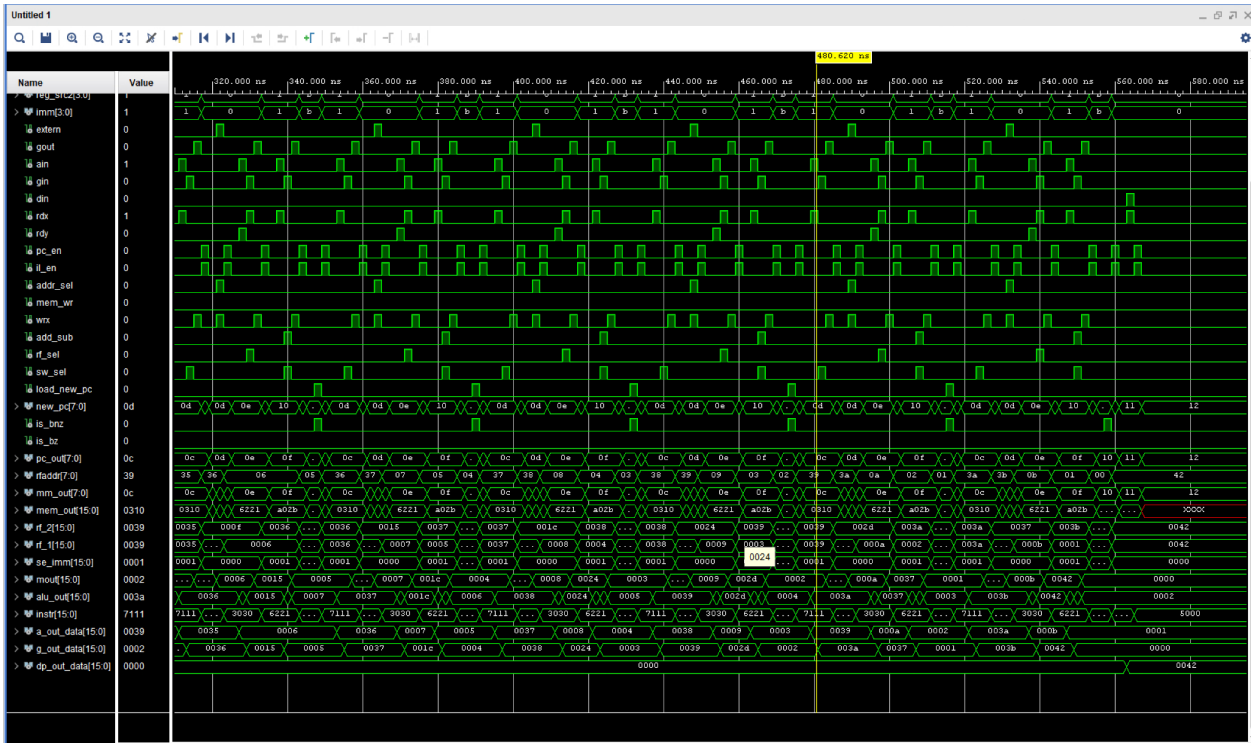
Curtis Robinson, Jack Landers
ELEN 122 Lab Report #6
March 9th, 2023

Lab Report #6

Timing Diagrams:



• What problems did you encounter while implementing and testing your system?

While we were implementing the system we had an issue with the displaying of the final values in the DP. We did not realize that we were not supposed to have any of the other control signals enabled while DP enable was on. This just confused us even though values were appearing in the DP.

• Did any problems arise when demonstrating for the TA? In other words, did the TA ask you to demonstrate something that you did not think of yourself? What was the scenario that you were asked to demonstrate? Provide some thoughts about why you didn't think of this yourself.

When we were demoing to the TA we did have some issues with the mux selection signals. Some of the wires we were trying to connect were not right. This prevented us from the selection of the right values. We possibly could have prevented this by fully understanding the block diagram and how data was flowing through the schematic.

• Pick another CTI operation that could have been implemented. Describe what changes to the datapath would have been necessary to support this operation, and how your state machine would need to be modified to correctly sequence the relevant control signals.

