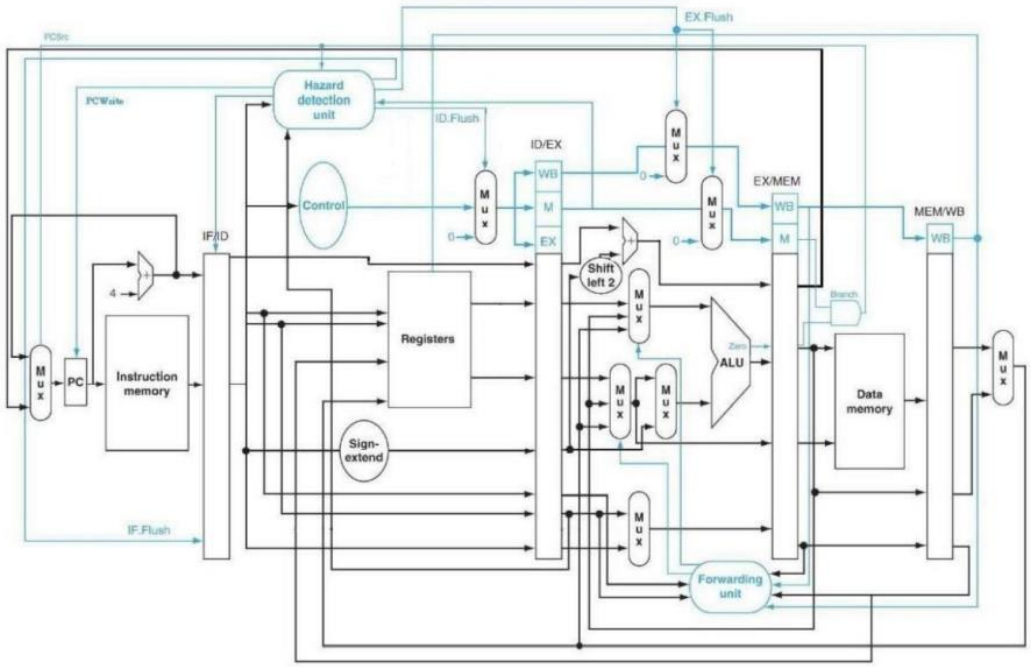
Computer Organization Lab5

Name:林秉

ID:109550112

Architecture diagrams:



Hardware module analysis:

Forwarding unit:

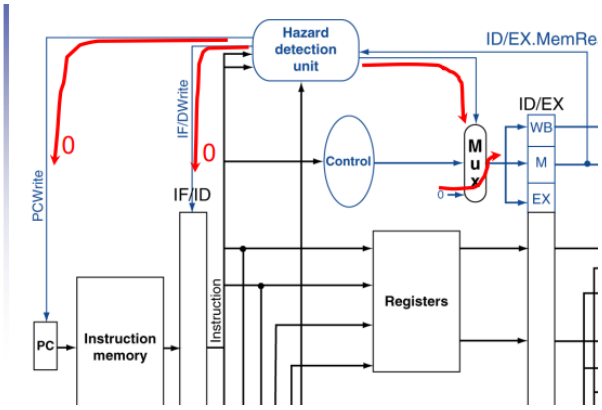
We assign value of forwarding A and B by the below table:

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Hazard detection:

- ID/EX.MemRead and
  ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
  (ID/EX.RegisterRt = IF/ID.RegisterRt))

If above situation happens, or instructions need to branch, we need to flush the pipe register of below.




```
always@(*)begin
    if((pc_src == 1 && typebranch == 0) || (pc_src == 0 && branchselect == 1 && typebranch == 1) || (pc_src == 1 && typebranch == 2) || (pc_src == 1 && typebranch == 3))
        if_flush = 1;
    else
        if_flush = 0;
end
```

(The if conditions will be explained in alu)
And we need to control the write register signal to 0 to prevent the register's value is changed into the wrong value.

