

***Introduction***

In this lab we Implemented a MIPS single cycle processor with new functionality. This was done by adding new hardware to a base version provided to us.

Tasks which were completed in lab:

* Custom official draft of extended datapath and control unit.
* Custom official draft of extended MIPS microarchitecture.
* Control unit truth table.
* Design and verify MIPS processor extension design.
  + Waveform software simulation
  + FPGA hardware simulation

***Design Methodology***

The basis of this lab was the single-cycle MIPS processor provided in previous labs. This processor was capable of the ADD, SUB, AND, OR, SLT, LW, SW, BEQ, J, and ADDI functions. The content of this lab required that we add more to the architecture to provide functionality for the MULTU, MFHI, MFLO, JR, and JAL. Machine code was provided to the us for testing purposes. With this code, we were able to verify proper functionality through a visual testbench as well as implementing the design on the Nexys4 DDR FPGA. In the following sections, we have provided images and tables that demonstrate the validation process. These figures demonstrate both the expected results as well as the actual outcome. This lab taught us how to validate a processor design using a FPGA board. This allows us a deeper understanding of processor functionality.

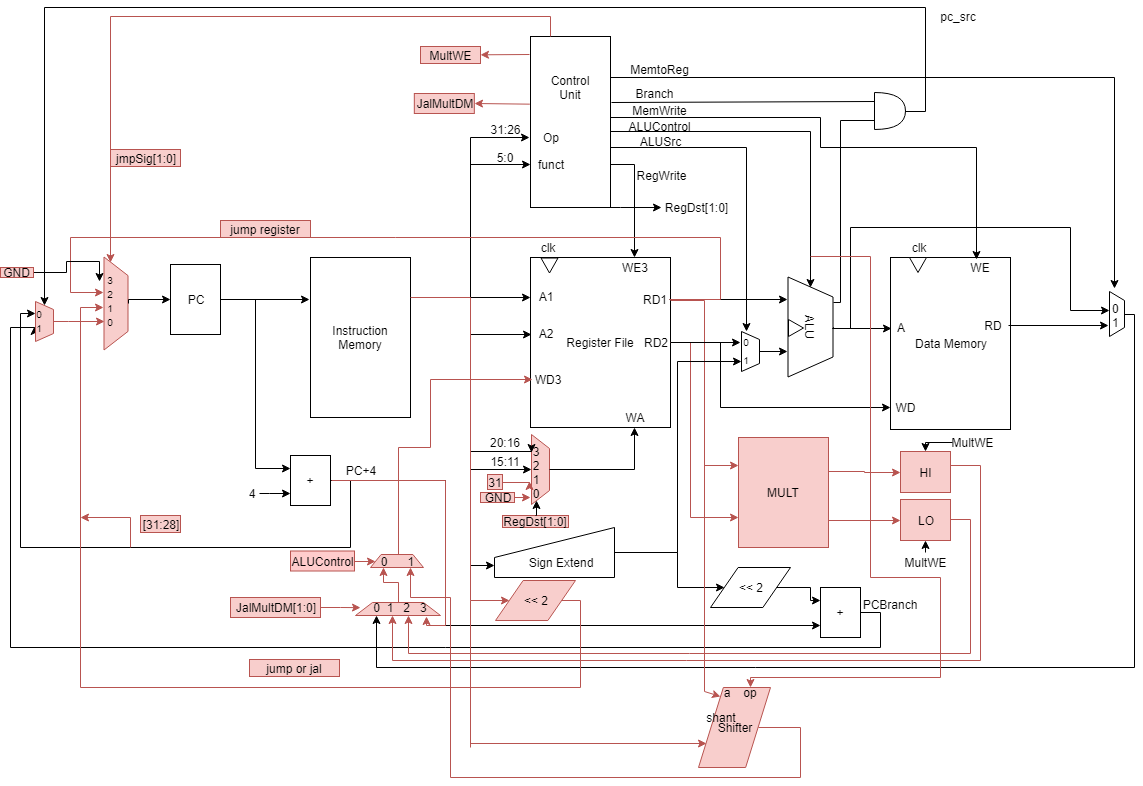


Figure 1. Extended Datapath and Control Unit

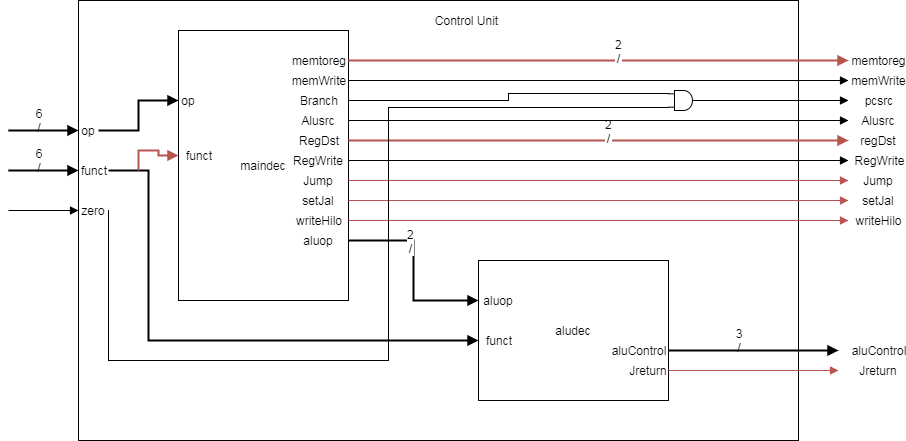


Figure 2. Control Unit

Table 1. Control Unit Output Table

|  | Input | | Output | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | opcode | Funct | RegWrite | RegDst [1:0] | AluSrc | Branch | MemWrite | MemtoReg [1:0] | Jump | JReturn | selJal | writehilo | Aluop | AluControl |
| add | 000000 | 100000 | 1 | 01 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 10 | 010 |
| sub | 000000 | 100010 | 1 | 01 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 10 | 110 |
| and | 000000 | 100100 | 1 | 01 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 10 | 000 |
| or | 000000 | 100101 | 1 | 01 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 10 | 001 |
| slt | 000000 | 101010 | 1 | 01 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 10 | 111 |
| lw | 100011 | x | 1 | 00 | 1 | 0 | 0 | 01 | 0 | 0 | 0 | 0 | 00 | 010 |
| sw | 101011 | x | 0 | xx | 1 | 0 | 1 | xx | 0 | 0 | 0 | 0 | 00 | 010 |
| beq | 000100 | x | 0 | xx | 0 | 1 | 0 | xx | 0 | 0 | 0 | 0 | 01 | 110 |
| addi | 001000 | x | 1 | 00 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 00 | 010 |
| j | 000100 | x | 0 | xx | x | x | 0 | xx | 1 | 0 | 0 | 0 | xx | xxx |
| jr | 000000 | 001000 | 0 | xx | x | 0 | 0 | 00 | 0 | 1 | 0 | 0 | 10 | 010 |
| jal | 000011 | x | 1 | 10 | 0 | 0 | 0 | xx | 1 | 0 | 1 | 0 | xx | xxx |
| slr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ssl |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

***Simulation Test Plan***

The simulation was the result of the testbench used to validate the architecture design built in this lab. The testbench ran the provided machine code through our design and the simulation output the results. By careful examination, we could validate inputs and outputs of the new functions included in our design. We could also validate the contents of specific registers at any given time in the program runtime.

