

1. **Design Methodology**

This lab used the initial design of a MIPS processor provided to each group. Multiple extensions were added to support the functionality of these instructions: MULTU, MFHI, MFLO, JR, JAL. The original set of instructions include: add, sub, and, or , slt, lw, sw, bew, j, addi. The new single-cycle processor has been enhanced to allow additional functionality. A new 32-bit machine code memfile was used. The processor has been verified using a self-testing testbench for software verification. Hardware validation was also done using the Nexys4 FPGA. Below are provided figures of both software and hardware validation, including a control unit output table, and figures for the enhanced datapath and control unit. The learning outcome for this lab was to validate the MIPS processor design using the FPGA board. This allows the cpu to undergo software and hardware validation.

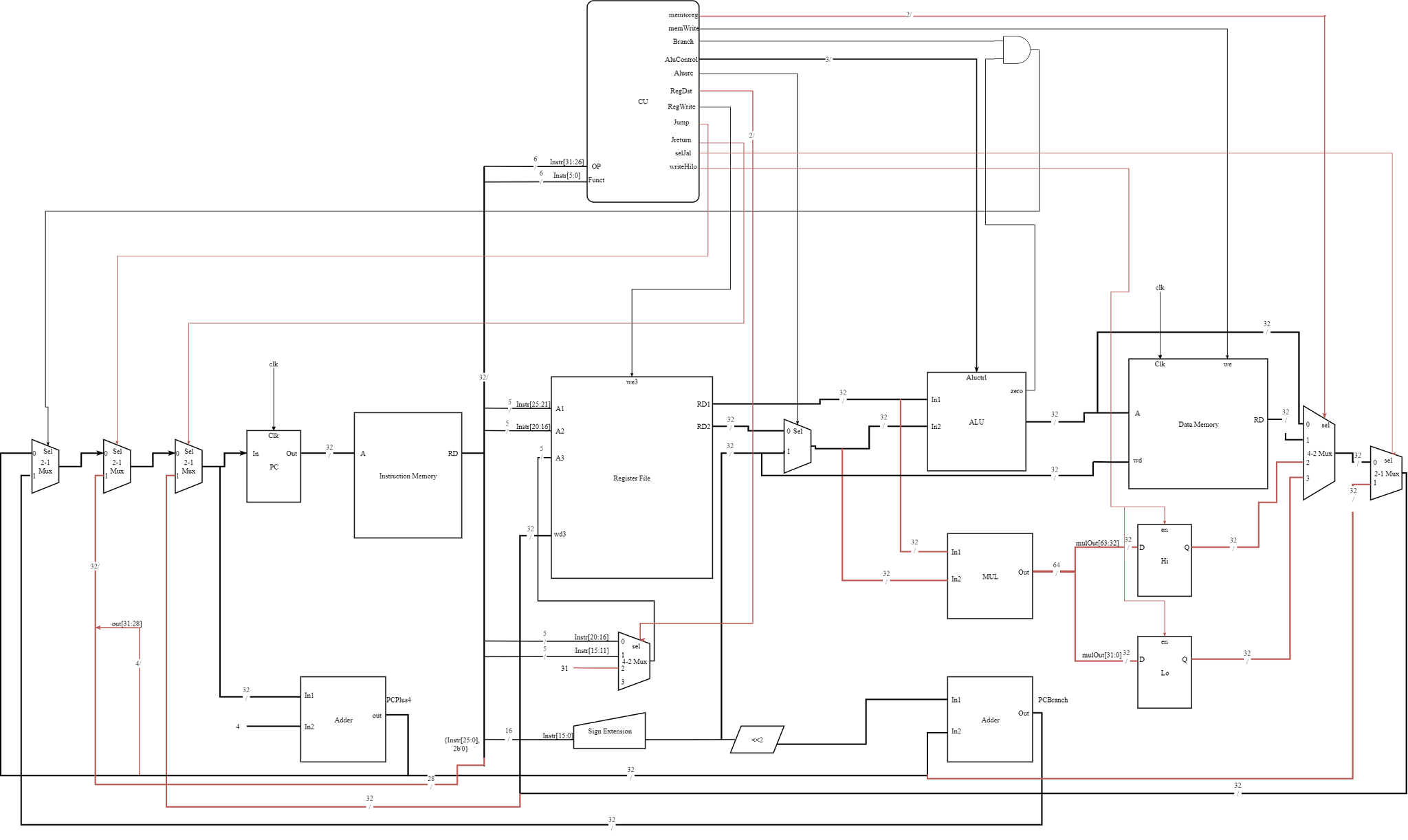
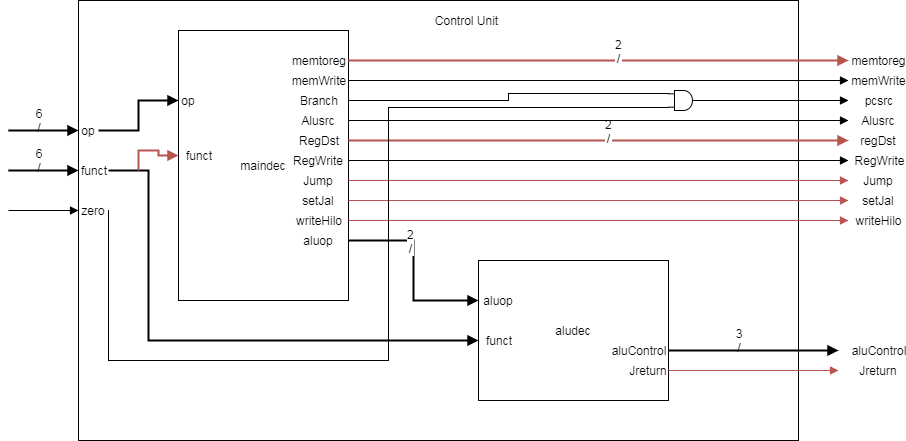


