin
case(aluop)
        2'b00 : alu_controlline=4'b0010;
        2'b01 : alu_controlline=4'b0111;
```

```
            2'b10 : begin
            if(instruction31_21==11'b10001011000)
            alu_controlline=4'b0010;
            else if(instruction31_21==11'b11001011000)
            alu_controlline=4'b0110;
            else if(instruction31_21==11'b10001010000)
            alu_controlline=4'b0000;
            else if(instruction31_21==11'b10101010000)
            alu_controlline=4'b0001;
            end
endcase
end
endmodule
module datamemory(
input [63:0] main_aluresult,
input [63:0] datamem_writedata,
input memwrite,
input memread,
output reg [63:0] datamem_readdata
            );

integer i=0;
reg [63:0] storage[31:0];

initial begin
for(i=0;i<64;i=i+1) begin
            storage[i]<=2*i;
end
end

always@(*) begin
```

```
        if(memwrite==1'b1)begin
        storage[main_aluresult]=datamem_writedata;
        end

        if(memread==1'b1) begin
        datamem_readdata= storage[main_aluresult];
        end
        end

endmodule
module instructionmemory(
        input [63:0] pc,
        output reg [31:0] instruction31_0
        );
        reg [31:0] instructions[63:0];
        initial begin
        //Instructions
        instructions[0]  = 32'b00000000000000000000000000000000;
        instructions[4]  = 32'b10001011000001000000000010100011; // ADD    x3, x5, x4
        instructions[8]  = 32'b11001011000001100000000000100001; // SUB    x1, x1, x6
        instructions[12] = 32'b10001010000001000000000001100111; // AND   x7, x3, x4
        instructions[16] = 32'b10101010000001100000000010101000; // ORR   x8, x5, x6
        instructions[20] = 32'b11111000010000000100000100001010; // LDUR   x10, [x8, #4]
        instructions[24] = 32'b11111000000000001000000101101100; // STUR x12, [ x16, #8]
        instructions[28] = 32'b10110100000000000000000000000000; // CBZ    x0, #0
        end
        always@(pc) begin
        instruction31_0 = instructions[pc];
        end
endmodule
```

II. Test-Bench Code Files:

```
`timescale 1ns / 1ps


module testbench(
        );


        // Inputs
        reg clk;
        reg reset;
        reg [31:0] instruction_from_testbench;
        reg [63:0] datamemreaddata_to_mux3;
        reg [63:0] pc_to_readaddress;
        reg [63:0] mainalu_to_datamemreaddata_transfer;
        reg [63:0] readdata2_to_datamemwritedata_transfer;
        reg memread_transfer;
        reg memwrite_transfer;
        wire [63:0] pcoutput_from_cpu;
        wire memread;
        wire memwrite;
        wire [63:0] mainalu_to_datamemreaddata;
        wire [63:0] readdata2_to_datamemwritedata;
        wire[31:0] instruction_from_testbench_transfer;
        wire[63:0]datamemreaddata_to_mux3_transfer;


        CPU uut (
        .clk(clk),
        .reset(reset),
        .instruction_from_testbench(instruction_from_testbench),
        .pcoutput_to_testbench(pcoutput_from_cpu),
        .mainalu_to_datamemreaddata(mainalu_to_datamemreaddata),
        .readdata2_to_datamemwritedata(readdata2_to_datamemwritedata),
```

```
.memwrite(memwrite),
.memread(memread),
.datamemreaddata_to_mux3(datamemreaddata_to_mux3)
);
initial begin
clk = 0;
reset = 0;
instruction_from_testbench = 0;

#100;

end
always@(*)begin
mainalu_to_datamemreaddata_transfer = mainalu_to_datamemreaddata;
readdata2_to_datamemwritedata_transfer=readdata2_to_datamemwritedata;
memwrite_transfer= memwrite;
memread_transfer=memread;
end

datamemory datamem_inst
(.main_aluresult(mainalu_to_datamemreaddata_transfer),.datamem_writedata(readdata2_to_datamemwrite
data_transfer),.memwrite(memwrite_transfer),.memread(memread_transfer),.datamem_readdata(datamemreaddata_
to_mux3_transfer));
always@(*)begin
datamemreaddata_to_mux3 = datamemreaddata_to_mux3_transfer;
end
always@(*)begin
pc_to_readaddress=pcoutput_from_cpu;
end

instructionmemory
```

```
Instrucmem_inst (.pc(pc_to_readaddress),.instruction31_0(instruction_from_testbench_transfer));
always@(*)begin
instruction_from_testbench=instruction_from_testbench_transfer;

end
always @ (clk) begin
#50;
clk <= ~clk;
end
endmodule
```

b and c) Assembly language and Machine Language

Order	Assembly Instruction Set	Binary Representation
1.	ADD x3, x5, x4	32'b 10001011000001000000000010100011
2.	SUB x1, x1, x6	32'b 11001011000001100000000000100001
3.	AND x7, x3, x4	32'b 10001010000001000000000001100111
4.	ORR x8, x5, x6	32'b 10101010000001100000000010101000
5.	LDUR x10, [x8, #4]	32'b 11111000010000000100000100001010
6.	STUR x12, [x16, #8]	32'b 11111000000000001000000101101100
7.	CBZ x0, #0	32'b 10110100000000000000000000000000

Table 1: *Test program in ARM assembly language & initial program memory load file*

d) Waveforms

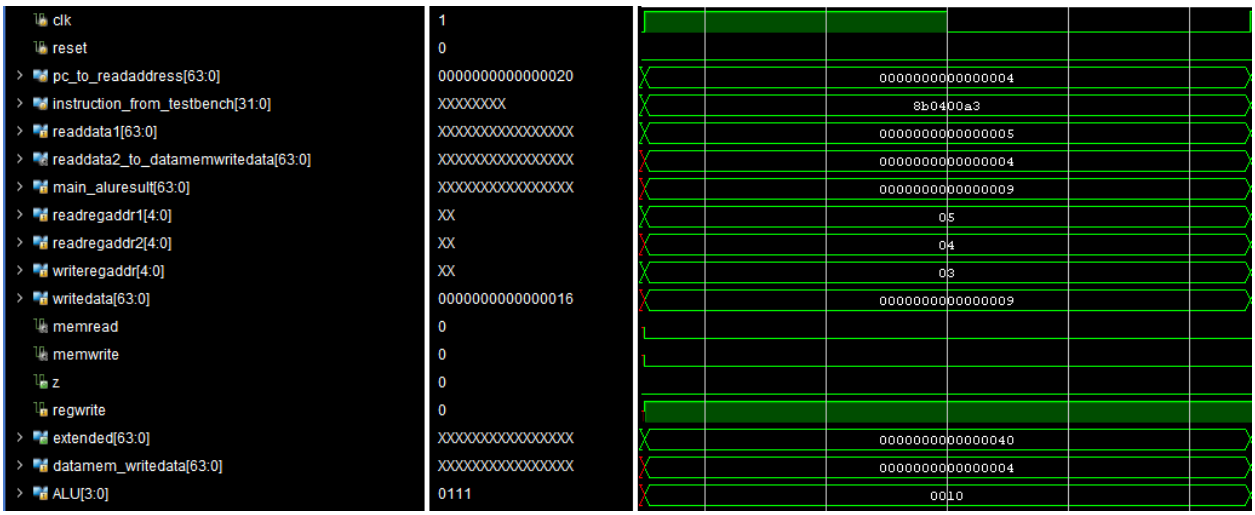

Figure 1: *ADD Instruction (x3, x5, x4)*

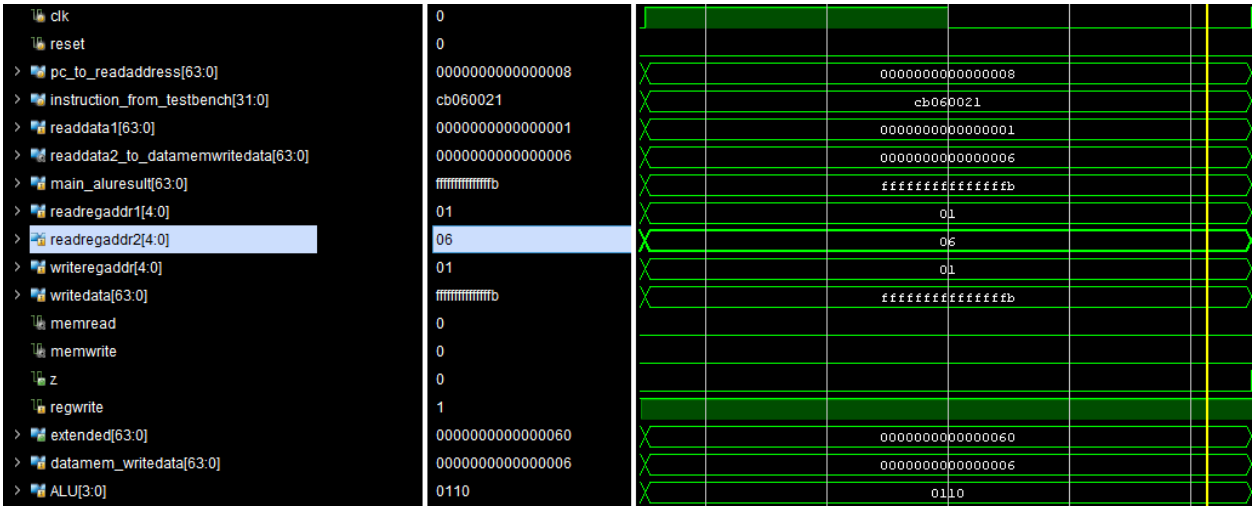

Figure 2: *SUB Instruction (x1, x1, x6)*

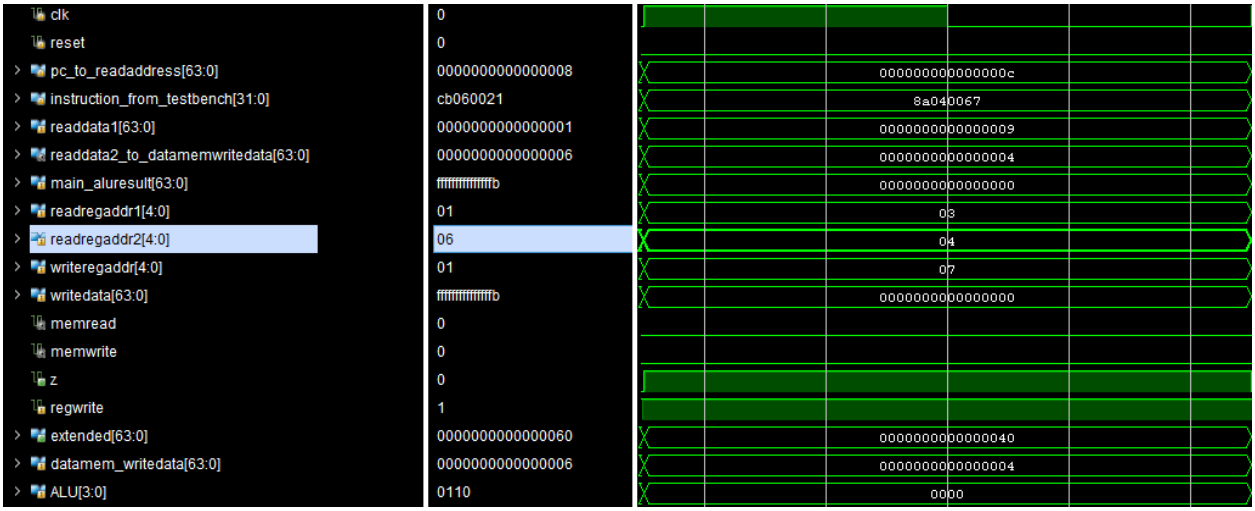

Figure 3: *AND Instruction (x7, x3, x4)*

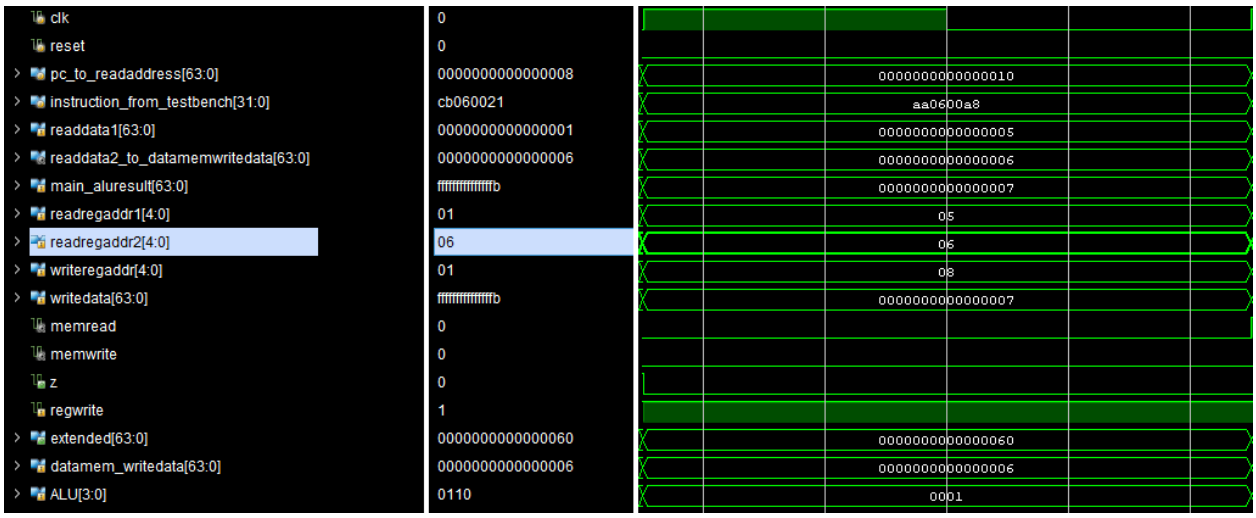

Figure 4: *ORR Instruction (x8, x5, x6)*

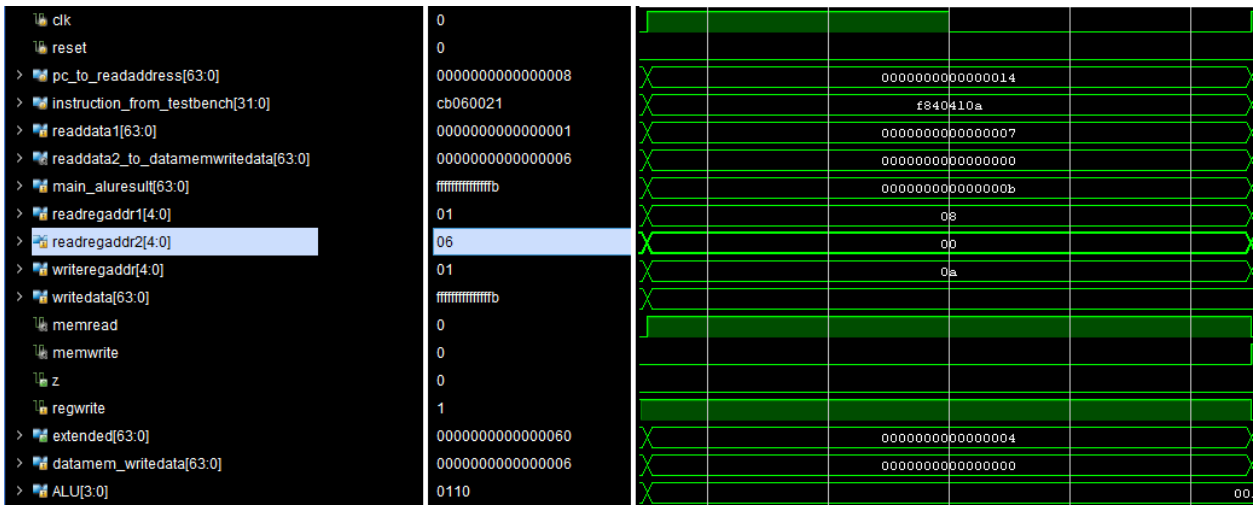

Figure 5: *LDUR Instruction (x10, [x8, #4])*

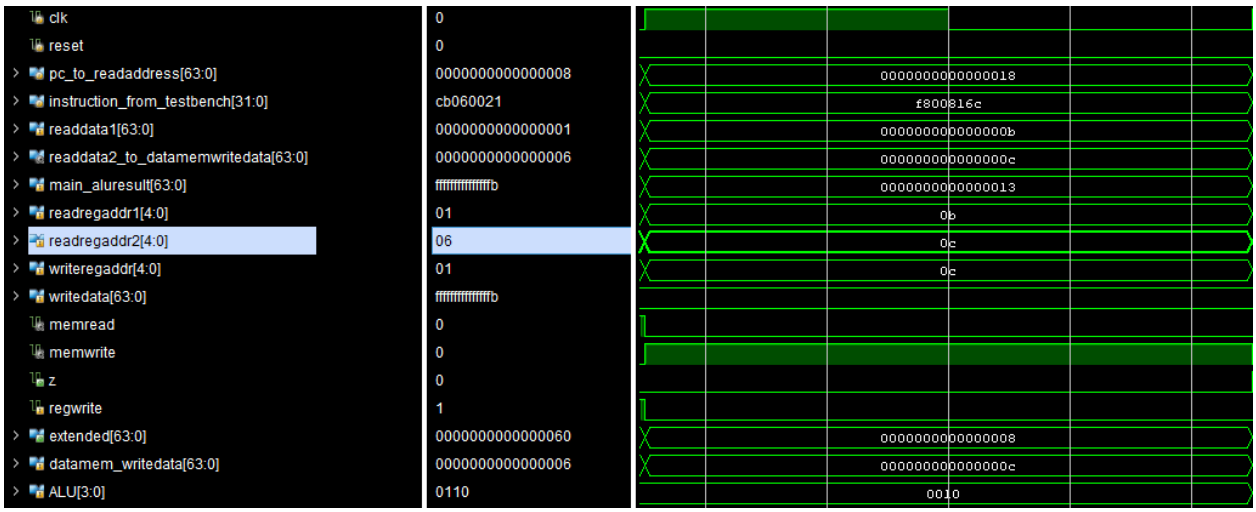

Figure 6: *STUR Instruction (x12, [x16, #8])*

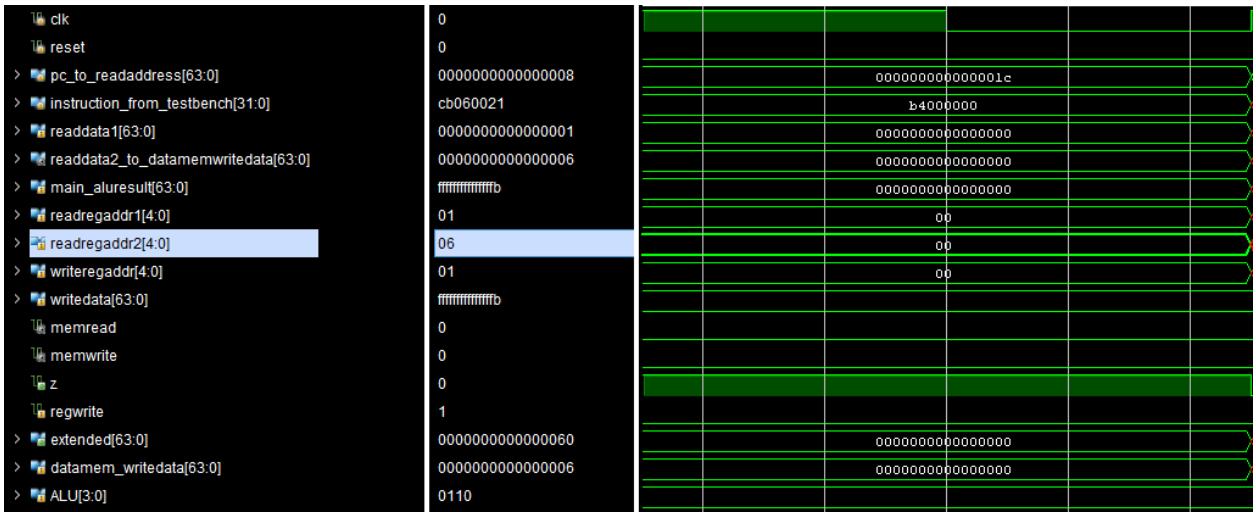

Figure 7: *CBZ Instruction (x0, #0)*

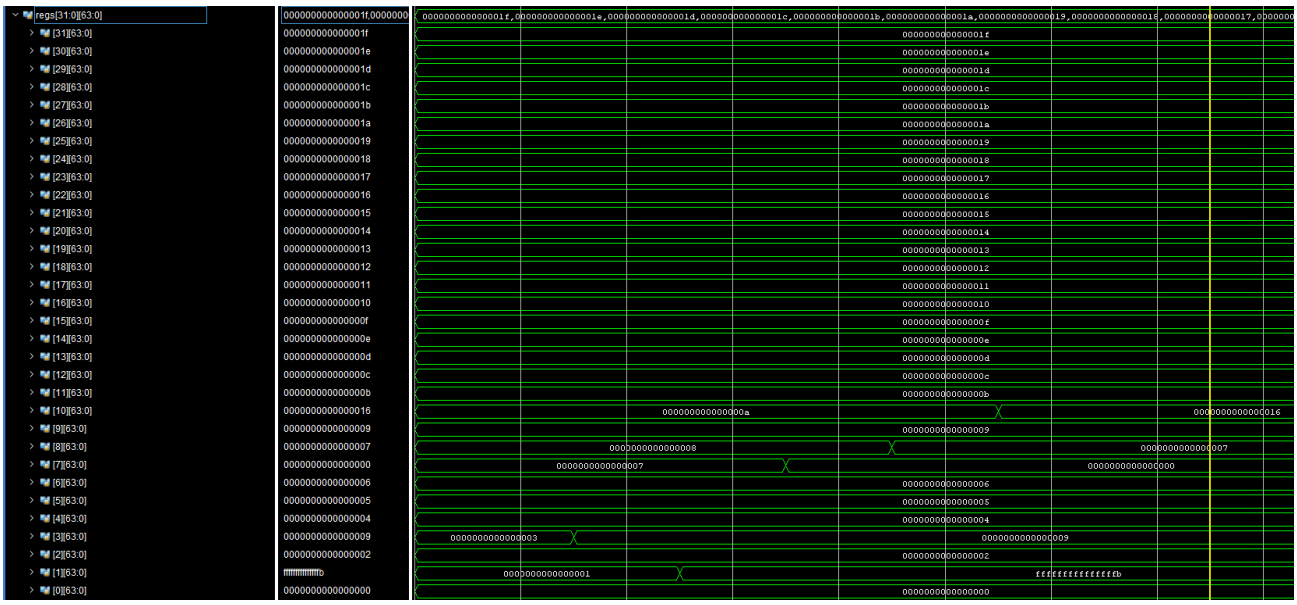

Figure 8: *Register Values*

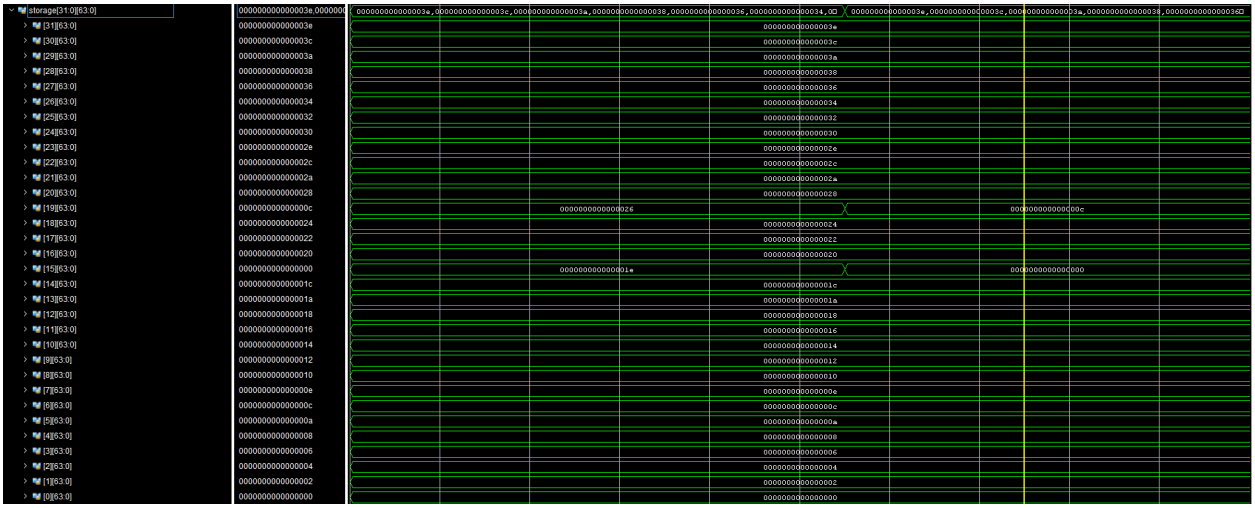

Figure 9: *Data memory Values*

Discussion:

Students approached this design with a bottom-up methodology, which refers to an approach within programming that dictates that the implementation of a project is divided up into sizable modules, all with individual functionality that are then connected together to finally format the

final design. Additionally, students in this lab used the schematic within the lab handout of an overall architecture of an ARM processor as a reference for this project. Determining what those individual elements would look like, students isolated the principal components such as the instruction memory, data memory, control and the register file, as well as less pertinent components such as the individual muxes, ALU's, sign extend/shift left 2, and adders.

Conclusions:

This lab helped develop programming skills within Xilinx Verilog software, as well as improving students methodology regarding high level abstraction in processor design. Another critical aspect was further introducing students to the bottoms-up approach to program design formatting. It simultaneously provided more experience in general coding/debugging and introduced the system of elements that make up a simple processor design in ARM architecture. While definitely being one of the more complex projects done in this lab, this system has great potential for practical application in industrial settings. Experience gained from understanding the development and implementation of processor schematics on a fundamental level will prove to be invaluable later on.