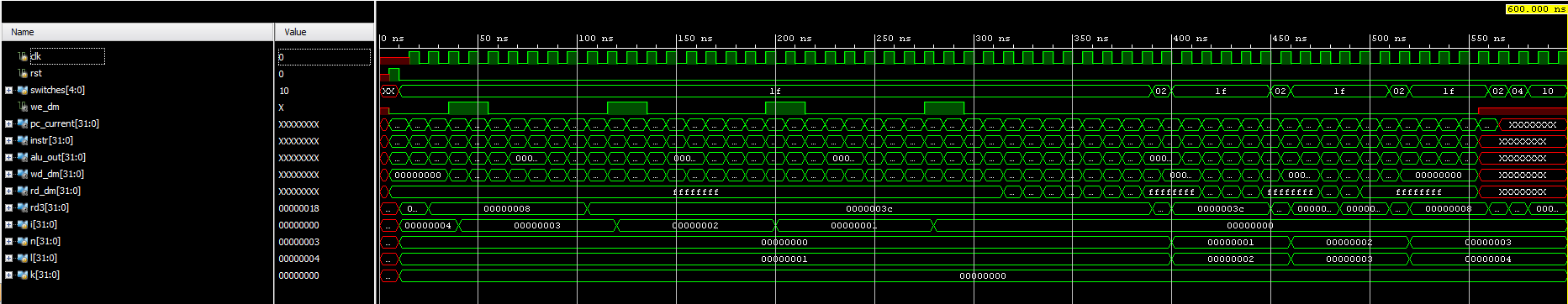


Figure 3. Processor Testbench Validation

***FPGA Test Plan***

The hardware implementation of this design was performed on the Nexys4 DDR FPGA board. The 7-Segment LEDs were used to output the contents of various registers determined by the DIP-switches below them. For validation purposes of this lab, we were interested in the contents of the wd\_dm register which was signified by writing a hex value of 10 to the switches.

