San Jose State University
Department of Computer Engineering

CMPE 140 Lab Report

Lab 7 Report

Title Enhanced Single-cycle MIPS Processor

Semester Spring 2020 **Date** _03/27/20______

by

Name _Austin Tran___ **SID** 010645453
(typed) *(typed)*
Name William Major **SID** 009807629
(typed) *(typed)*

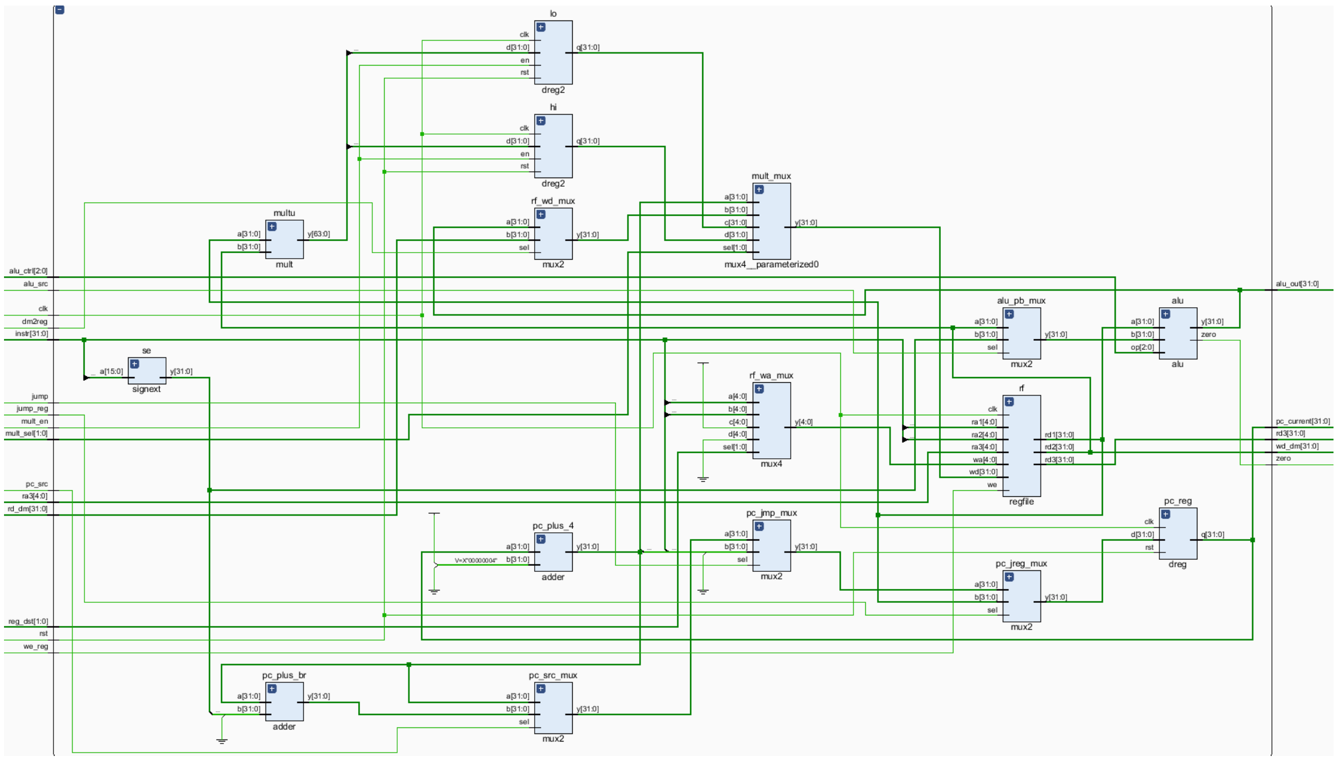
Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	A.T. W.M.	[illegible]	100%		
2		[illegible]	90%		