One of the other CTI operations we would add would be a jump. This would require us to get the newPC in a different location which would require us to receive that location in another latch. We would need to update our state diagram to add another state called jump which would not be a conditional but a direct jump to a new address dictated by the newPC.

i6_SM.v

```
module l6_SM(input clk,
                input reset,
                input [3:0] operation,
                output reg _Extern,
                output reg Gout,
                output reg Ain,
                output reg Gin,
                output reg DPin,
                output reg RdX,
                output reg RdY,
                output reg WrX,
                output reg add_sub,
                output reg pc_en,
                output reg ILin,
                output reg rf_sel,
                output reg sw_sel,
                output reg MemWr,
                output reg AddrSel,
                output reg bz,  /* new */
                output reg bnz, /* new */
                output [3:0] cur_state);

    // state definitions
     parameter FETCH        =  4'b0000;
     parameter DECODE       =  4'b0001;
     parameter LOAD         =  4'b0010;
     parameter READ_Y       =  4'b0011;
     parameter READ_X       =  4'b0100;
     parameter ADD          =  4'b0101;
     parameter SUB          =  4'b0110;
     parameter MV           =  4'b0111;
     parameter WRITE_X      =  4'b1000;
     parameter ADDI         =  4'b1001;
     parameter SUBI         =  4'b1010;
     parameter DISP         =  4'b1011;
     parameter HALT         =  4'b1100;
```

```
    parameter STORE        =  4'b1101;
    /* TODO #3-1: add two states for bz and bnz */
    parameter BNZ          =  4'b1110;
    parameter BZ           =  4'b1111;

    reg [3:0] state = FETCH;
    assign cur_state = state;


    always@(*)
    begin
       case(state)
           FETCH:
               begin
                   _Extern = 1'b0;
                   Gout = 1'b0;
                   //Iout = 1'b0;
                   Ain = 1'b0;
                   Gin = 1'b0;
                   DPin = 1'b0;
                   RdX = 1'b0;
                   RdY = 1'b0;
                   WrX = 1'b0;
                   add_sub = 1'b0;
                   pc_en = 1'b1;
                   ILin = 1'b1;
                   rf_sel = 1'b0;
                     sw_sel = 1'b0;
                     MemWr = 1'b0;
                     AddrSel = 1'b0;
                     /* TODO #3-3: output handling for bz and
bnz */
                     bz = 1'b0;
                     bnz = 1'b0;
               end
           DECODE:
               begin
```

```
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            LOAD:
                begin
                    _Extern = 1'b1;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b1;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
```

```
                    MemWr = 1'b0;
                    AddrSel = 1'b1;
                     /* TODO #3-3: output handling for bz and
bnz */
                    bz = 1'b0;
                    bnz = 1'b0;
                end
            READ_Y:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b1;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b1;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            READ_X:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b1;
                    Gin = 1'b0;
```

```
                    DPin = 1'b0;
                    RdX = 1'b1;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            ADD:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b1;
                    DPin = 1'b0;
                    RdX = 1'b1;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b1;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
```

```
                bnz = 1'b0;
            end
        SUB:
            begin
                _Extern = 1'b0;
                Gout = 1'b0;
                //Iout = 1'b0;
                Ain = 1'b0;
                Gin = 1'b1;
                DPin = 1'b0;
                RdX = 1'b0;
                RdY = 1'b1;
                WrX = 1'b0;
                add_sub = 1'b1;
                pc_en = 1'b0;
                ILin = 1'b0;
                rf_sel = 1'b1;
                  sw_sel = 1'b0;
                  MemWr = 1'b0;
                  AddrSel = 1'b0;
                   /* TODO #3-3: output handling for bz and
bnz */
                  bz = 1'b0;
                  bnz = 1'b0;
            end
        MV:
            begin
                _Extern = 1'b0;
                Gout = 1'b0;
                //Iout = 1'b0;
                Ain = 1'b0;
                Gin = 1'b1;
                DPin = 1'b0;
                RdX = 1'b0;
                RdY = 1'b0;
                WrX = 1'b0;
                add_sub = 1'b0;
```

```
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b1;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            WRITE_X:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b1;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b1;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            HALT:
                begin
                    _Extern = 1'b0;
```

```
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            DISP:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b0;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b1;
                    RdX = 1'b1;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
```

```
                    AddrSel = 1'b0;
                     /* TODO #3-3: output handling for bz and
bnz */
                    bz = 1'b0;
                    bnz = 1'b0;
                end
            ADDI:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b1;
                    Ain = 1'b0;
                    Gin = 1'b1;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b1;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            SUBI:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b1;
                    Ain = 1'b0;
                    Gin = 1'b1;
                    DPin = 1'b0;
```

```
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b1;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b1;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
                end
            STORE:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b1;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b1;
                      AddrSel = 1'b1;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b0;
```

```
                end
            BNZ:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b1;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b0;
                      bnz = 1'b1;
                end
            BZ:
                begin
                    _Extern = 1'b0;
                    Gout = 1'b0;
                    //Iout = 1'b1;
                    Ain = 1'b0;
                    Gin = 1'b0;
                    DPin = 1'b0;
                    RdX = 1'b0;
                    RdY = 1'b0;
                    WrX = 1'b0;
                    add_sub = 1'b0;
                    pc_en = 1'b0;
```

```
                    ILin = 1'b0;
                    rf_sel = 1'b0;
                      sw_sel = 1'b0;
                      MemWr = 1'b0;
                      AddrSel = 1'b0;
                       /* TODO #3-3: output handling for bz and
bnz */
                      bz = 1'b1;
                      bnz = 1'b0;
                end
        endcase
     end



/*
opcode encodings
0000 - load
0001 - move
0010 - subtract
0011 - add
0100 - disp
0101 - HALT
0110 - subi
0111 - addi
1000 - store
1001 - bz
1010 - bnz
*/

     always@(posedge clk or posedge reset)
     begin

          if (reset==1) state <= FETCH;
          else
          case(state)
               FETCH:
```

```
                        state <= DECODE;
                DECODE:
                        if(operation == 4'b0000) state <= LOAD;
                        else if(operation == 4'b0001) state <=
READ_Y;
                        else if(operation == 4'b0010) state <=
READ_X;
                        else if(operation == 4'b0011) state <=
READ_Y;
                        else if(operation == 4'b0100) state <=
DISP;
                        else if(operation == 4'b0101) state <=
HALT;
                        else if(operation == 4'b0110) state <=
READ_X;
                        else if(operation == 4'b0111) state <=
READ_X;
                        else if(operation == 4'b1000) state <=
STORE;
                        /* TODO #3-2: state transitions for bz
and bnz */
                        else if(operation == 4'b1001) state <=
BZ;
                        else if(operation == 4'b1010) state <=
BNZ;
                        else state <= FETCH;
                LOAD:
                        //state <= DONE;
                        state <= FETCH;
                READ_Y:
                             if(operation == 4'b0001) state <=
MV;
                             else if(operation == 4'b0011)
state <= ADD;
                             else state <= READ_Y;
                READ_X:
```

```
                                if(operation == 4'b0010) state <=
SUB;
                                else if(operation == 4'b0110)
state <= SUBI;
                                else if(operation == 4'b0111)
state <= ADDI;
                                else state <= READ_X;
                ADD:
                          state <= WRITE_X;
                SUB:
                          state <= WRITE_X;
                MV:
                          state <= WRITE_X;
                WRITE_X:
                          state <= FETCH;
                DISP:
                          state <= FETCH;
                ADDI:
                          state <= WRITE_X;
                SUBI:
                         state <= WRITE_X;
                STORE:
                          state <= FETCH;
               /* TODO #3-2: state transitions for bz and bnz */
                BNZ:
                          state <= FETCH;
                BZ:
                          state <= FETCH;
                HALT:
                          state <= HALT;
                default: state <= FETCH;

           endcase

      end //end always
```

```
endmodule
```