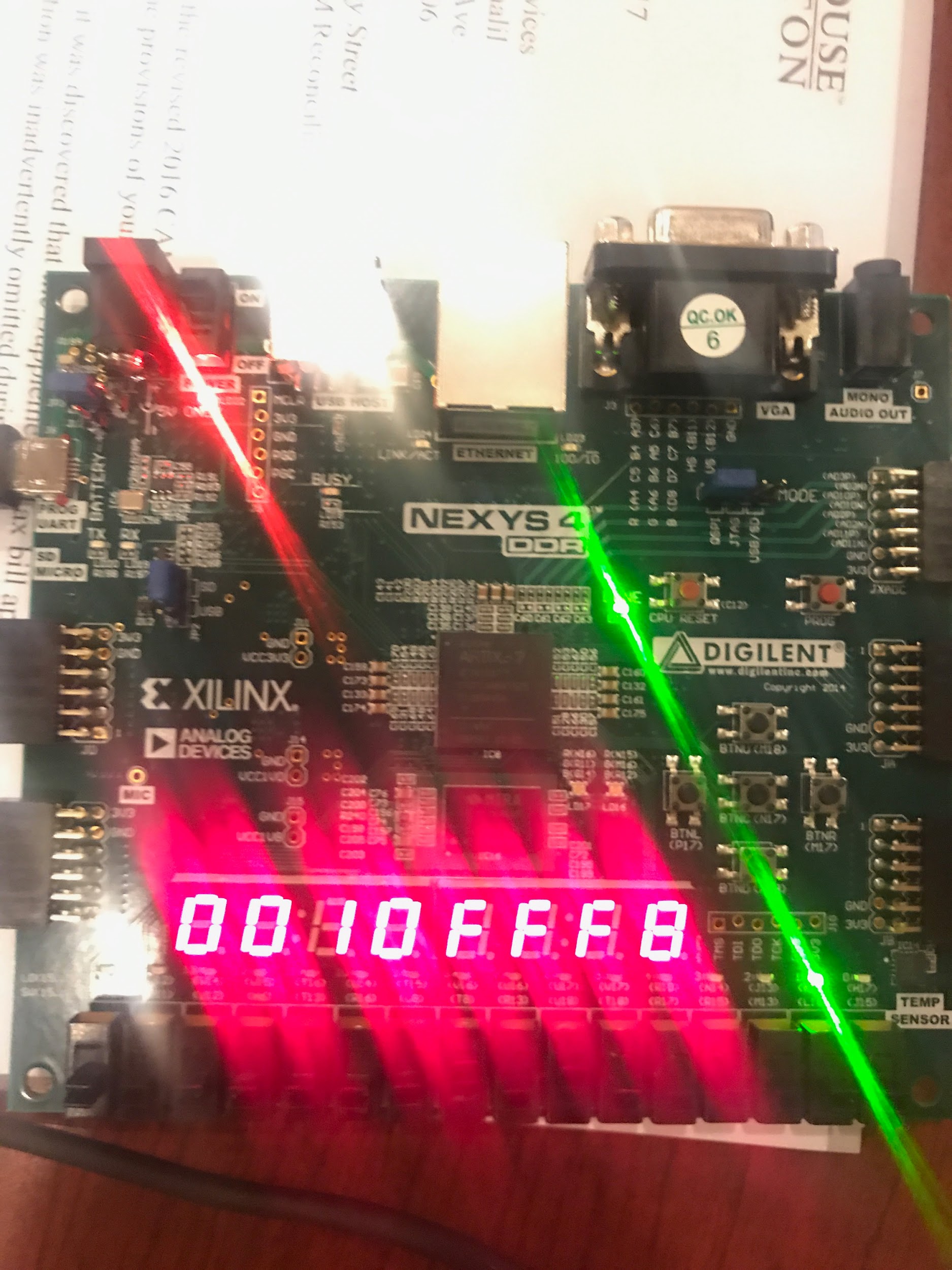


Figure 4. Processor Hardware Verification

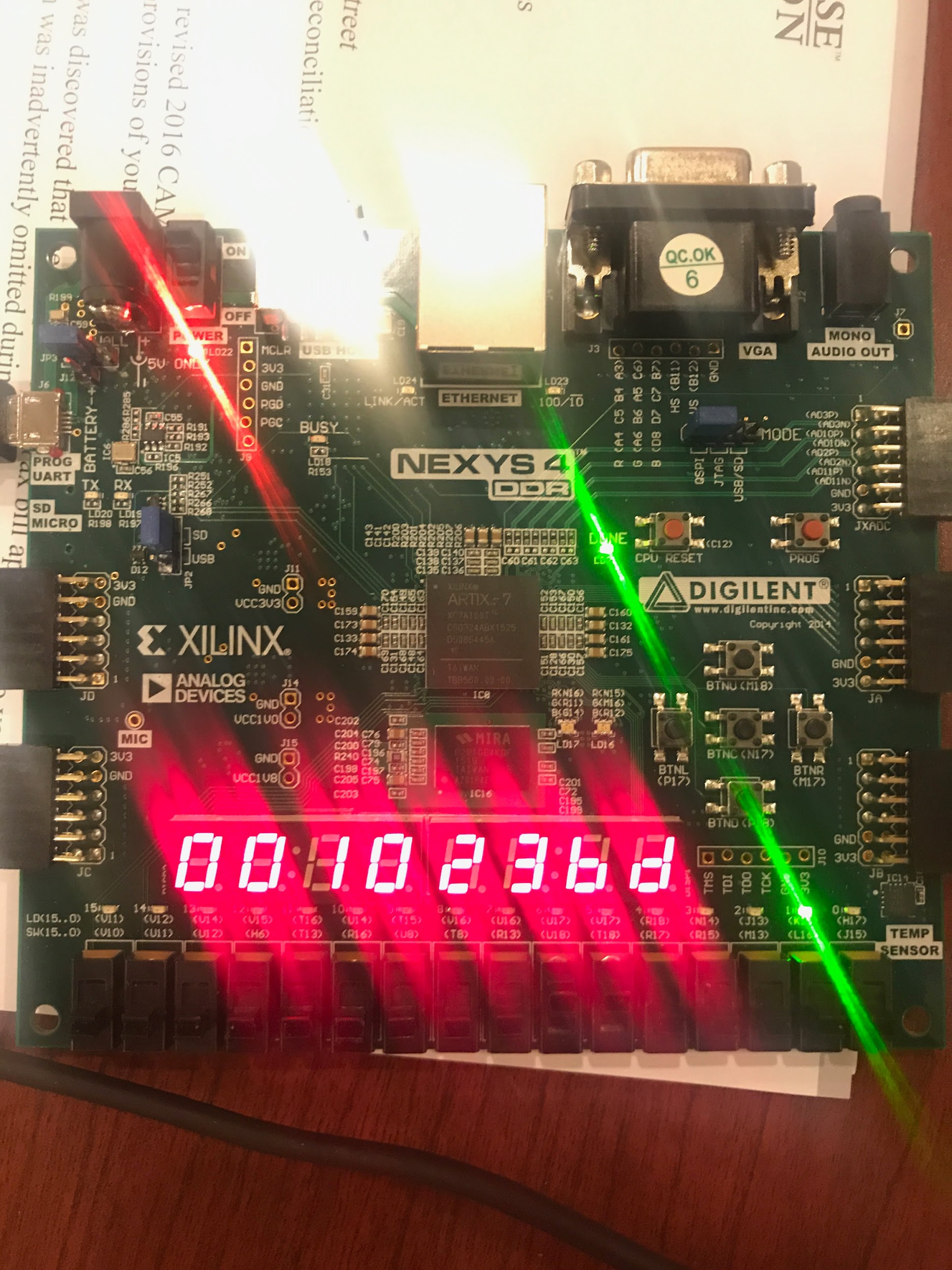


Figure 5. Extended Processor Hardware Verification

***Conclusion***

In summary, this lab was meant to familiarize the students with microprocessor design. We were provided a single cycle MIPS processor and tasked with adding to the microarchitecture to allow for functionality of additional functions. Once the design was completed, it was tested through visual validation of a waveform as well as hardware implementation using a Nexys4 DDR FPGA board. The final functionality performed according to the expected results, and overall our understanding of the MIPS processor was greatly improved.