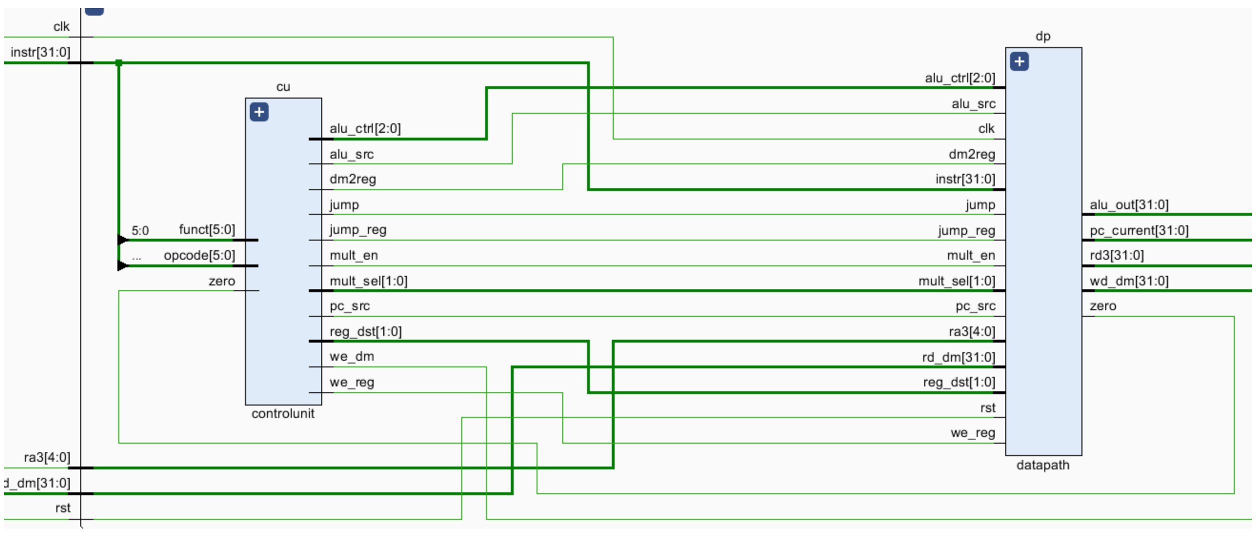
*** Detailed descriptions must be given in the report.**

Diagrams:

Data Path



Control Unit – Data Path

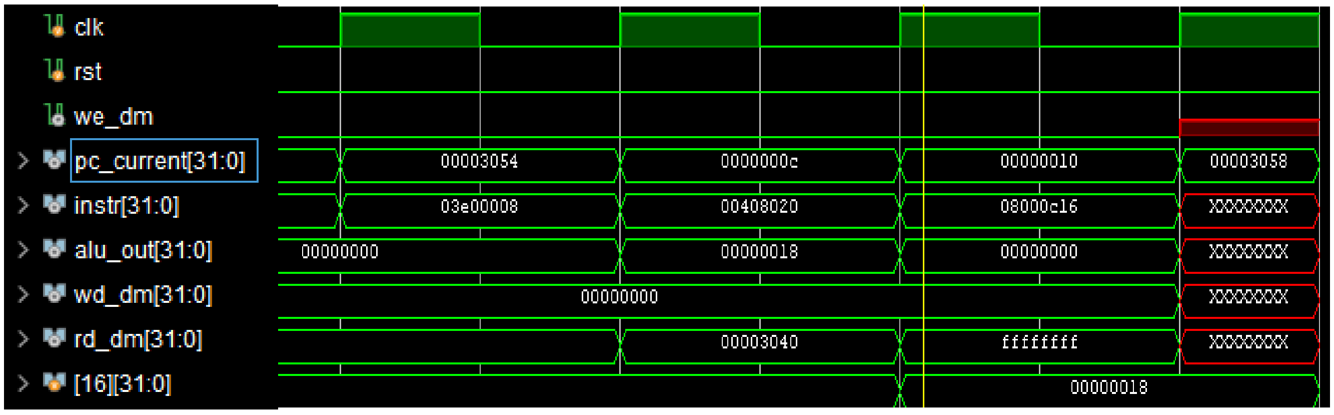


Truth Table

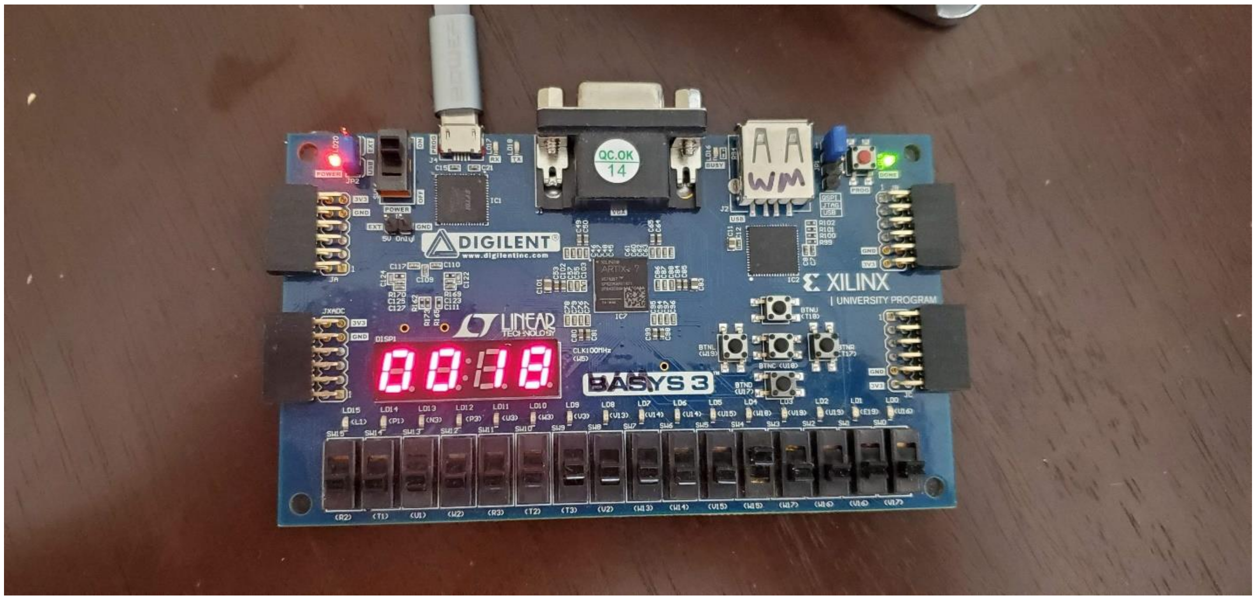
Main Dec	Branch	jump	reg_dst	we_reg	alu_src	we_dm	dm2reg	alu_op
R-Type	0	0	01	1	0	0	0	11
ADDI	0	0	00	1	1	0	0	00
BEQ	1	0	00	0	0	0	0	01
J	0	1	00	0	0	0	0	00
SW	0	0	00	0	1	1	0	00
LW	0	0	00	1	1	0	1	00
JAL	0	1	10	1	0	0	0	10

Aux Dec	alu_ctrl	multi_sel	multi_en	jump_reg
AND	000	01	0	0
OR	001	01	0	0
ADD	010	01	0	0
SUB	110	01	0	0
SLT	111	01	0	0
MULTU	000	01	1	0
MFHI	000	11	0	0
MFLO	000	10	0	0
JR	000	01	0	1