Figure 1. Extended Datapath and Control Unit

Figure 2. Control Unit for Extended MIPS Processor

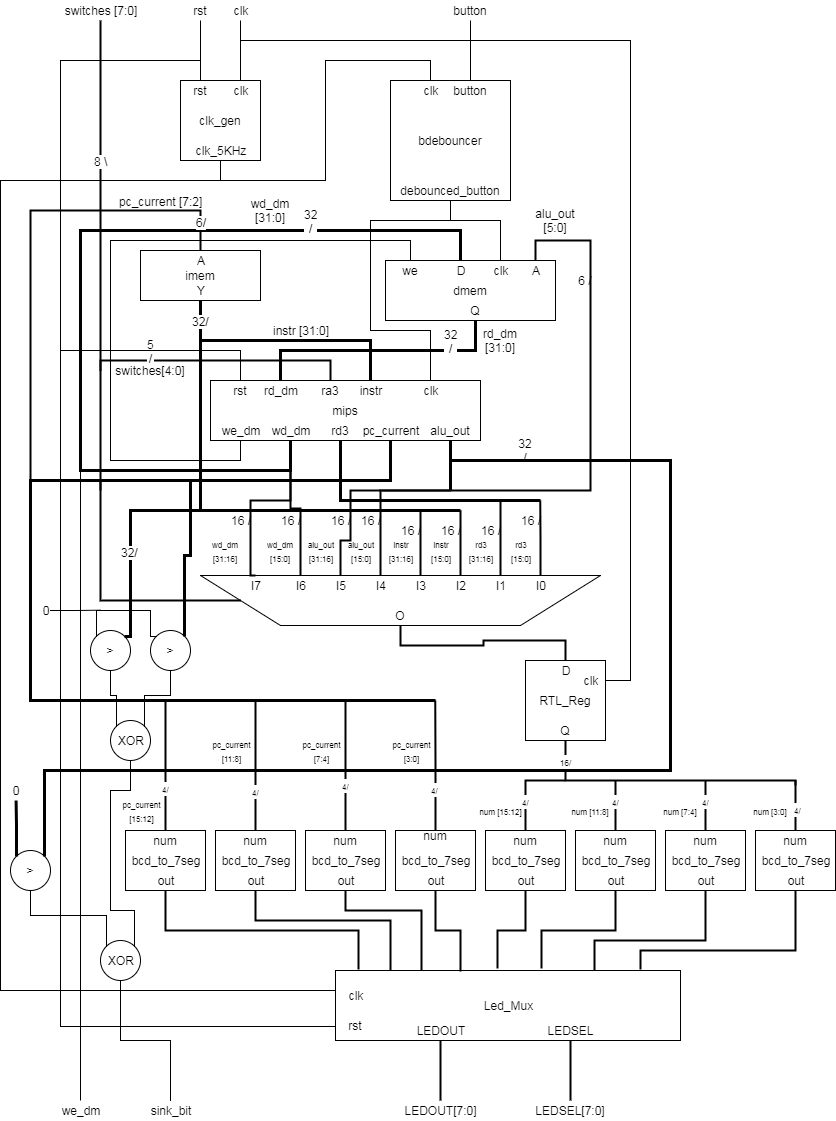


Figure 3. Design for the FPGA Extended MIPS Processor

**II. 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

Table 1. Control Unit Output

|  | 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 |

**III. Test Plan for Simulation**

The simulation was used to test the test bench file that was created. The test bench file test the inputs and outputs of the mips assembler. We specifically checked the functionality of the mips assembler when the functions J, JAL, JR, MULTU, MFLO, and MFHI. Also, we compared the values of the register file in the created mips assembler against the values found in the MARS MIPS assembler.