***Source Code***

***test\_code.asm***

main:

# addi $sp, $0, 48 not for SPIM

addi $a0, $0, 4 # set arg

jal factorial # compute the factorial

add $s0, $v0, $0 # move result into $s0

j end

factorial:

addi $sp, $sp, -8 # make room on stack

sw $a0, 4($sp) # store $a0

sw $ra, 0($sp) # store $ra

addi $t0, $0, 2 # $t0 = 2

slt $t0, $a0, $t0 # a <= 1 ?

beq $t0, $0, else # no - goto else

addi $v0, $0, 1 # yes - return 1

addi $sp, $sp, 8 # restore $sp

jr $ra # return

else:

addi $a0, $a0, -1 # n = n - 1

jal factorial # recursive call

lw $ra, 0($sp) # restore $ra

lw $a0, 4($sp) # restore $a0

addi $sp, $sp, 8 # restore $sp

multu $a0, $v0 # n \* factorial(n-1)

mflo $v0 # mv result into $v0

jr $ra

end:

***Mips\_fpga.xdc***

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }];

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { button }];

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { rst }];

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { switches[0] }];

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { switches[1] }];

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { switches[2] }];

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { switches[3] }];

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { switches[4] }];

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { switches[5] }];

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { switches[6] }];

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { switches[7] }];

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { we\_dm }];

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { sink\_bit }];

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }];

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }];

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }];

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }];

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[5] }];

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }];

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[7] }];

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[0] }];

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[1] }];

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[2] }];

***tb.v***

module tb\_mips\_top;

reg clk, rst;

wire we\_dm;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm;

mips\_top DUT (clk, rst, we\_dm, pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

task tick; begin #5 clk = 1; #5 clk = 0; end endtask

task rest; begin #5 rst = 1; #5 rst = 0; end endtask

initial begin

rest;

while(pc\_current != 32'h54) tick;

$display("passed");

$finish;

end

endmodule

***cu.v***

module controlunit

(input zero, [5:0] opcode, funct, output pc\_src, jump, we\_reg, alu\_src, we\_dm, JReturn, writeHilo, selJal, [1:0] dm2reg, reg\_dst, [2:0] alu\_ctrl);

wire [1:0] alu\_op;

assign pc\_src = branch & zero;

maindec md (opcode, funct, branch, jump, writeHilo, selJal, we\_reg, alu\_src, we\_dm, dm2reg, reg\_dst, alu\_op);

auxdec ad (alu\_op, funct, JReturn, alu\_ctrl);

Endmodule

***dp.v***

module datapath

(input clk, rst, pc\_src, jump, JReturn, writeHilo, selJal, we\_reg, alu\_src, [1:0] dm2reg, reg\_dst, [2:0] alu\_ctrl, [4:0] ra3, [31:0] instr, rd\_dm, output zero, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire [4:0] rf\_wa;

wire [31:0] pc\_plus4, pc\_pre, pc\_next, sext\_imm, ba, bta, jta, alu\_pa, alu\_pb, mulHi, mulLo, q\_mulHi, q\_mulLo, wd\_rf, data\_out, pc\_jump, shamt\_out, wd\_final;

wire wd\_sel;

assign wd\_sel = (alu\_ctrl == 3'b100 | alu\_ctrl == 3'b101);

assign ba = {sext\_imm[29:0], 2'b00};

assign jta = {pc\_plus4[31:28], instr[25:0], 2'b00};

// --- PC Logic --- //

dreg pc\_reg (clk, rst, pc\_next, pc\_current);

adder pc\_plus\_4 (pc\_current, 4, pc\_plus4);

adder pc\_plus\_br (pc\_plus4, ba, bta);

mux2 #(32) pc\_src\_mux (pc\_src, pc\_plus4, bta, pc\_pre);

mux2 #(32) pc\_jmp\_mux (jump, pc\_pre, jta, pc\_jump);

mux2 #(32) pc\_jr\_mux (JReturn, pc\_jump, wd\_final, pc\_next)

