Lab05: 5-Stage Pipeline Processor

team 29, teammate : 109550053 楊立嘉，109550137 徐敏芝

Part1. Detailed description of the implementation

- All instructions we implement in Lab5:
  - nop
  - R type: add, sub, and, or, xor, slt
  - I type: addi, slti, slli
  - load/store: sw, lw
  - branch/jump: beq, jal
- The six files: Adder, ALU_Ctrl, alu, decoder, immGen, MUX are almost the same as Lab4, we just add codes corresponding to new instruction in Lab5 in them. Other unit implementation are description as the following.
  - Note that in `decoder.v`, we turn 1 bit control signal `MemToReg` into 2 bit control signal `WriteBackA, WriteBackB`, but we didn't use the variable `WriteBackB`(still give it a value with zero), because they are used for `jalr` in Lab4 but we do not need to implement it in Lab5, so you can just treat `WriteBackA` as `MemToReg`, but we want to save `WriteBackB` for future use. As the same result, we change control signal `WB_i` in all .v files from 3 bits(`{RegWrite, MemToReg, jump}`) to 4 bits (`{RegWrite,WriteBackA,WriteBackB,jump}`).

ForwardingUnit

```
/* Write your code HERE */
always @(*) begin
    // for src1
    if(EXEMEM_RegWrite && (EXEMEM_RD != 0) && (EXEMEM_RD == IDEXE_RS1))  //EX hazard
        ForwardA = 2'b10;
    else if (MEMWB_RegWrite && (MEMWB_RD != 0) && (MEMWB_RD == IDEXE_RS1)) //Mem hazard
        //因為我用 else if 所以可以不用打判斷優先權那一串
        ForwardA = 2'b01;
    else
        ForwardA = 2'b00;

    // for src2
    if(EXEMEM_RegWrite && (EXEMEM_RD != 0) && (EXEMEM_RD == IDEXE_RS2))  //EX hazard
        ForwardB = 2'b10;
    else if (MEMWB_RegWrite && (MEMWB_RD != 0) && (MEMWB_RD == IDEXE_RS2)) //Mem hazard
        //因為我用 else if 所以可以不用打判斷優先權那一串
        ForwardB = 2'b01;
    else
        ForwardB = 2'b00;

end
```

- Just type dependency condition as teacher's slide in the class, but note that Mem hazard condition do not contain `!(EXEMEM_RegWrite && (EXEMEM_RD != 0) && (EXEMEM_RD == IDEXE_RS1))`, I use `else if` to give EX hazard priority.

Hazard_detection

```
/* Write your code HERE */
always @(*)begin
    if (IDEXE_memRead && (IDEXE_regRd == IFID_regRs || IDEXE_regRd == IFID_regRt))begin
        PC_write = 1'b0;
        IFID_write = 1'b0;
        control_output_select = 1'b1;//會把 contorl 都變0
    end
    else begin
        PC_write = 1'b1;
        IFID_write = 1'b1;
        control_output_select = 1'b0;//正常的 control 訊號
    end
end
```

- Just type hazard condition as teacher's slide in the class.

Shift_Left_1

```
/* Write your code HERE */
assign data_o = {data_i[31-1:0],1'b0};
```

IFID_register

```
/* Write your code HERE */
always @(posedge clk_i or negedge rst_i)begin

    if (~rst_i) begin //全部設定成0
        address_o <= 32'b0;
        instr_o <= 32'b0;
        pc_add4_o <= 32'b0;
    end
    else if (~IFID_write) begin
        //不給改寫，PC應該是由 PC write 直接控制讓他不變 stall
        address_o <= address_i;
        instr_o <= instr_o;
        pc_add4_o <= pc_add4_i;
    end
    else if(flush) begin
        address_o <= address_i;
        instr_o <= 32'b0;
        pc_add4_o <= pc_add4_i;
    end
    else begin
        address_o <= address_i;
        instr_o <= instr_i;
        pc_add4_o <= pc_add4_i;
    end
end
```

- Consider three cases other then normal case:
  - Test case have not start yet: set everything to be zero
  - If load use appear, then disable `IFID_Write`, keep instruction the same.
  - If jump or branch happens, flush instruction to zero, and also set PCSrc to 1 in `Pipeline_CPU.v`:
    - `assign PCSrc = Jump || (Branch && Branch_zero);`
    - `assign IFID_Flush = Jump || (Branch && Branch_zero);`

IDEXE_register/EXEMEM_register/MEMWB_register

```
/* Write your code HERE */
always @(posedge clk_i or negedge rst_i)begin
    if (~rst_i)begin //全部設定成0
        instr_o <= 0;
        WB_o <= 0;
        Mem_o <= 0;
        Exe_o <= 0;

        data1_o <= 0;
        data2_o <= 0;
        immgen_o <= 0;
        alu_ctrl_input <= 0;
        WBreg_o <= 0;
        pc_add4_o <= 0;
        RS1_o <= 0;
        RS2_o <= 0;
    end
    else begin
        instr_o <= instr_i;
        WB_o <= WB_i;
        Mem_o <= Mem_i;
        Exe_o <= Exe_i;

        data1_o <= data1_i;
        data2_o <= data2_i;
        immgen_o <= immgen_i;
        alu_ctrl_input <= alu_ctrl_instr;
        WBreg_o <= WBreg_i;
        pc_add4_o <= pc_add4_i;
        RS1_o <= RS1_i;
        RS2_o <= RS2_i;
    end


end
```

```
/* Write your code HERE */
always @(posedge clk_i or negedge rst_i)begin
    if (~rst_i)begin //全部設定成0
        instr_o <= 0;
        WB_o <= 0;
        Mem_o <= 0;
        zero_o <= 0;
        alu_ans_o <= 0;
        rtdata_o <= 0;
        WBreg_o <= 0;
        pc_add4_o <= 0;
    end
    else begin
        instr_o <= instr_i;
        WB_o <= WB_i;
        Mem_o <= Mem_i;
        zero_o <= zero_i;
        alu_ans_o <= alu_ans_i;
        rtdata_o <= rtdata_i;
        WBreg_o <= WBreg_i;
        pc_add4_o <= pc_add4_i;
    end
end
```

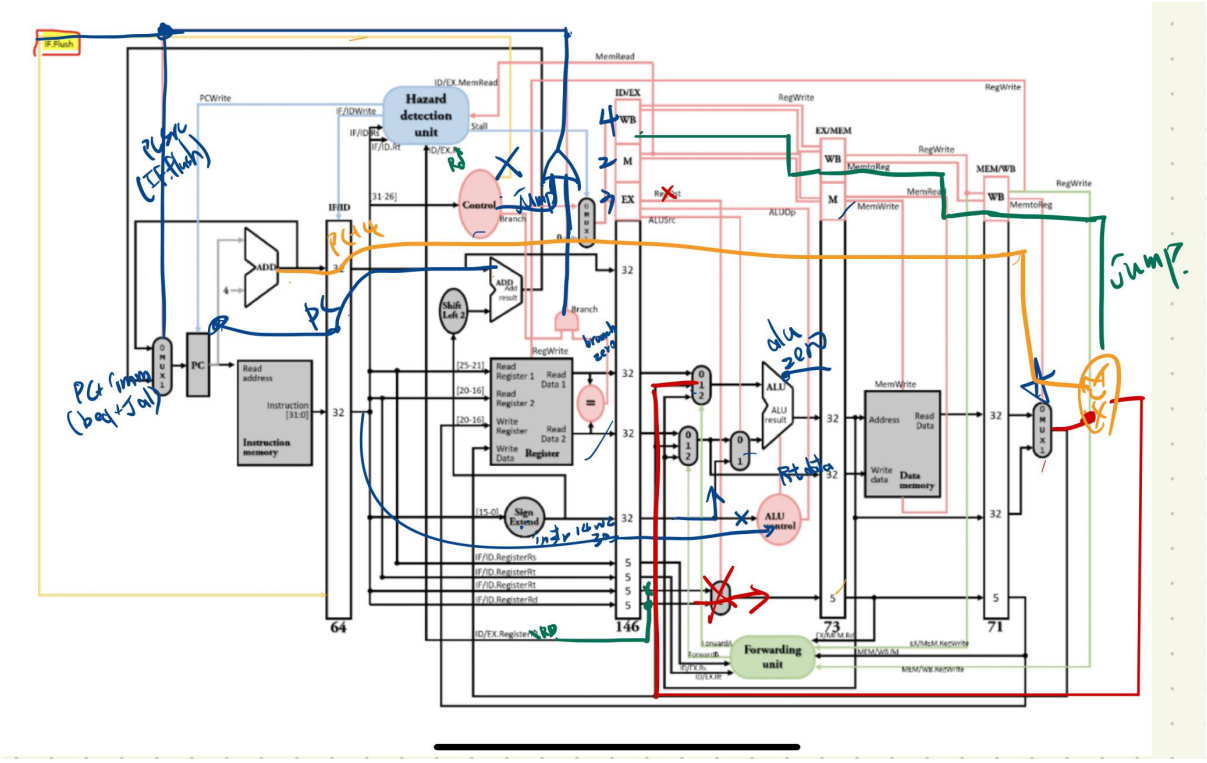
```
/* Write your code HERE */
always @(posedge clk_i or negedge rst_i)begin
    if (~rst_i)begin //全部設定成0
        WB_o = 4'b0;
        DM_o = 32'b0;
        alu_ans_o = 32'b0;
        WBreg_o = 5'b0;
        pc_add4_o = 32'b0;
    end
    else begin
        WB_o = WB_i;
        DM_o = DM_i;
        alu_ans_o = alu_ans_i;
        WBreg_o = WBreg_i;
        pc_add4_o = pc_add4_i;
    end


end
```

- Test case have not start yet: set everything to be zero
- Otherwise, output every input, do not change anything.

Pipeline_CPU

- Too much codes so we do not screenshot, but we will describe how we solve jump, branch in this part. The whole circuit is like following picture.



- jump

  The control line of **Jump** is sent by each level and the information of **PC+4**, finally they reach L5.

  In WB, we check **Jump** and decide the output of `MUX_2to1 MUX_jump` , if the instruction need to jump (Jump = 1), we get the output `MEMWB_PC_Add4_o` , if Jump = 0, which means we are not going to jump, than just send the `Write Date` we suppose to send at first.

- branch

  Add the immediate generated by *Imm_Gen* and add it with our PC to decide our destination, send it to the MUX before PC.

  In L2, we set `Branch_zero = (RSdata_o == RTdata_o )? 1:0;` and combine it with `Branch` comes from Decoder to decide whether we're going to jump or not.

  If yes, the control line `PCSrc` will be 1 and send to MUX before PC, MUX will choose **PC+Imm** we just sent.

Part2. Implementation results

Part3. Problems encountered and solutions

1. In lab4, the MUX controlled by `MemReg` has opposite order of 0 and 1 in lab5. So at first, our output of ALU can't be passed to Register successfully, the output always be 0. The reason is that we didn't modify the code when pasting from our previous lab, the problem was solved since we check the code again and again.
2. We met the problem about PC at testcase 11, our PC is 8 more than what it should be. We thought this problem came from the module *Shift_Left_1* we wrote by ourselves, but it seemed really well. At the end, we found out that in *Imm_Gen*, we already shift the immediate for branch instruction, so the immediate was shifted for 2 bits, we modified *Imm_Gen* and solved this problem.
3. For jump instruction, we tried to do it in L2 at first, but it couldn't send the address to `WriteRegister` in Register, so we move it to L5.