Decoder:

```
wire r, addi, slti, beq,lw,sw,bne,bge,bgt;

//Main function

assign r = (instr_op_i == 0);
assign addi = (instr_op_i == 8);
assign slti = (instr_op_i == 10);
assign beq = (instr_op_i == 4);
assign lw =  (instr_op_i == 35);
assign sw =  (instr_op_i == 43);
assign bne = (instr_op_i == 5);
assign bge = (instr_op_i == 1);
assign bgt = (instr_op_i == 7);

always@(*)begin
        if(r | addi | slti | lw)
            RegWrite_o = 1;
    else
        RegWrite_o = 0;
end

always@(*)begin
        if(r)
            ALU_op_o = 1;
        else if(addi)
            ALU_op_o = 2;
        else if(sltilbge)
            ALU_op_o = 3;
        else if(beqlbne)
            ALU_op_o = 4;
        else if(lw)
            ALU_op_o = 5;
        else if(sw)
            ALU_op_o = 6;
        else if(bgt)
            ALU_op_o = 7;
```

We assign additional variables for extra branch instructions, and since branch has new types, we assign a new register branchtype to tell which branch instruction it is.

ALU_ctrl:

```
if(ALUOp_i == 1) begin///r-format
   case(funct_i)
   6'b100000:  ALUCtrl_o[2:0] = 2;///add
   6'b100010:  ALUCtrl_o[2:0] = 6;///sub
   6'b100100:  ALUCtrl_o[2:0] = 0;///and
   6'b100101:  ALUCtrl_o[2:0] = 1;///or
   6'b101010:  ALUCtrl_o[2:0] = 7;///slt
   6'b011000:  ALUCtrl_o[2:0] = 3;///mult
   endcase

end
else if(ALUOp_i == 4)begin///branch
   ALUCtrl_o[2:0] = 3'b110;
end
else if(ALUOp_i == 2)begin///addi
   ALUCtrl_o[2:0] = 3'b010;
end
else if(ALUOp_i == 3)begin///slti bge
   ALUCtrl_o[2:0] = 3'b111;
end
else if(ALUOp_i == 5)begin///lw
   ALUCtrl_o[2:0] = 3'b010;
end
else if(ALUOp_i == 6)begin///sw
   ALUCtrl_o[2:0] = 3'b010;
end
else if(ALUOp_i == 7)begin///sw
   ALUCtrl_o[2:0] = 3'b100;
end
ALUCtrl_o[3] = 0;
```

According to the new opcode, we assign new control value to ALU.

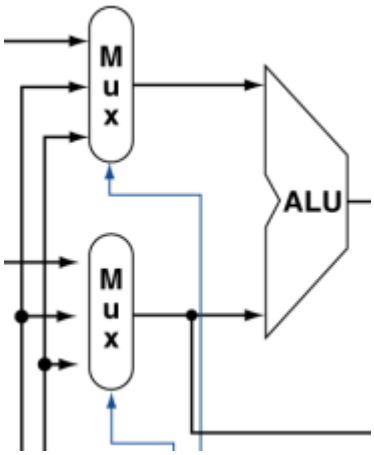
ALU:

```
always@(*)
    begin
    result_o = 0;
    case(ctrl_i)
        0: result_o = src1_i & src2_i;
        1: result_o = src1_i | src2_i;
        2: result_o = src1_i + src2_i;
            3: result_o = src1_i * src2_i;
            4: if(src1_i<=src2_i)
              begin
                 result_o = 1;
              end
            else
              begin
                 result_o = 0;
              end
        6: result_o = src1_i - src2_i;
        7: if(src1_i<src2_i)
              begin
                 result_o = 1;
              end
            else
              begin
                 result_o = 0;
              end
        12: result_o = ~(src1_i | src2_i);
    endcase
    end

assign zero_o = (result_o == 0);
```

Ctrl4 is for bgt, ctrl6 is for beq and bne, ctrl7 is for bge. If these need to branch, the zero will be set into 1 except bne, after anding zero and branch, we send the value to hazard detect to check whether we need to flush instructions.

Mux3to1:



We add 3to1 mux for the alu and the control is the forward unit output.