// --- RF Logic --- //

mux4 #(5) rf\_wa\_mux (reg\_dst, instr[20:16], instr[15:11], 31, 0, rf\_wa);

regfile rf (clk, we\_reg, instr[25:21], instr[20:16], ra3, rf\_wa, wd\_final, alu\_pa, wd\_dm, rd3);

signext se (instr[15:0], sext\_imm);

// --- ALU Logic --- //

mux2 #(32) alu\_pb\_mux (alu\_src, wd\_dm, sext\_imm, alu\_pb);

alu alu (alu\_ctrl, alu\_pa, alu\_pb, zero, alu\_out);

//--- MUL Logic ---//

mul mulOp (alu\_pa, alu\_pb, mulHi, mulLo);

en\_dreg mul\_hi\_reg (writeHilo, rst, mulHi, q\_mulHi);

en\_dreg mul\_lo\_reg (writeHilo, rst, mulLo, q\_mulLo);

// --- SHIFT Logic --- //

shifter SHAMT (alu\_ctrl, alu\_pa, instr[10:6], shamt\_out);

// --- MEM Logic --- //

mux4 #(32) rf\_data\_mux (dm2reg, alu\_out, rd\_dm, q\_mulHi, q\_mulLo, data\_out);

mux2 #(32) rf\_wd\_mux (selJal, data\_out, pc\_plus4, wd\_rf);

mux2 #(32) wd\_sel\_mux (wd\_sel, wd\_rf, shamt\_out, wd\_final);

endmodule

***cu\_modules.v***

module maindec

(input [5:0] opcode, [5:0] funct, output branch, jump, writeHilo, setJal, we\_reg, alu\_src, we\_dm, [1:0] dm2reg, reg\_dst, alu\_op);

reg [12:0] ctrl;

assign {branch, jump, reg\_dst, we\_reg, alu\_src, we\_dm, dm2reg, alu\_op, writeHilo, setJal} = ctrl;

always @ (opcode, funct)

begin

case (opcode)

6'b00\_0000:

begin

if(funct == 6'b00\_1000) //JumpReturn

ctrl = 13'b0\_0\_01\_0\_1\_0\_00\_10\_0\_0;

else if(funct == 6'b01\_1001)

ctrl = 13'b0\_0\_01\_0\_0\_0\_00\_10\_1\_0; //MultU

else if(funct == 6'b01\_0000)

ctrl = 13'b0\_0\_01\_1\_0\_0\_10\_10\_0\_0; //MfHi

else if(funct == 6'b01\_0010)

ctrl = 13'b0\_0\_01\_1\_0\_0\_11\_10\_0\_0; //MfLo

else

ctrl = 13'b0\_0\_01\_1\_0\_0\_00\_10\_0\_0; // R-type

end

6'b00\_1000: ctrl = 13'b0\_0\_00\_1\_1\_0\_00\_00\_0\_0; // ADDI

6'b00\_0100: ctrl = 13'b1\_0\_00\_0\_0\_0\_00\_01\_0\_0; // BEQ

6'b00\_0010: ctrl = 13'b0\_1\_00\_0\_0\_0\_00\_00\_0\_0; // J

6'b10\_1011: ctrl = 13'b0\_0\_00\_0\_1\_1\_00\_00\_0\_0; // SW

6'b10\_0011: ctrl = 13'b0\_0\_00\_1\_1\_0\_01\_00\_0\_0; // LW

6'b00\_0011: ctrl = 13'b0\_1\_10\_1\_0\_0\_00\_00\_0\_1; //JAL

default: ctrl = 13'bx\_x\_xx\_x\_x\_x\_xx\_xx\_x\_x;

endcase

end

endmodule

module auxdec

(input [1:0] alu\_op, [5:0] funct, output Jreturn, [2:0] alu\_ctrl);

reg [2:0] ctrl;

reg jReg;

assign {alu\_ctrl} = ctrl;

assign {Jreturn} = jReg;

always @ (alu\_op, funct)

begin

case (alu\_op)

2'b00:

begin

ctrl = 3'b010;

jReg = 0;

end

2'b01:

begin

ctrl = 3'b110;

jReg = 0;

end

default: case (funct)