Testbench waveform



Board Validation



Source Code

```
module mips_fpga
(
               input  wire       clk,
        input  wire       rst,
        input  wire       button,
        input  wire [8:0] switches,
        output wire       we_dm,
        output wire [3:0] LEDSEL,
        output wire [7:0] LEDOUT
);
    reg  [15:0] reg_hex;
    wire        clk_sec, clk_5KHz, clk_pb;
    wire [7:0]  digit0, digit1, digit2, digit3;
    wire [31:0] pc_current, instr, alu_out, wd_dm, rd_dm, dispData;

  clk_gen clk_gen (
               .clk100MHz          (clk),
               .rst                (rst),
               .clk_4sec           (clk_sec),
               .clk_5KHz           (clk_5KHz)
       );
   button_debouncer bd (
               .clk                (clk_5KHz),
               .button             (button),
               .debounced_button   (clk_pb)
       );
       mips_top mips_top (
               .clk                          (clk),
               .rst                          (rst),
               .ra3                          (switches[4:0]),
               .instr                        (instr),
               .we_dm                        (we_dm),
               .pc_current                   (pc_current),
               .alu_out                      (alu_out),
               .wd_dm                        (wd_dm),
               .rd3                          (dispData)
       );
    hex_to_7seg hex3 (
            .HEX                (reg_hex[15:12]),
            .s                  (digit3)
        );

    hex_to_7seg hex2 (
            .HEX                (reg_hex[11:8]),
            .s                  (digit2)
        );

    hex_to_7seg hex1 (
            .HEX                (reg_hex[7:4]),
            .s                  (digit1)
        );

    hex_to_7seg hex0 (
            .HEX                (reg_hex[3:0]),
            .s                  (digit0)
        );

    led_mux led_mux (
            .clk                (clk_5KHz),
            .rst                (rst),
            .LED3               (digit3),
            .LED2               (digit2),
            .LED1               (digit1),
            .LED0               (digit0),
            .LEDSEL             (LEDSEL),
            .LEDOUT             (LEDOUT)
        );

               always @ (posedge clk) begin
        case ({switches[8:5]})
               4'b0000: reg_hex = dispData[15:0];
               4'b0001: reg_hex = dispData[31:16];
               4'b0010: reg_hex = instr[15:0];
               4'b0011: reg_hex = instr[31:16];
               4'b0100: reg_hex = alu_out[15:0];
               4'b0101: reg_hex = alu_out[31:16];
               4'b0110: reg_hex = wd_dm[15:0];
               4'b1000: reg_hex = pc_current[15:0];
               4'b1001: reg_hex = pc_current[31:16];
               default: reg_hex = pc_current[15:0];
        endcase
end
endmodule


module mips(
               input wire                   clk,
               input wire                   rst,
               input wire [4:0]      ra3,
               input wire [31:0]     instr,
               input wire [31:0]     rd_dm,
               output wire                  we_dm,
               output wire [31:0]    pc_current,
               output wire [31:0]    alu_out,
               output wire [31:0]    wd_dm,
               output wire    [31:0]  rd3
        );
    wire       pc_src, jump, we_reg, alu_src, dm2reg, mult_en, jump_reg;
    wire [1:0] mult_sel, reg_dst;
    wire [2:0] alu_ctrl;
    wire zero;
    datapath    dp (clk, mult_en, jump_reg, rst, pc_src, jump, we_reg, alu_src, dm2reg, alu_ctrl,
reg_dst, mult_sel, 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, dm2reg,
mult_en, jump_reg, reg_dst, mult_sel, alu_ctrl);
endmodule


module maindec
(input [5:0] opcode, output branch, output jump, output [1:0] reg_dst, output we_reg, output
alu_src,
output we_dm, output dm2reg, output [1:0] alu_op);
    reg [9:0] ctrl;
    assign {branch, jump, reg_dst, we_reg, alu_src, we_dm, dm2reg, alu_op} = ctrl;
    always @ (opcode)
    begin
        case (opcode)
            6'b00_0000: ctrl = 10'b0_0_01_1_0_0_0_11; // R-type
            6'b00_1000: ctrl = 10'b0_0_00_1_1_0_0_00; // ADDI
            6'b00_0100: ctrl = 10'b1_0_00_0_0_0_0_01; // BEQ
            6'b00_0010: ctrl = 10'b0_1_00_0_0_0_0_00; // J
            6'b10_1011: ctrl = 10'b0_0_00_0_1_1_0_00; // SW
            6'b10_0011: ctrl = 10'b0_0_00_1_1_0_1_00; // LW
            6'b00_0011: ctrl = 10'b0_1_10_1_0_0_0_10; // JAL
            default:    ctrl = 10'bx_x_xx_x_x_x_x_xx;
        endcase
    end
endmodule

module auxdec
(input [1:0] alu_op, input [5:0] funct, output [2:0] alu_ctrl, output [1:0] multi_sel, output
multi_en, output jump_reg);
    reg [6:0] ctrl;