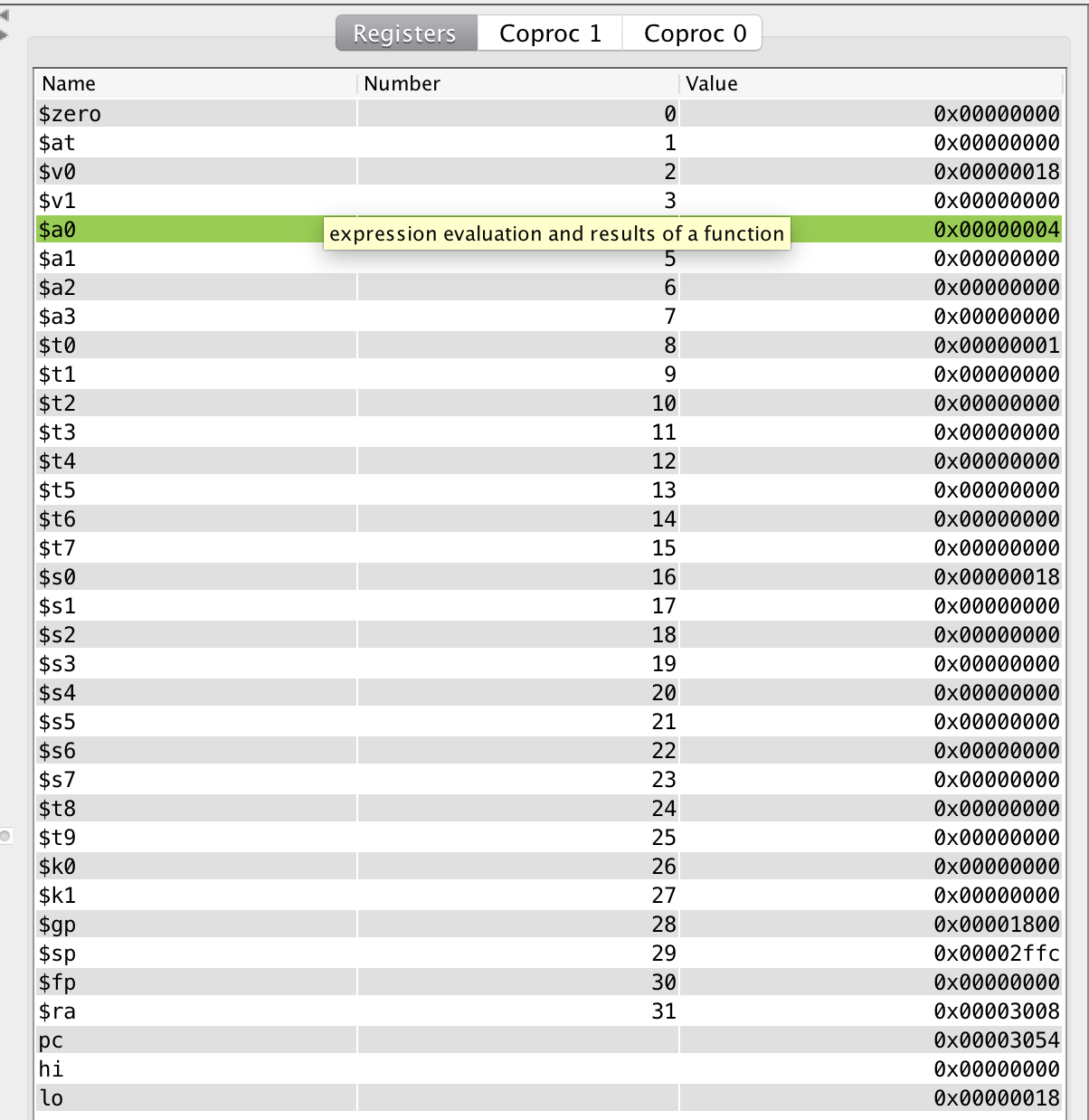


Figure 4. Register Output of Extended Test Code

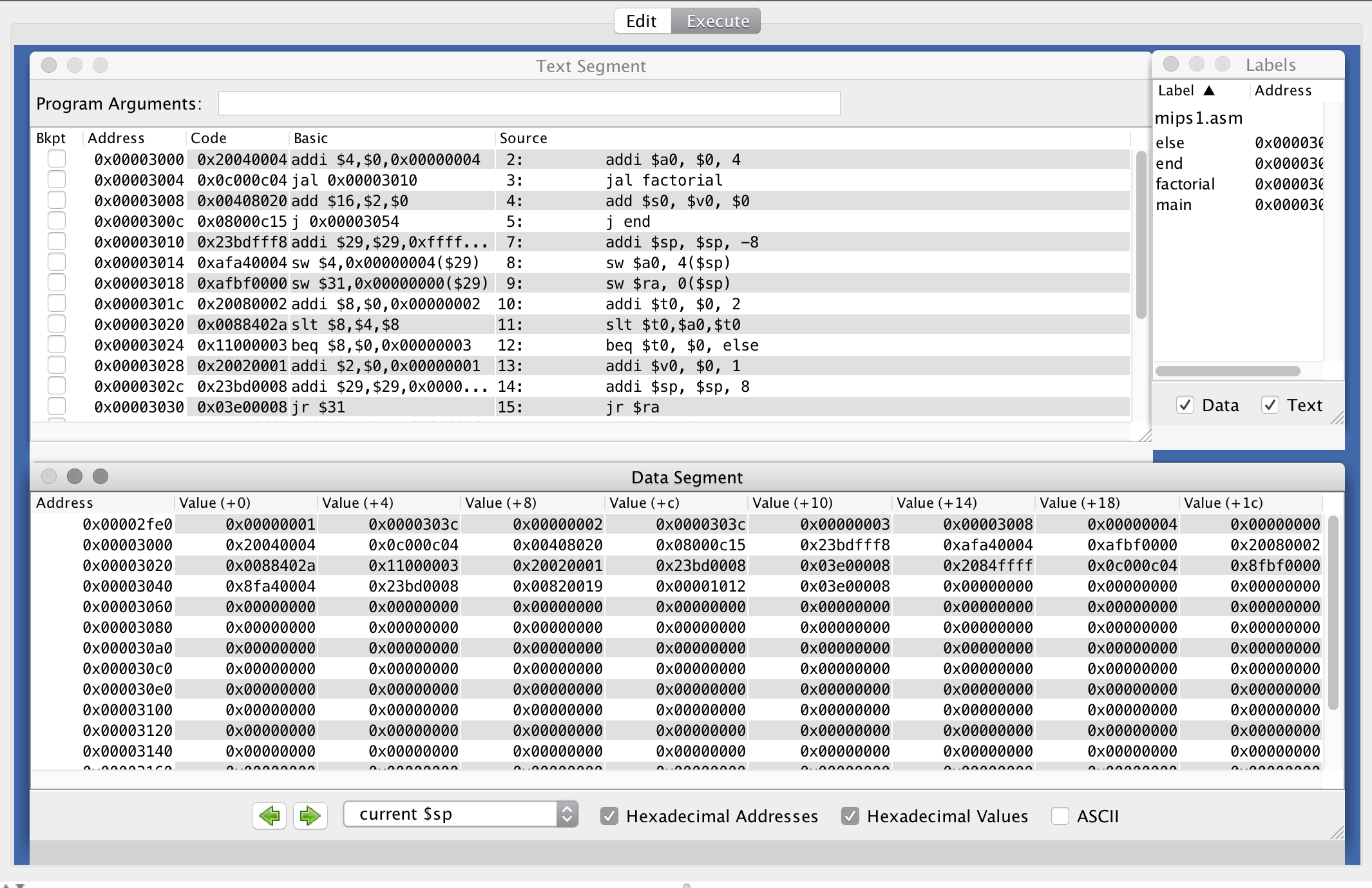


Figure 5. Extended Test Code MARS Simulation

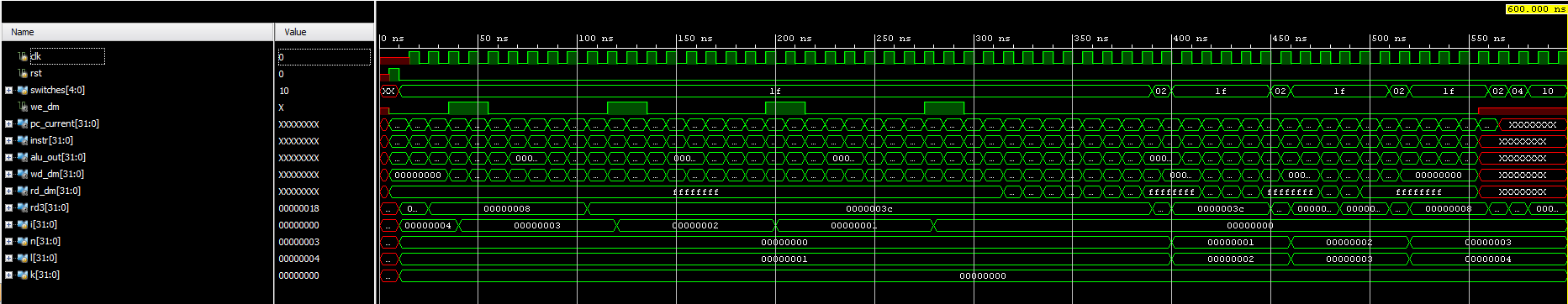


Figure 6. Extended Processor Testbench Validation

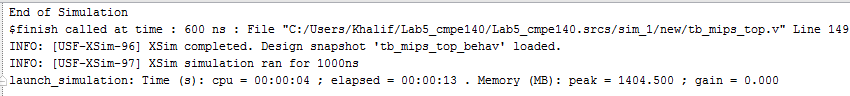


Figure 7. Extended Processor Software Verification

The console output shows that no errors occurred when testing the modules, within the test bench we had error message that would appear if an error occurred.