Pipe_reg:

```
wire        branchselect;
wire [1:0]  bratype;
/***************************************
Instantiate modules
****************************************/
assign pc_src = MEM_branch & MEM_zero;
assign branchselect = MEM_branch;
assign IF_ID_rst = (rst_i == 0 || IF_flush == 1) ? 0 : 1;
```

We add some variables for data hazard. Bratype(branchtype) is generated in decoder and inputs in hazard detection.

```
hazard_detect HD(
    .id_ex_memread(EX_memread),
    .id_ex_rt(EX_regrt),
    .if_id_rs(ID_ins_mem_o[25:21]),
    .if_id_rt(ID_ins_mem_o[20:16]),
    .pc_src(pc_src),
    .branchselect(branchselect),
    .typebranch(bratype),
    .pc_write(pc_write),
    .if_id_write(IF_ID_write),
    .if_flush(IF_flush),
    .id_flush(ID_flush),
    .ex_flush(EX_flush)
    );
```

```
forward_unit     fwd(
    .ex_mem_rd(MEM_Write_Reg),
    .mem_wb_rd(WB_Write_Reg),
    .id_ex_rs(EX_regrs),
    .id_ex_rt(EX_regrt),
    .ex_mem_regwrite(MEM_regwrite),
    .mem_wb_regwrite(WB_regwrite),

    .forwardA(forwardA),
    .forwardB(forwardB)
);
```

And we have the new added module for forwarding and hazard detection unit.

Finished part:

Data1:(clock15)

```
################################ clk_count = 15###################################
==================================Register=====================================
r0 =    0, r1 =   16, r2 =  256, r3 =    8, r4 =   16, r5 =    8, r6 =   24, r7 =   26

r8 =    8, r9 =    1, r10=    0, r11=    0, r12=    0, r13=    0, r14=    0, r15=    0

r16=    0, r17=    0, r18=    0, r19=    0, r20=    0, r21=    0, r22=    0, r23=    0

r24=    0, r25=    0, r26=    0, r27=    0, r28=    0, r29=    0, r30=    0, r31=    0

==================================Memory=======================================
m0 =    0, m1 =   16, m2 =    0, m3 =    0, m4 =    0, m5 =    0, m6 =    0, m7 =    0

m8 =    0, m9 =    0, m10=    0, m11=    0, m12=    0, m13=    0, m14=    0, m15=    0

m16=    0, m17=    0, m18=    0, m19=    0, m20=    0, m21=    0, m22=    0, m23=    0

m24=    0, m25=    0, m26=    0, m27=    0, m28=    0, m29=    0, m30=    0, m31=    0
```

Data2:(clock59)

```
################################ clk_count = 59###################################
==================================Register=====================================
r0 =    0, r1 =    0, r2 =   16, r3 =    1, r4 =    0, r5 =   16, r6 =    0, r7 =    0

r8 =    2, r9 =    0, r10=    0, r11=    0, r12=    0, r13=    0, r14=    0, r15=    0

r16=    0, r17=    0, r18=    0, r19=    0, r20=    0, r21=    0, r22=    0, r23=    0

r24=    0, r25=    0, r26=    0, r27=    0, r28=    0, r29=    0, r30=    0, r31=    0

==================================Memory=======================================
m0 =    0, m1 =    3, m2 =    0, m3 =    1, m4 =    0, m5 =    0, m6 =    0, m7 =    0

m8 =    0, m9 =    0, m10=    0, m11=    0, m12=    0, m13=    0, m14=    0, m15=    0

m16=    0, m17=    0, m18=    0, m19=    0, m20=    0, m21=    0, m22=    0, m23=    0

m24=    0, m25=    0, m26=    0, m27=    0, m28=    0, m29=    0, m30=    0, m31=    0
```

Problems you met and solutions:

I had trouble on the program counter since I think my design is correct, but it still failed to read data. After asking classmates for help, I find out that I need to add write signal as additional input, after debugging the problem was solved.

Summary:

The concept of this lab is far more difficult from the previous ones, not only we need to take care of the forwarding and hazard detect, but we need to add additional instructions for branch, this is the most difficult part, it took me 2 days to finish only a portion of it, after that, I find myself exhausted for debugging, but it's good to know that the semester is finally over, what a relief.