6'b10\_0100:

begin

ctrl = 3'b000;

jReg = 0;

end

6'b10\_0101:

begin

ctrl = 3'b001;

jReg = 0;

end

6'b10\_0000:

begin

jReg = 0;

ctrl = 3'b010;

end

6'b10\_0010:

begin

jReg = 0;

ctrl = 3'b110;

end

6'b10\_1010:

begin

jReg = 0;

ctrl = 3'b111;

end

6'b00\_1000:

begin

jReg = 1;

ctrl = 3'b011;

end

6'b01\_1001:

begin

jReg = 0;

ctrl = 3'b000;

end

6'b01\_0000:

begin

jReg = 0;

ctrl = 3'b000;

end

6'b01\_0010:

begin

jReg = 0;

ctrl = 3'b000;

end

6'b00\_0000:

begin

jReg = 0;

ctrl = 3'b100;

end

6'b00\_0010:

begin

jReg = 0;

ctrl = 3'b101;

end

default: ctrl = 3'bxxx;

endcase

endcase

end

endmodule

***Memfile.dat***

20040004

0C000004

00408020

08000015

23BDFFF8

AFA40004

AFBF0000

20080002

0088402A

10080003

20020001

23BD0008

03E00008

2084FFFF

0C000004

8FBF0000

8FA40004

23BD0008

00820019

00001012

03E00008

module mux2 #(parameter wide = 8)

(input sel, [wide-1:0] a, b, output [wide-1:0] y);

assign y = (sel) ? b : a;

endmodule

module mux4 #(parameter wide = 8)

(input [1:0] sel, [wide-1:0] a, b, c, d, output reg [wide-1:0] y);

always@ (sel, a, b, c, d)

begin

case(sel)

2'b00: y = a;

2'b01: y = b;

2'b10: y = c;

2'b11: y = d;

endcase

end

endmodule

module adder

(input [31:0] a, b, output [31:0] y);

assign y = a + b;

endmodule

module signext

(input [15:0] a, output [31:0] y);

assign y = {{16{a[15]}}, a};

endmodule

module alu

(input [2:0] op, [31:0] a, b, output zero, reg [31:0] y);

assign zero = (y == 0);

always @ (op, a, b)

begin

case (op)

3'b000: y = a & b;

3'b001: y = a | b;

3'b010: y = a + b;

3'b011: y = a + 0;

3'b110: y = a - b;

3'b111: y = (a < b) ? 1 : 0;

endcase

end

endmodule

module mul

(input [31:0] x, y, output reg [31:0] outHi, outLo);

reg [63:0] out;

always @ (x, y)

begin

out = x \* y;

outHi = out[63:32];

outLo = out[31:0];

end

endmodule

module shifter #(parameter DATA\_WIDTH = 32) (

input wire [2:0] enable\_op,

input wire [DATA\_WIDTH - 1:0] rd, shamt,

output reg [DATA\_WIDTH - 1:0] data\_out

);

always @(\*) begin

case (enable\_op)

3'b100: data\_out = rd << shamt;

3'b101: data\_out = rd >> shamt;

default: data\_out = data\_out;

endcase

end

endmodule

module dreg

(input clk, rst, [31:0] d, output reg [31:0] q);

always @ (posedge clk, posedge rst)

begin

if (rst) q <= 0;

else q <= d;

end

endmodule

***dp\_modules.v***

module en\_dreg

(input en, rst, [31:0] d, output reg [31:0] q);

always @ (d, en, rst)

begin

if (rst) q <= 0;

else if(en)

q <= d;

else q <= q;

end

endmodule

module regfile

(input clk, we, [4:0] ra1, ra2, ra3, wa, [31:0] wd, output [31:0] rd1, rd2, rd3);

reg [31:0] rf [0:31];

integer n;

initial begin

for (n = 0; n < 32; n = n + 1) rf[n] = 32'h0;

end

always @ (posedge clk)

begin

if (we) rf[wa] <= wd;

end

assign rd1 = (ra1 == 0) ? 0 : rf[ra1];

assign rd2 = (ra2 == 0) ? 0 : rf[ra2];

assign rd3 = (ra3 == 0) ? 0 : rf[ra3];

endmodule

***Mips.v***

module mips\_top

(input clk, rst, output we\_dm, [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm);