**IV. Test Plan For FPGA Board**

The test plan for the enhanced single-cycle by using an eight bit DIP-switch and a button debouncer to control the clock. The eight bit DIP-switch was used to view the content value on the first four LED lights of a specific register in the register file. The last four LED lights displayed the program count. The figures below displayed the contents of the wd\_dm output at instruction located at 10-hex loaded onto the Nexys4 DDR FPGA board.

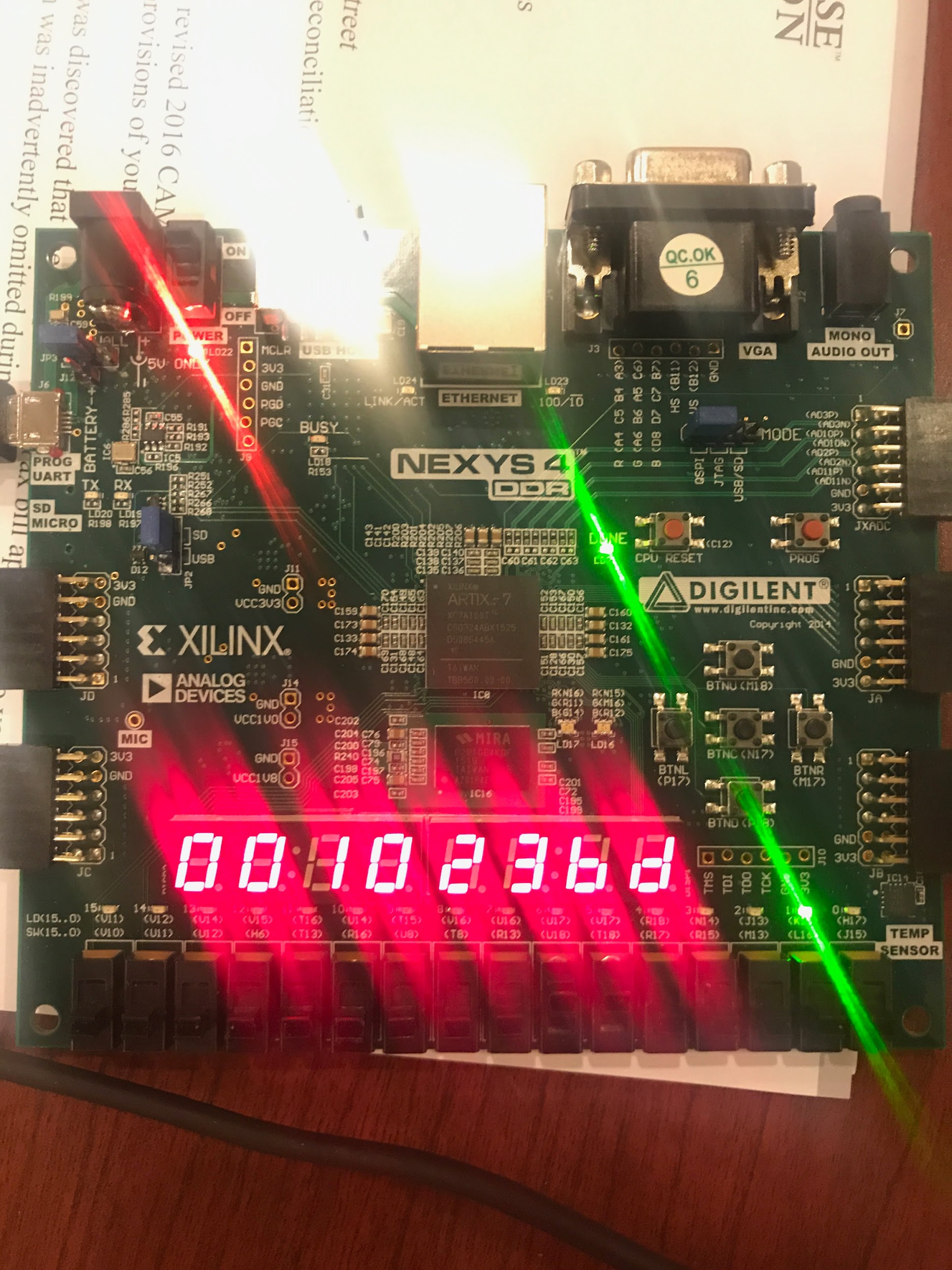


Figure 8. Extended Processor Hardware Verification

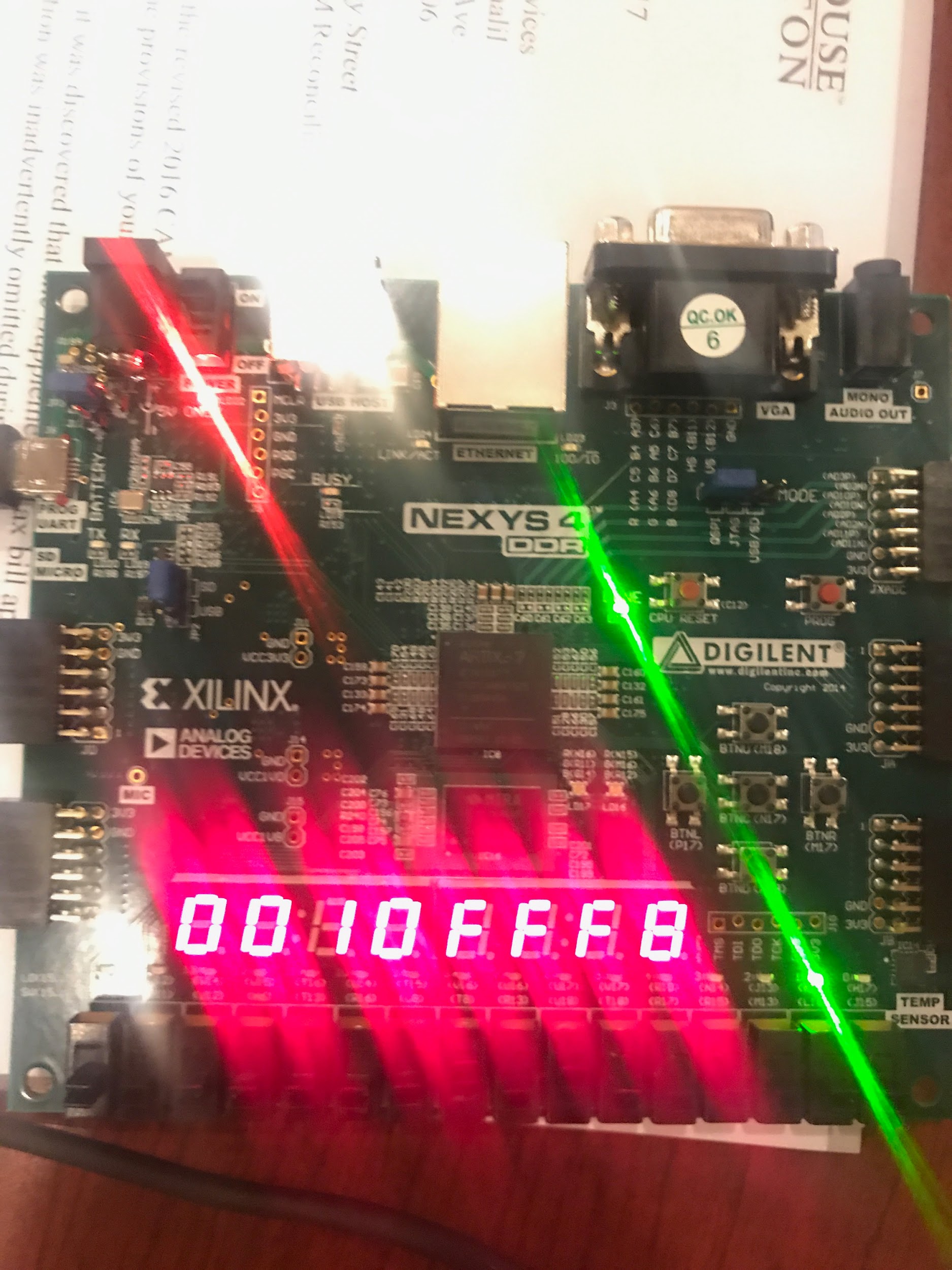


Figure 9. Extended Processor Hardware Verification

**V. Conclusion**

This lab was completed successfully and gave us more exposure to the mips assembler architecture. We completed several tasks that included designing a custom official draft of the extended datapath and control unit, a new control unit truth table, and a design and verification of the MIPS processor extension design. The first task involved understanding the mips assembler architecture and the added assembly instructions. The second task involved following that design and programming it into the datapath and the control unit. The third task involved verifying the design using a testbench and programming a FPGA board. In turn, my partner and I became more familiar with the architecture and the functionality of the single-cycle MIPS processor during this lab.