16_TB.v

```
module l6_TB();
    /* top-level design file for lab 5 */

    reg clk, rst;   // clock and reset

    //state machine signals

    wire [3:0] smstate;

    wire [15:0] IM_out;

     /* decoded instruction signals: see the instruction
encoding */
     wire [3:0] opcode;
     wire [3:0] reg_dst;
     wire [3:0] reg_src1;
     wire [3:0] reg_src2;
     wire [3:0] imm;

     /* control signals */
     wire extern;       /* d_mux selection signal 1 */
     wire gout;         /* d_mux selection signal 2 */
     wire ain;          /* latch a enable */
     wire gin;          /* latch g enable */
     wire din;          /* latch dp enable */
    wire rdx, rdy;      /* read register enable: will not be
used */
     wire pc_en;        /* PC enable */
     wire il_en;        /* Instruction latch enable */
     wire addr_sel;     /* Mem Mux selection signal */
     wire mem_wr;        /* Memory write enable */
     wire wrx;           /* RF write enable */
```

```
     wire add_sub;        /* alu control */
     wire rf_sel;       /* imm_mux selection signal 1 */
     wire sw_sel;       /* imm_mux selection singal 2 */
     wire load_new_pc;  /* load new PC */

     /* datapath signals related to TODO #4 */
     wire [7:0] new_pc;             /* new PC calculated by
branchPC */
     wire is_bnz;                   /* is this instruction bnz?
*/
     wire is_bz;                    /* is this instruction bz?
*/

     wire [7:0] pc_out;             /* PC output: PC -> Mem Mux
*/
    wire [7:0] rfaddr;              /* address from RF */
    wire [7:0] mm_out;              /* memory mux output: Mem
Mux -> MEM */
     wire [15:0] mem_out;           /* MEM output: MEM -> ilatch
or MEM -> d_mux */
    wire [15:0] rf_2, rf_1;         /* two rf output */
    wire [15:0] se_imm;             /* sign-extended imm signal:
SE -> imm_mux */
    wire [15:0] mout;               /* data mux (d_mux) out */
    wire [15:0] alu_out;            /* ALU out: ALU -> latch g
*/

    wire [15:0] instr;              /* instruction: ilatch
output */

    wire [15:0] a_out_data;          /* latch a output */
    wire [15:0] g_out_data;          /* latch g output */
    wire [15:0] dp_out_data;         /* latch dp output */
```

```
    /* TODO # 4: complete the datapath with the three
modified/added modules:
                l6_PC,
                l6_branchPC, and
                l6_branch */

    l6_branchPC
branchPC(.currPC(pc_out),.offset(imm),.adjustedPC(new_pc));
    l6_branch
branch(.branch(load_new_pc),.rf_data_in(rf_1),.bz(is_bz),.bnz(is
_bnz));
    l6_PC
pc(.clk(clk),.countEn(pc_en),.reset(rst),.loadNewPC(load_new_pc
),.newPC(new_pc),.Address(pc_out));

    assign rfaddr = rf_1[7:0];

    /* Memory Mux */
     mux_2_to_1 #(.bit_width(8)) mm(.in0(pc_out),
                                    .in1(rfaddr),
                                    .sel(addr_sel),
                                    .mux_output(mm_out));
    /* Memory (MEM) */
     l6_MEM mem(.clk(clk),
                .address(mm_out),
                .DataIn(rf_2),
                .MemWr(mem_wr),
                .DataOut(mem_out));

     /* instruction latch */
    A #(.bit_width(16)) ilatch(.Ain(mem_out), .load_en(il_en),
.Aout(instr));

    /* instruction decode */
    assign opcode = instr[15:12];
    assign reg_dst = instr[11:8];
    assign reg_src1 = instr[7:4];
```

```
assign reg_src2 = instr[3:0];
assign imm = instr[3:0];

/* 16-bit sign-extension */
SE_16 se(.imm(imm), .extended(se_imm));

l6_SM sm(.clk(clk),
         .reset(rst),
         .operation(opcode),
         ._Extern(extern),
         .Gout(gout),
         .Ain(ain),
         .Gin(gin),
            .DPin(din),
         .RdX(rdx),
         .RdY(rdy),
         .WrX(wrx),
         .add_sub(add_sub),
            .rf_sel(rf_sel),
            .sw_sel(sw_sel),
            .pc_en(pc_en),
            .ILin(il_en),
            .MemWr(mem_wr),
            .AddrSel(addr_sel),
            .bz(is_bz),
            .bnz(is_bnz),
         .cur_state(smstate));


l5_RF RF(.clk(clk),
      .rst(rst),
      .DataIn(mout),
      .raddr_2(reg_src2),
      .raddr_1(reg_src1),
      .waddr(reg_dst),
      .WrX(wrx),
      .out_data_2(rf_2),
```

```
      .out_data_1(rf_1)
        );

ALU alu (.in_A(a_out_data),
             .in_B(IM_out),
             .add_sub(add_sub),
             .adder_out(alu_out));

 A #(.bit_width(16)) a(.Ain(rf_1),
     .load_en(ain),
     .Aout(a_out_data));

 A #(.bit_width(16)) g(.Ain(alu_out),
     .load_en(gin),
     .Aout(g_out_data));

 A #(.bit_width(16)) dp(.Ain(rf_1),
      .load_en(din),
      .Aout(dp_out_data));


/* this is now a modified mux */
data_mux d_mux(.switch_data(mem_out),
            .G_data(g_out_data),
            .Gout(gout),
            ._Extern(extern),
            .mux_output(mout));

 /* imm_mux */
 imm_mux imm_mux(.rf_data(rf_2),
           .sw_data(se_imm),
           .rf_select(rf_sel),
           .sw_select(sw_sel),
           .adder_b(IM_out));


initial begin
```

```
        clk = 0;
          //this testbench is a bit different from the others
          //the instructinos come entirely from the program
          //to verify correctness, you will need to see that the
          //results match what you expect from the program
          rst = 1;
          #20;
          rst = 0;
          #1000;
          $finish;
    end

    always #1 clk = !clk;

endmodule
```