1. Source code and the note

```
6'b011000: ALUCtrl_o <= 4'b0011;    // mult 0011
```

這次新加的 MULT 把 ALU control 設成 0011

```
Pipe_Reg #(.size(64)) IF_ID(       //N is the total length of input/output
        .clk_i(clk_i),
        .rst_i(rst_i),
        .data_i({pc_out_added, instr}),
        .data_o({pc_out_added_id, instr_id})
);
```

用來連接IF/ID stage 的 pipelined register

```
Pipe_Reg #(.size(159)) ID_EX(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .data_i({ReadData1, ReadData2, instr_id[20:0], pc_out_added_id, RegWrite,
                RegDst, ALUSrc, Branch, MemRead, MemWrite, MemtoReg, ALUOp, signed_addr}),
        .data_o({ReadData1_ex, ReadData2_ex, instr_ex, pc_out_added_ex, RegWrite_ex,
                RegDst_ex, ALUSrc_ex, Branch_ex, MemRead_ex, MemWrite_ex, MemtoReg_ex, ALUOp_ex, signed_addr_ex})
);
```

用來連接ID/EXstage 的 pipelined register

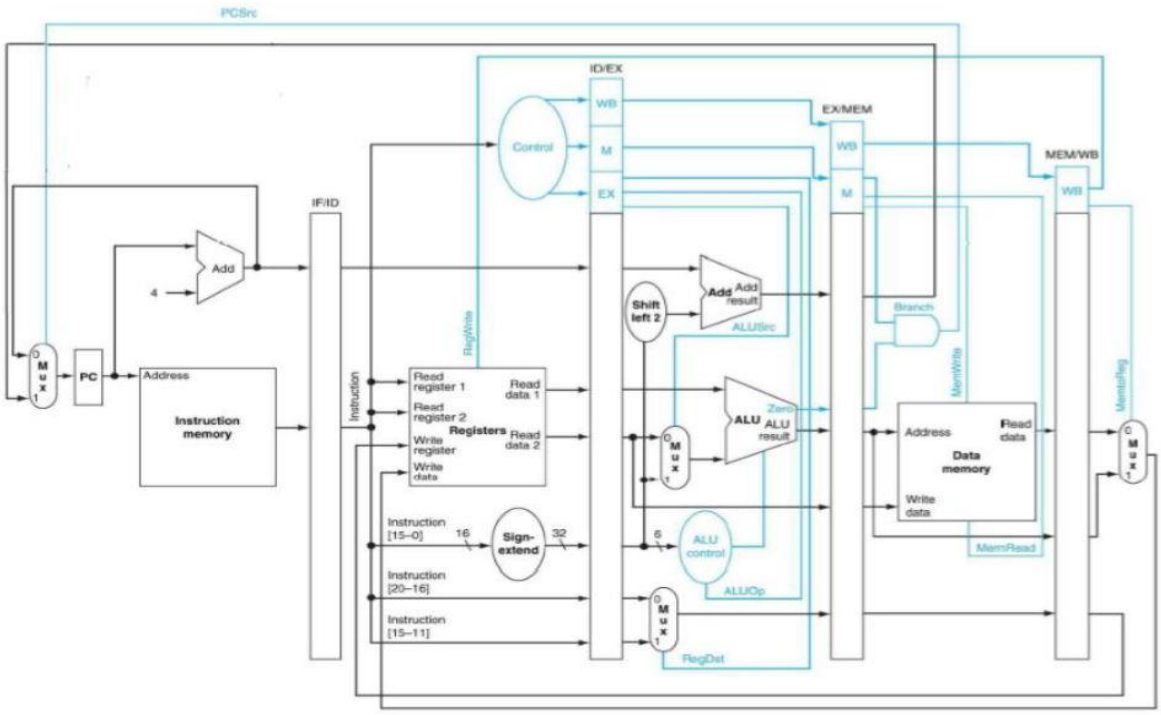
```
Pipe_Reg #(.size(107)) EX_MEM(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .data_i({alu_result, adder_out2, write_Reg_address, ReadData2_ex,
                RegWrite_ex, Branch_ex, MemRead_ex, MemWrite_ex, MemtoReg_ex, ALU_zero}),
        .data_o({alu_result_mem, adder_out2_mem, write_Reg_address_mem, ReadData2_mem,
                RegWrite_mem, Branch_mem, MemRead_mem, MemWrite_mem, MemtoReg_mem, ALU_zero_mem})
);
```

用來連接EX/MEM stage 的 pipelined register

```
Pipe_Reg #(.size(71)) MEM_WB(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .data_i({MemRead_data, alu_result_mem, write_Reg_address_mem, MemtoReg_mem, RegWrite_mem}),
        .data_o({MemRead_data_wb, alu_result_wb, write_Reg_address_wb, MemtoReg_wb, RegWrite_wb})
);
```

用來連接MEM/WB stage 的 pipelined register

2. Your architecture



3. Hardware module analysis

跟Lab3的一樣， 只多了Pipe_Reg和Pipe_CPU

Adder: 使用**2**個**Adder**

(i) 負責計算**PC+4**

(ii) 負責計算**imm**

ALU: 負責邏輯及加減運算

ALU_Ctrl: 負責**ALU**的**control signal**

Data_Memory: 負責存放**data**的**memory**

Decoder: 針對不同的**operation,** 給予對應的**control signal**，個人認為最重要的核心，要傳到不同階段的Pipe_Reg

Instr_Memory: 負責解讀**instruction**

MUX_2to1: 使用**4**個**2 x 1 MUX**

(I) 判斷**Write Register**的值

(II) 檢查**R –**format 還是I -format

(III) 判斷Write Data 的值

(IV) 檢查是否執行**branch**

ProgramCounter: 了解目前的**PC**

Reg_File: 拆解**operation**里要用到的**register**，并存到對應**register**

Shift_Left_Two_32: 向左移**2**個**bit**

Sign_Extend: 延伸**leftmost bit**

Pipe_CPU_1: 不同階段的data存到Pipe_Reg并連起來

Pipe_Reg: 使用4個Pipe_Reg

(I) 存 IF/ID stage

(II) 存 ID/EX stage

(III) 存 EX/MEM stage

(IV) 存 MEM/WB stage

4. Problems you met and solutions

沒什麼問題，就是把Lab3的線路重新接過，再加入MULT

5. Summary

了解簡單的 Pipelined CPU 及 可能面對的 hazard

6. Bonus

(i) I1/I2 data hazard

I2 和 I3 的順序交換，再加上 1 個 NOP

(ii) I5/I6, I8/I9

可以通過 reorder instruction 同時解決

Modified Machine Code:

00100000000000010000000000010000

00100000000000110000000000001000

00000000000000000000000000000000

00100000001000100000000000000100

10101100000000010000000000000100

10001100000001000000000000000100

00100000001001110000000000001010

00000000011000010011000000100000

00000000100000110010100000100010

00000000111000110100000000100100

00100000000010010000000001100100