***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:

***Memfile.dat***

20040004

0C000004

00408020

08000015

23BDFFF8

AFA40004

AFBF0000

20080002

0088402A

10080003

20020001

23BD0008

03E00008

2084FFFF

0C000004

8FBF0000

8FA40004

23BD0008

00820019

00001012

03E00008

***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] }];

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[3] }];

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[4] }];

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[5] }];

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[6] }];

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[7] }];

***Controlunit.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

***Cu\_parts.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; // add

jReg = 0;

end

2'b01:

begin

ctrl = 3'b110; // sub

jReg = 0;

end

default: case (funct)

6'b10\_0100:

begin

ctrl = 3'b000; // AND

jReg = 0;

end

6'b10\_0101:

begin

ctrl = 3'b001; // OR

jReg = 0;

end

6'b10\_0000:

begin

jReg = 0;

ctrl = 3'b010; // ADD

end

6'b10\_0010:

begin

jReg = 0;

ctrl = 3'b110; // SUB

end

6'b10\_1010:

begin

jReg = 0;

ctrl = 3'b111; // SLT

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

default: ctrl = 3'bxxx;

endcase

endcase

end

endmodule

***Datapath.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;

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\_rf, 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\_rf, 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);

// --- 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);

endmodule

***Dp\_parts.v***

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 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

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

***Mem\_parts.v***

module imem

(input [5:0] a, output [31:0] y);

reg [31:0] rom [0:63];

initial begin

$readmemh ("memfile.dat", rom);

end

assign y = rom[a];

endmodule

module dmem

(input clk, we, [5:0] a, [31:0] d, output [31:0] q);

reg [31:0] ram [0:63];

integer n;

initial begin

for (n = 0; n < 64; n = n + 1) ram[n] = 32'hFFFFFFFF;

end

always @ (posedge clk)

begin

if (we) ram[a] <= d;

end

assign q = ram[a];

endmodule

***Mips.v***

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

***Mips\_fpga.v***

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

***Mips\_top.v***

module mips\_top

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

wire [31:0] DONT\_USE;

mips mips (clk, rst, switches, 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

***Utility.v***

module clk\_gen

(input clk100MHz, rst, output reg clk\_sec, clk\_5KHz);

integer count1, count2;

always @ (posedge clk100MHz)

begin

if (rst)

begin

count1 = 0; clk\_sec = 0;

count2 = 0; clk\_5KHz = 0;

end

else

begin

if (count1 == 50000000)

begin

clk\_sec = ~clk\_sec;

count1 = 0;

end

if (count2 == 10000)

begin

clk\_5KHz = ~clk\_5KHz;

count2 = 0;

end

count1 = count1 + 1;

count2 = count2 + 1;

end

end

endmodule

/\* 7-segment values \*/

`define D0 8'b10001000 /\* 0 \*/

`define D1 8'b11101101 /\* 1 \*/

`define D2 8'b10100010 /\* 2 \*/

`define D3 8'b10100100 /\* 3 \*/

`define D4 8'b11000101 /\* 4 \*/

`define D5 8'b10010100 /\* 5 \*/

`define D6 8'b10010000 /\* 6 \*/

`define D7 8'b10101101 /\* 7 \*/

`define D8 8'b10000000 /\* 8 \*/

`define D9 8'b10000100 /\* 9 \*/

`define DA 8'b10100000 /\* A \*/

`define DB 8'b11010000 /\* B \*/

`define DC 8'b11110010 /\* C \*/

`define DD 8'b11100000 /\* D \*/

`define DE 8'b10010010 /\* E \*/

`define DF 8'b10010011 /\* F \*/

`define DX 8'b01111111 /\* All segments off except decimal point \*/

module bcd\_to\_7seg

(input [3:0] num, output reg [7:0] out);

always @ (num)

begin

case (num)

4'h0: begin out=`D0; end

4'h1: begin out=`D1; end

4'h2: begin out=`D2; end

4'h3: begin out=`D3; end

4'h4: begin out=`D4; end

4'h5: begin out=`D5; end

4'h6: begin out=`D6; end

4'h7: begin out=`D7; end

4'h8: begin out=`D8; end

4'h9: begin out=`D9; end

4'hA: begin out=`DA; end

4'hB: begin out=`DB; end

4'hC: begin out=`DC; end

4'hD: begin out=`DD; end

4'hE: begin out=`DE; end

4'hF: begin out=`DF; end

default: begin out=`DX; end

endcase

end

endmodule

module led\_mux

(input clk, rst, [7:0] LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7, output [7:0] LEDSEL, LEDOUT);

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDSEL, LEDOUT} = led\_ctrl;

always@(posedge clk)

begin

index <= (rst) ? 3'd0 : (index + 3'd1);

end

always @(index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)

begin

case(index)

3'd0: led\_ctrl <= {8'b11111110, LED7};

3'd1: led\_ctrl <= {8'b11111101, LED6};

3'd2: led\_ctrl <= {8'b11111011, LED5};

3'd3: led\_ctrl <= {8'b11110111, LED4};

3'd4: led\_ctrl <= {8'b11101111, LED3};

3'd5: led\_ctrl <= {8'b11011111, LED2};

3'd6: led\_ctrl <= {8'b10111111, LED1};

3'd7: led\_ctrl <= {8'b01111111, LED0};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

module bdebouncer #(parameter depth = 16)

