

Ayden Dauenhauer and Joseph Khamisy

Prof. Yang

ELEN 122L

20 February 2024

Postlab 5

QUESTIONS:

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

We had a small mistake where we misunderstood how the load instruction worked when loading addresses which ended up costing a decent bit of time debugging since it turned out our results were correct.

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 did not think of this yourself.

After debugging and understanding the code for so long, we were able to demo for the TA completely.

This design uses memory addresses directly from registers, rather than allowing for an offset to be added to a register value. How would you modify the datapath to allow for an offset, particularly for a store operation? How would this impact your state machine design?

We could introduce an additional unit in the datapath that can add an offset to a base register value. This could be an adder that takes the value from the base and adds an immediate value (the offset) to it. We would need an additional state and flow paths to accommodate the new steps in the store operation.

CODE:

```
module SE_16(
```

```

        input [3:0] imm,
        output [15:0] extended
    );

    /* TODO #3-1: prefix the MSB 12 times */
    assign extended = {{12{imm[3]}}, {imm}};
    /* end of TODO #3-1 */

endmodule

```

```

module l5_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 signal 2 */

/* datapath signals */
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 */

```

```

PC pc(.clk(clk),
      .countEn(pc_en),
      .reset(rst),
      .Address(pc_out));

```

```

/* TODO #1: Instantiate Memory Mux */
assign rfaddr = rf_1[7:0];
mux_2_to_1 #(.bit_width(8)) mm(.in0(pc_out),
                                .in1(rfaddr),
                                .sel(addr_sel),
                                .mux_output(mm_out));
/* end of TODO #1 */

```

```

/* Memory (MEM) */

```

```

15_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));

15_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),
         .cur_state(smstate));

```

```

15_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));

/* TODO #2-2: instantiate the data MUX (data_mux.v) */
data_mux d_mux(.G_data(g_out_data),
    .switch_data(mem_out),
    .Gout(gout),
    ._Extern(extern),
    .mux_output(mout));

/* end of TODO #2-2 */

/* TODO #3-2: instantiate the imm mux (imm_mux.v) */
imm_mux imm_mux(.rf_data(rf_2),
    .sw_data(se_imm),

```

```

        .sw_select(sw_sel),
        .rf_select(rf_sel),
        .adder_b(IM_out));
/* end of TODO #3-2 */

```

```

initial begin
    clk = 0;
    //this testbench is a bit different from the others
    //the instructions 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

```

```

//this actually the L3_SM
//we add halt, rename idle to be fetch
module l5_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 [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;
```

```
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;      /* TODO #4: complete the control logic output */
    sw_sel = 1'b0;      /* TODO #4: complete the control logic output */
    MemWr = 1'b0;      /* TODO #4: complete the control logic output */
    AddrSel = 1'b0;     /* TODO #4: complete the control logic output */
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; /* TODO #4 */
    sw_sel = 1'b0; /* TODO #4 */
    MemWr = 1'b0; /* TODO #4 */
    AddrSel = 1'b0; /* TODO #4 */
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; /* TODO #4 */
        sw_sel = 1'b0; /* TODO #4 */
        MemWr = 1'b0; /* TODO #4 */
        AddrSel = 1'b1; /* TODO #4 */
    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; /* TODO #4 */
        sw_sel = 1'b0; /* TODO #4 */
        MemWr = 1'b0; /* TODO #4 */
        AddrSel = 1'b0; /* TODO #4 */
    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; /* TODO #4 */
        sw_sel = 1'b0; /* TODO #4 */

```

```

        MemWr = 1'b0; /* TODO #4 */
        AddrSel = 1'b0; /* TODO #4 */
    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; /* TODO #4 */
            sw_sel = 1'b0; /* TODO #4 */
            MemWr = 1'b0; /* TODO #4 */
            AddrSel = 1'b0; /* TODO #4 */
    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; /* TODO #4 */
            sw_sel = 1'b0; /* TODO #4 */
            MemWr = 1'b0; /* TODO #4 */
            AddrSel = 1'b0; /* TODO #4 */
    end

```

```

    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'b0; /* TODO #4 */
            sw_sel = 1'b0; /* TODO #4 */
            MemWr = 1'b0; /* TODO #4 */
            AddrSel = 1'b0; /* TODO #4 */
    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; /* TODO #4 */
            sw_sel = 1'b0; /* TODO #4 */
            MemWr = 1'b0; /* TODO #4 */
            AddrSel = 1'b1; /* TODO #4 */
    end
    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; /* TODO #4 */
    sw_sel = 1'b0; /* TODO #4 */
    MemWr = 1'b0; /* TODO #4 */
    AddrSel = 1'b0; /* TODO #4 */
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; /* TODO #4 */
    sw_sel = 1'b0; /* TODO #4 */
    MemWr = 1'b0; /* TODO #4 */
    AddrSel = 1'b1; /* TODO #4 */
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; /* TODO #4 */
        sw_sel = 1'b1; /* TODO #4 */
        MemWr = 1'b0; /* TODO #4 */
        AddrSel = 1'b0; /* TODO #4 */
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; /* TODO #4 */
        sw_sel = 1'b1; /* TODO #4 */
        MemWr = 1'b0; /* TODO #4 */
        AddrSel = 1'b0; /* TODO #4 */
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; /* TODO #4 */
        sw_sel = 1'b0; /* TODO #4 */
        MemWr = 1'b1; /* TODO #4 */
        AddrSel = 1'b1; /* TODO #4 */
    end
endcase
end

```

```

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

```

```

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;
else state <= FETCH;
```

LOAD:

```
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;
```

HALT:

```
state <= HALT;
```

```

        default: state <= FETCH;

    endcase

end //end always

endmodule

module data_mux(input [15:0] switch_data, // data from mem
    input [15:0] G_data,          // data from latch G (ALU)
    input Gout,
    input _Extern,
    output reg [15:0] mux_output);

always@(*)
begin
    /* TODO #2-1: complete the design of Data MUX */
    /* if Gout == 1 and switch_data == 0 G_data is chosen for output */
    if(Gout)
        mux_output <= G_data;
    else if(switch_data)
        mux_output <= switch_data;
    else
        mux_output <= 0;
    /* if Gout == 0 and switch_data == 1 switch_data is chosen for output */
    /* if both are 0, output is just deasserted */
    /* end of TODO #2-1 */
end

endmodule

2000 // sub r0, r0, r0
7006 // addi r0, r0, 6
3100 // add r1, r0, r0
3211 // add r2, r1, r1
3222 // add r2, r2, r2
0320 // load r3, r2(=48)
7221 // addi r2, r2, 1

```

```

0420 // load r4, r2(=49)
7221 // addi r2, r2, 1
0520 // load r5, r2(=50)
3634 // add r6, r3, r4
8056 // store r5, r6
3665 // add r6, r6, r5
4060 // disp r6
5000 // halt

```

DIAGRAM:

