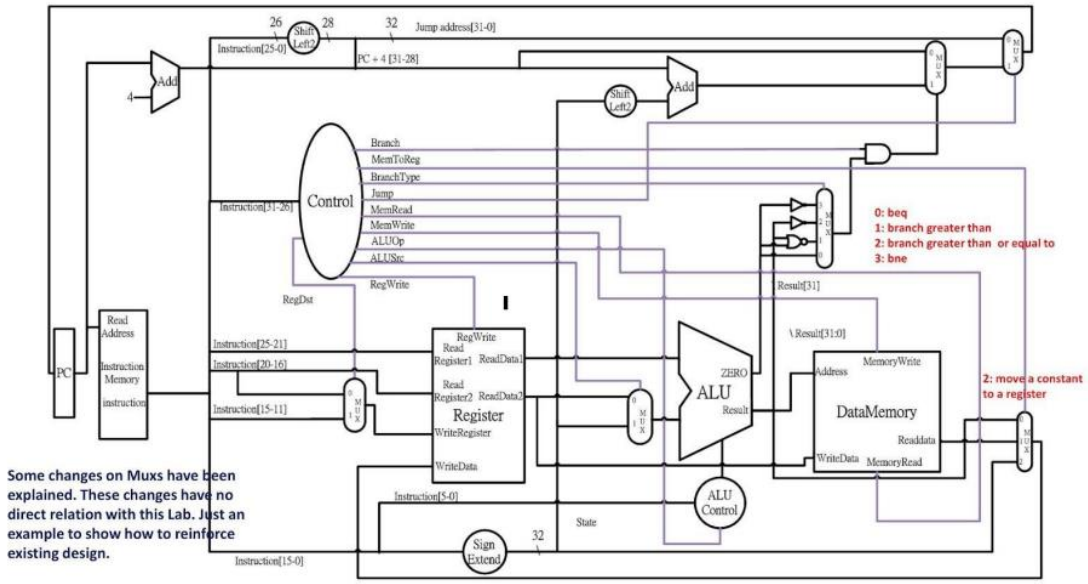
Computer Organization Lab3

Name:林秉

ID:109550112

Architecture diagrams:



Hardware module analysis:

Since this lab is build based on lab2, so lots of structure are the same, I only introduce the different part.

Decoder:

```
//I/O ports
input  [6-1:0] instr_op_i;

output         RegWrite_o;
output [3-1:0] ALU_op_o;
output         ALUSrc_o;
output [2-1:0] RegDst_o;
output         Branch_o;
output         Jump_o;
output         MemRead_o;
output         MemoryWrite_o;
output [2-1:0] MemtoReg_o;

//Internal Signals
reg    [3-1:0] ALU_op_o;
reg            ALUSrc_o;
reg            RegWrite_o;
reg    [2-1:0] RegDst_o;
reg            Branch_o;
reg            Jump_o;
reg            MemRead_o;
reg            MemoryWrite_o;
reg    [2-1:0] MemtoReg_o;
```

Since we need to add more instructions, some output need to have more bits and need to add more output ports. Like regdst and memorywrite.

```
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 jump = (instr_op_i == 2);
assign jal = (instr_op_i == 3);

always@(*)begin
        if(r)
           ALU_op_o = 1;
        else if(addi)
           ALU_op_o = 2;
        else if(slti)
           ALU_op_o = 3;
        else if(beq)
           ALU_op_o = 4;
        else if(lw)
           ALU_op_o = 5;
        else if(sw)
           ALU_op_o = 6;
        else
           ALU_op_o = 0;

        if(jal)
            RegDst_o = 2;
        else if(r)
            RegDst_o = 1;
        else
            RegDst_o = 0;



        if(jal)
            MemtoReg_o = 2;
        else if(lw)
            MemtoReg_o = 1;
        else
            MemtoReg_o = 0;

        Branch_o = beq;
        RegWrite_o = r | addi | slti | lw | jal;
        Jump_o = jal | jump;
        MemRead_o = lw;
        MemoryWrite_o = sw;
        ALUSrc_o = addi | slti | lw | sw;
end
```

After declaring the registers, we follow the MIPS code to assign value and do the corresponding movement of the given op code.

