Computer Organization

1. The input fields of each pipeline register:

```
// IF/ID
wire [32-1:0] ID_PC_add1, ID_instr;

Pipeline_Register #(.size(64)) IF_ID(
    .clk_i(clk_i),
    .rst_i(rst_n),
    .data_i({PC_add1, instr}),
    .data_o({ID_PC_add1, ID_instr})
);
```

```
// ID/EX
wire [32-1:0] EX_PC_add1, EX_signextend, EX_ReadData1, EX_ReadData2;
wire EX_ALUSrc, EX_Branch, EX_MemRead, EX_MemWrite, EX_RegWrite;
wire [3-1:0] EX_ALUOP;
wire [21-1:0] EX_instr_20_0;
wire [2-1:0] EX_RegDst, EX_MemtoReg;

Pipeline_Register #(.size(161)) ID_EX(
    .clk_i(clk_i),
    .rst_i(rst_n),
    .data_i({ID_PC_add1, signextend, ReadData1, ReadData2, ALUSrc, Branch, MemRead, MemWrite, RegWrite,
             MemtoReg, ALUOP, ID_instr[20:0], RegDst}),
    .data_o({EX_PC_add1, EX_signextend, EX_ReadData1, EX_ReadData2, EX_ALUSrc, EX_Branch,
             EX_MemRead, EX_MemWrite, EX_RegWrite, EX_MemtoReg, EX_ALUOP, EX_instr_20_0, EX_RegDst})
    );
```

```
// EX/MEM
wire [5-1:0] MEM_WriteReg_addr;
wire [32-1:0] MEM_PC_add2, MEM_ALUResult, MEM_ReadData2;
wire MEM_Branch, MEM_zero, MEM_MemWrite, MEM_RegWrite, MEM_MemRead;
wire [2-1:0] MEM_MemtoReg;

Pipeline_Register #(.size(108)) EX_MEM(
    .clk_i(clk_i),
    .rst_i(rst_n),
    .data_i({WriteReg_addr, PC_add2, ALUResult, EX_ReadData2, EX_Branch, zero, EX_MemWrite, EX_RegWrite, EX_MemRead, EX_MemtoReg}),
    .data_o({MEM_WriteReg_addr, MEM_PC_add2, MEM_ALUResult, MEM_ReadData2, MEM_Branch,
             MEM_zero, MEM_MemWrite, MEM_RegWrite, MEM_MemRead, MEM_MemtoReg})
);
```

```
// MEM/WB
wire [5-1:0] WB_WriteReg_addr;
wire [32-1:0] WB_DM_ReadData, WB_ALUResult;
wire WB_RegWrite;
wire [2-1:0] WB_MemtoReg;

Pipeline_Register #(.size(72)) MEM_WB(
    .clk_i(clk_i),
    .rst_i(rst_n),
    .data_i({MEM_WriteReg_addr, DM_ReadData, MEM_ALUResult, MEM_RegWrite, MEM_MemtoReg}),
    .data_o({WB_WriteReg_addr, WB_DM_ReadData, WB_ALUResult, WB_RegWrite, WB_MemtoReg})
);
```

2. Compared with lab4, the extra modules:

Pipeline_Register.v

```
module Pipeline_Register( clk_i, rst_i, data_i, data_o);

// I/O ports
input clk_i, rst_i;
input [size-1:0] data_i;
output reg [size-1:0] data_o;

parameter size = 0;

// Writing data when postive edge clk_i
always @(negedge rst_i or posedge clk_i) begin
    if (~rst_i)
        data_o <= 0;
    else
        data_o <= data_i;
end

endmodule
```

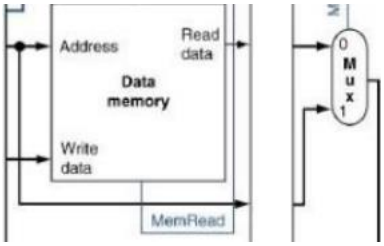
Pipeline_Register，用來傳送在下一階段(或以後)需要用到的值。

3. Explain your control signals in sixth cycle (both test patterns C0_P5_test_data1 and C0_P5_test_data2 are needed):

Picture:

C0_P5_test_data1	C0_P5_test_data2
Regwrite = 1	Regwrite = 1
MemtoReg = 0	MemtoReg = 0
MemWrite = 0	MemWrite = 0
MemRead = 0	MemRead = 0
Branch = 0	Branch = 0
ALUSrc = 0	ALUSrc = 1
ALUOp0 = 0	ALUOp0 = 1
ALUOp1 = 1	ALUOp1 = 1
RegDst = 1	RegDst = 0

4. Problems you met and solutions:



在 spec 架構圖上，WB 階段的 multiplexer 處理方式和前幾次作業不同，需要調整 decoder 的 control signal 或調整接線位置。

5. Summary:

本次作業模擬未處理 hazard 的 pipeline CPU.