wire [31:0] DONT\_USE;

mips mips (clk, rst, 0, instr, rd\_dm, we\_dm, pc\_current, alu\_out, wd\_dm, DONT\_USE);

imem imem (pc\_current[7:2], instr);

dmem dmem (clk, we\_dm, alu\_out[7:2], wd\_dm, rd\_dm);

endmodule

module mips

(input clk, rst, [4:0] ra3, [31:0] instr, rd\_dm, output we\_dm, [31:0] pc\_current, alu\_out, wd\_dm, rd3);

wire pc\_src, jump, we\_reg, alu\_src, JReturn, writeHilo, setJal;

wire [2:0] alu\_ctrl;

wire [1:0] reg\_dst, dm2reg;

datapath dp (clk, rst, pc\_src, jump, JReturn, writeHilo, setJal, we\_reg, alu\_src, dm2reg, reg\_dst, alu\_ctrl, ra3, instr, rd\_dm, zero, pc\_current, alu\_out, wd\_dm, rd3);

controlunit cu (zero, instr[31:26], instr[5:0], pc\_src, jump, we\_reg, alu\_src, we\_dm, JReturn, writeHilo, setJal, dm2reg, reg\_dst, alu\_ctrl);

endmodule

module mips\_fpga

(input clk, rst, button, [7:0] switches, output we\_dm, sink\_bit, [7:0] LEDSEL, LEDOUT);

reg [15:0] reg\_hex;

wire clk\_sec, clk\_5KHz, clk\_pb;

wire [7:0] digit0, digit1, digit2, digit3, digit4, digit5, digit6, digit7;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm, dispData;

clk\_gen clk\_gen (clk, rst, clk\_sec, clk\_5KHz);

bdebouncer bd (clk\_5KHz, button, clk\_pb);

mips mips (clk\_pb, rst, switches[4:0], instr, rd\_dm, we\_dm, pc\_current, alu\_out, wd\_dm, dispData);

/\*

switchs[4:0] are used as the 3rd read address (ra3) of the RF,

dispData is the register contents from the RF's 3rd read port (rd3).

\*/

imem imem (pc\_current[7:2], instr);

dmem dmem (clk\_pb, we\_dm, alu\_out[7:2], wd\_dm, rd\_dm);

bcd\_to\_7seg bcd7 (pc\_current[15:12], digit7);

bcd\_to\_7seg bcd6 (pc\_current[11:8], digit6);

bcd\_to\_7seg bcd5 (pc\_current[7:4], digit5);

bcd\_to\_7seg bcd4 (pc\_current[3:0], digit4);

bcd\_to\_7seg bcd3 (reg\_hex[15:12], digit3);

bcd\_to\_7seg bcd2 (reg\_hex[11:8], digit2);

bcd\_to\_7seg bcd1 (reg\_hex[7:4], digit1);

bcd\_to\_7seg bcd0 (reg\_hex[3:0], digit0);

led\_mux led\_mux (clk\_5KHz, rst, digit7, digit6, digit5, digit4, digit3, digit2, digit1, digit0, LEDSEL, LEDOUT);

/\*

switches[7:5] = 000 : Display lower half word of register selected by switches[4:0]

switches[7:5] = 001 : Display higher half word of register selected by switches[4:0]

switches[7:5] = 010 : Display lower half word of 'instr'

switches[7:5] = 011 : Display higher half word of 'instr'

switches[7:5] = 100 : Display lower half word of 'alu\_out'

switches[7:5] = 101 : Display higher half word of 'alu\_out'

switches[7:5] = 110 : Display lower half word of 'wd\_dm'

switches[7:5] = 111 : Display higher half word of 'wd\_dm'

\*/

always @ (posedge clk)

begin

case ({switches[7], switches[6], switches[5]})

3'b000: reg\_hex = dispData[15:0];

3'b001: reg\_hex = dispData[31:16];

3'b010: reg\_hex = instr[15:0];

3'b011: reg\_hex = instr[31:16];

3'b100: reg\_hex = alu\_out[15:0];

3'b101: reg\_hex = alu\_out[31:16];

3'b110: reg\_hex = wd\_dm[15:0];

3'b111: reg\_hex = wd\_dm[31:16];

endcase

end

assign sink\_bit = (pc\_current > 0) ^ (instr > 0) ^ (alu\_out > 0);

endmodule