ALUCTRL:

```
assign jr_o = (ALUOp_i == 1 && funct_i == 8);

always@(*)
begin
        if(ALUOp_i == 1) //r-format
    begin
            ALUCtrl_o[0] = (funct_i == Or) || (funct_i == slt);
            ALUCtrl_o[1] = (funct_i == add) || (funct_i == sub) || (funct_i == slt);
            ALUCtrl_o[2] = (funct_i == sub) || (funct_i == slt);
        end
        else if(ALUOp_i == 4)///beq
        begin
            ALUCtrl_o[2:0] = 3'b110;
        end
        else if(ALUOp_i == 2)//addi
        begin
            ALUCtrl_o[2:0] = 3'b010;
        end
        else if(ALUOp_i == 3)//slti
        begin
            ALUCtrl_o[2:0] = 3'b111;
        end
        else if(ALUOp_i == 5)//lw
        begin
            ALUCtrl_o[2:0] = 3'b010;
        end
        else if(ALUOp_i == 6)//sw
        begin
            ALUCtrl_o[2:0] = 3'b010;
        end
        ALUCtrl_o[3] = 0;
end
//Select exact operation
```

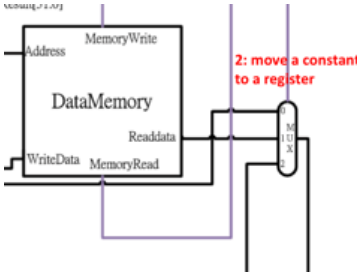
We assign extra control value for the new instructions according to the given ALUOP.

MUX3to1:

```
always@(*)
  begin
    if(select_i == 2)
      begin
        data_o = data2_i;
      end
    else if(select_i == 1)
      begin
        data_o = data1_i;
      end
    else
      begin
        data_o = data0_i;
      end
```

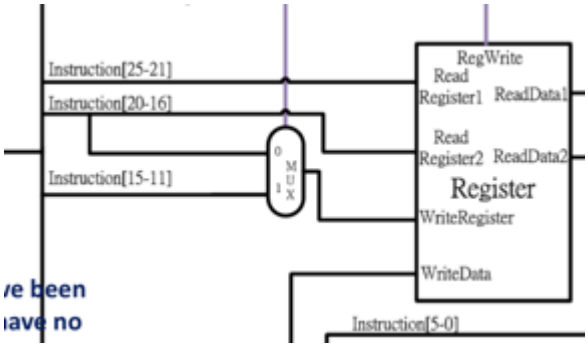
Since the structure exists 3to1's mux, we create a mux3to1 module to help us complete the CPU.

```
MUX_3to1 #(.size(32)) Mux_Mem_to_Reg(///vv
        .data0_i(alu_result),
        .data1_i(data_out),
        .data2_i(add1_o),
        .select_i(MemtoReg),
        .data_o(writereg)
        );
```

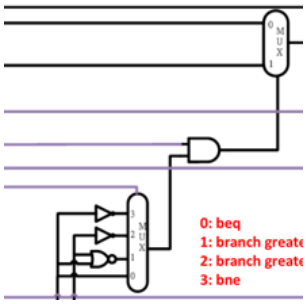


Above code is for the mux below.

```
MUX_3to1 #(.size(5)) Mux_Write_Reg(///vv
        .data0_i(ins_memory[20:16]),
        .data1_i(ins_memory[15:11]),
        .data2_i(5'd31),
        .select_i(regdst),
        .data_o(muxtoreg)
        );
```



Above code is for the below mux , for jal , we assign 11111 for muxtoreg.



This 4to1mux's output can be simplified into adder_select(zero&branch) in the code, so we don't need a mux to generate control for the 2to1mux.

Finished part:

Data1:

```
PC =          704
Data Memory =          1,          2,          0,          0,          0,          0,          0,          0
Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
Registers
R0 =          0, R1 =          1, R2 =          2, R3 =          3, R4 =          4, R5 =          5, R6 =          1, R7 =          2
R8 =          4, R9 =          2, 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 =        128, R31 =          0, R32 =          0
```

Data2:

```
PC =          236
Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
Data Memory =          0,          0,          0,          0,         68,          2,          1,         68
Data Memory =          2,          1,         68,          4,          3,         16,          0,          0
Registers
R0 =          0, R1 =          0, R2 =          5, R3 =          0, R4 =          0, R5 =          0, R6 =          0, R7 =          0
R8 =          0, 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 =        128, R31 =          0, R32 =         16
```

Both result satisfies the answer.

Problems you met and solutions:

I once find the output is slightly different from the answer, but I don't think my structure is wrong, after many double-checks, I find the problem is I have a register's size too small, after adding an extra bit, the output is correct.

Summary:

This lab is very similar to lab2, so I first think it's easy, but after having many bugs on my design, I find myself too careless on my coding, the structure is similar

indeed, but it has a lot of details to check, which are so small that you wouldn't expect to have problem, I think I need to be careful on my next design.