    assign {alu_ctrl, multi_sel, multi_en, jump_reg} = ctrl;
    always @ (alu_op, funct)
    begin
        case (alu_op)
            2'b00: ctrl = 7'b010_01_0_0; // add
            2'b01: ctrl = 7'b110_01_0_0; // sub
            2'b10: ctrl = 7'b000_00_0_0; // jal
            default: case (funct)
                6'b10_0100: ctrl = 7'b000_01_0_0; // AND
                6'b10_0101: ctrl = 7'b001_01_0_0; // OR
                6'b10_0000: ctrl = 7'b010_01_0_0; // ADD
                6'b10_0010: ctrl = 7'b110_01_0_0; // SUB
                6'b10_1010: ctrl = 7'b111_01_0_0; // SLT
                6'b01_1001: ctrl = 7'b000_01_1_0; // MULTU
                6'b01_0000: ctrl = 7'b000_11_0_0; // MFHI
                6'b01_0010: ctrl = 7'b000_10_0_0; // MFLO
                6'b00_1000: ctrl = 7'b000_01_0_1; // JR
                default:    ctrl = 7'bxxx_xx_x_x;
            endcase
        endcase
    end
endmodule

module controlunit(
               input zero,
               input [5:0] opcode,
               input [5:0] funct,
               output pc_src,
               output jump,
               output we_reg,
               output alu_src,
               output we_dm,
               output dm2reg,
               output mult_en,
               output jump_reg,
               output [1:0] reg_dst,
               output [1:0] mult_sel,
               output [2:0] alu_ctrl
       );
    wire [1:0] alu_op;
    wire branch;

    maindec md (opcode, branch, jump, reg_dst, we_reg, alu_src, we_dm, dm2reg, alu_op);

    auxdec  ad (alu_op, funct, alu_ctrl, mult_sel, mult_en, jump_reg);

    assign pc_src = branch & zero;

endmodule

module datapath(
               input wire clk,
               input wire mult_en,
               input wire jump_reg,
               input wire rst,
               input wire pc_src,
               input wire jump,
               input wire we_reg,
               input wire alu_src,
               input wire dm2reg,
               input wire [2:0] alu_ctrl,
               input wire [1:0] reg_dst,
               input wire [1:0] mult_sel,
               input wire [4:0] ra3,
               input wire [31:0] instr,
               input wire [31:0] rd_dm,
               output wire zero,
               output wire [31:0] pc_current,
               output wire [31:0] alu_out,
               output wire [31:0] wd_dm,
               output wire    [31:0] rd3
       );
    wire [4:0]  rf_wa;
    wire [31:0] pc_plus4, pc_pre, pc_next, sext_imm, ba, bta, jta, alu_pa, alu_pb, wd_rf;
    wire [63:0] mult_out;
    wire [31:0] lo_out, hi_out, mult_mux_out, jr_mux_out;

    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, jr_mux_out, 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_next);
    mux2 #(32) pc_jreg_mux (jump_reg, pc_next, alu_pa, jr_mux_out);

    // --- RF Logic --- //
    //mux2 #(5)  rf_wa_mux  (reg_dst, instr[20:16], instr[15:11], rf_wa);
    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, mult_mux_out,
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_ctrl, alu_pa, alu_pb, zero, alu_out);

    // --- MEM Logic --- //
    mux2 #(32) rf_wd_mux  (dm2reg, alu_out, rd_dm, wd_rf);

    // --- MULTU Logic --- //
    mult multu (alu_pa, wd_dm, mult_out);
    dreg2 lo (clk, rst, mult_en, mult_out[31:0], lo_out);
    dreg2 hi (clk, rst, mult_en, mult_out[63:32], hi_out);
    mux4 #(32) mult_mux(mult_sel, pc_plus4, wd_rf, lo_out, hi_out, mult_mux_out);

endmodule

module mips_top(
               input wire                    clk,
               input wire                    rst,
               input wire [4:0]       ra3,
               input wire [31:0]      instr,
               output wire            we_dm,
               output wire [31:0]     rd_dm,
               output wire [31:0]     pc_current,
               output wire [31:0]     alu_out,
               output wire [31:0]     wd_dm,
               output wire [31:0]     rd3
               );
    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
```

Discussion:

The purpose of this lab was to further our understanding of single-cycle CPUs by implementing MULTU, MFHI, MFLO, JAL, JR into the previous labs. This was a two-week lab where in the first week we designed the connections of the system and the control logic, and in the second week we implemented the code in Verilog and validated our results onboard our BASYS 3 FPGA. The results of the labs are above. Overall as visible in the test bench the we were able to achieve the correct output from our design. Our group struggled with getting the overall layout correct and spent a lot of time chasing down loose connections, however, once all the connections were wired in correctly our control logic was able to correctly control the process.