(input clk, button, output reg debounced\_button);

localparam history\_max = (2\*\*depth)-1;

reg [depth-1:0] history;

always @ (posedge clk)

begin

history <= {button, history[depth-1:1]};

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule

***tb\_mips\_top.v***

module tb\_mips\_top;

reg clk, rst;

reg [4:0] switches;

wire we\_dm;

wire [31:0] pc\_current, instr, alu\_out, wd\_dm, rd\_dm, rd3;

integer i;

integer n;

integer l;

integer k;

mips\_top DUT (clk, rst, switches, we\_dm, pc\_current, instr, alu\_out, wd\_dm, rd\_dm, rd3);

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;

switches = 5'b0\_0010;

i = 4;

n = 0;

k = 0;

l = 1;

while(pc\_current != 32'h54)

begin

case(pc\_current)

32'h0:

begin

switches = 5'b1\_1111;

if(instr != 32'h20040004) $display("Error with instruction at addr: 0");

end

32'h4:

begin

if(instr != 32'h0C000004) $display("Error with instruction at addr: 4");

if(alu\_out != 32'h0000) $display("Error with JAL");

end

32'h8: if(instr != 32'h00408020) $display("Error with instruction at addr: 8");

32'hC: if(instr != 32'h08000015) $display("Error with instruction at addr: C");

32'h10: if(instr != 32'h23BDFFF8) $display("Error with instruction at addr: 10");

32'h14:

begin

if(instr != 32'hAFA40004) $display("Error with instruction at addr: 14");

if(i != wd\_dm) $display("Error of Save word");

i = i - 1;

end

32'h18:

begin

if(instr != 32'hAFBF0000) $display("Error with instruction at addr: 18");

if(i == 3 && wd\_dm != 8) $display("Error with Save Word at addr: 18");

if(i < 3 && wd\_dm != 60) $display("Error with Save Word at addr: 18");

end

32'h1C: if(instr != 32'h20080002) $display("Error with instruction at addr: 1C");

32'h20: if(instr != 32'h0088402A) $display("Error with instruction at addr: 20");

32'h24: if(instr != 32'h10080003) $display("Error with instruction at addr: 24");

32'h28: if(instr != 32'h20020001) $display("Error with instruction at addr: 28");

32'h2C: if(instr != 32'h23BD0008) $display("Error with instruction at addr: 2C");

32'h30:

begin

if(instr != 32'h03E00008) $display("Error with instruction at addr: 30");

if(alu\_out != 32'h0000003C) $display("Error with Jump Return at addr: 30");

end

32'h34: if(instr != 32'h2084FFFF) $display("Error with instruction at addr: 34");

32'h38: if(instr != 32'h0C000004) $display("Error with instruction at addr: 38");

32'h3C:

begin

if(instr != 32'h8FBF0000) $display("Error with instruction at addr: 3C");

if(l < 3)

begin

if(rd\_dm != 32'h0000003C) $display("Error with loading stack position at addr: 3C");

end

if(l == 3)

begin

if(rd\_dm != 32'h8) $display("Error with loading stack position at addr: 3C");

end

end

32'h40:

begin

if(instr != 32'h8FA40004) $display("Error with instruction at addr: 40");

if(l < 3)

begin

if(rd\_dm != (l + 1)) $display("Error with loading factorial number at addr: 40");

end

if(l == 3)

begin

if(rd\_dm != 4) $display("Error with loading factorial number at addr: 40");

end

end

32'h44: if(instr != 32'h23BD0008) $display("Error with instruction at addr: 44");

32'h48:

begin

if(instr != 32'h00820019) $display("Error with instruction at addr: 48");

switches = 5'b0\_0010;

end

32'h4C:

begin

if(instr != 32'h00001012) $display("Error with instruction at addr: 4C");

if(n < 4 && n != 0)

begin

if(n \* (n + 1) != rd3) $display("Error with mul");

end

if(n == 0)

begin

if(rd3 != 1) $display("Error with mul");

end

n = n + 1;

l = l + 1;

switches = 5'b1\_1111;

end

32'h50:

begin

if(instr != 32'h03E00008) $display("Error with instruction at addr: 50");

if(n < 3 && alu\_out != 32'h0000003C) $display("Error with Jump Return at addr: 50");

if(n == 3 && alu\_out != 32'h00000008) $display("Error with Jump Return at addr: 50");

end

endcase

tick;

end

switches = 5'b0\_0010;

tick;

if(rd3 != 32'h18) $display("Error with factorial reg $2");

switches = 5'b0\_0100;

tick;

if(rd3 != 32'h4) $display("Error with factorial reg $4");

switches = 5'b1\_0000;

tick;

if(rd3 != 32'h18) $display("Error with factorial reg $16");

tick;

$display("End of Simulation");

$finish;

end

endmodule