Computer Organization

0316056 蔡妮芳 0316038 石瑾旋

A. Source code and the note:

```
module Pipe_CPU_1(
            clk_i,
            rst_i
            );

//Instantiate the components in IF stage
MUX_2to1 #(.size(32)) Mux_Return(
            .data0_i(add_1_o),
            .data1_i(add_2_exmem),
            .select_i(branch_mux),
            .data_o(pc_in_i)
                  );
ProgramCounter PC(
            .clk_i(clk_i),
            .rst_i (rst_i),
            .pc_in_i(pc_in_i),
            .pc_out_o(pc_out_o)
          );
Instruction_Memory IM(
            .addr_i(pc_out_o),
            .instr_o(instr_o)
            );
Adder Add_pc(
          .src1_i(pc_out_o),
          .src2_i(32'd4),
          .sum_o(add_1_o)
            );
Pipe_Reg #(.size(64)) IF_ID(          //N is the total length of input/output
            .clk_i(clk_i),
            .rst_i(rst_i),
            .data_i({add_1_o,instr_o}),
            .data_o({addpc_ifid,instr_ifid});
```

```
//Instantiate the components in ID stage
Reg_File RF(
            .clk_i(clk_i),
            .rst_i(rst_i),
            .RSaddr_i(instr_ifid[25:21]),
            .RTaddr_i(instr_ifid[20:16]),
            .RDaddr_i(writereg_memwb),
            .RDdata_i(RegSource_o),
            .RegWrite_i(RegWrite_memwb),
            .RSdata_o(RSdata_o),
            .RTdata_o(RTdata_o)
            );
Decoder Control(
            .instr_op_i(instr_ifid[31:26]),
            .RegWrite_o(RegWrite_o),
            .ALU_op_o(ALU_op_o),
            .ALUSrc_o(ALUSrc_o),
            .RegDst_o(RegDst_o),
            .Branch_o(Branch_o),
            .BranchType_o(BranchType_o),
            .SinExt_o(SinExt),
            .MemToReg_o(MemToReg_o),
            .Jump_o(Jump_o),
            .MemRead_o(MemRead_o),
            .MemWrite_o(MemWrite_o)
            );

Shift_Left_Two_28 J_Shifter(
           .data_i(instr_ifid[25:0]),
           .data_o(jump_shift_left_two_o)
           );

assign jump_address = {addpc_ifid[31:28],jump_shift_left_two_o[27:0]};

Sign_Extend Sign_Extend(
            .data_i(instr_ifid[15:0]),
            .select_i(SinExt),
            .data_o(Sign_Extend_o));
```

```
Pipe_Reg #(.size(184)) ID_EX(
		.clk_i(clk_i),
		.rst_i(rst_i),
		.data_i({instr_ifid[15:11], instr_ifid[20:16], Sign_Extend_o, RTdata_o, RSdata_o,
		addpc_ifid, RegWrite_o, ALU_op_o, ALUSrc_o, RegDst_o, Branch_o,
		BranchType_o, MemToReg_o, Jump_o, MemRead_o, MemWrite_o,
		jump_address}),
		.data_o({RD_idex, RT_idex, SignExt_idex, RTdata_idex, RSdata_idex, addpc_idex,
		RegWrite_idex, ALU_op_idex, ALUSrc_idex, RegDst_idex, Branch_idex,
		BranchType_idex, MemToReg_idex, Jump_idex, MemRead_idex, MemWrite_idex,
		jump_addr_idex})
		);

//Instantiate the components in EX stage
Shift_Left_Two_32 Shifter(
		.data_i(SignExt_idex),
		.data_o(shift_left_two_o)
		);
Adder Adder2(
		.src1_i(addpc_idex),
		.src2_i(shift_left_two_o),
		.sum_o(add_2_o)
		);
ALU ALU(
		.src1_i(RSdata_idex),
		.src2_i(Mux_ALU_o),
		.ctrl_i(ALUCtrl_o),
		.srl_i(SignExt_idex[10:6]),
		.result_o(ALU_res),
		.zero_o(zero_o),
		.jr_addr(jr_addr)
		);
ALU_Ctrl ALU_Control(
		.funct_i(SignExt_idex[5:0]),
		.ALUOp_i(ALU_op_idex),
		.ALUCtrl_o(ALUCtrl_o),
		.Jr_o(Jr_o)
		);
```

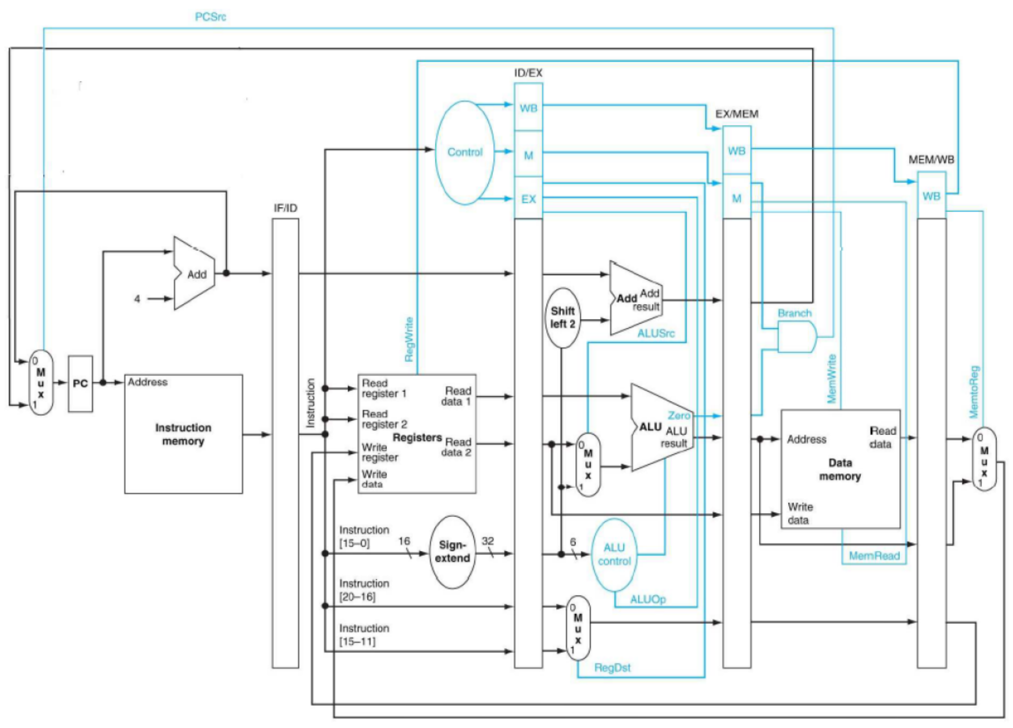
```
MUX_2to1 #(.size(32)) Mux_ALUSrc(
                .data0_i(RTdata_idex),
                .data1_i(SignExt_idex),
                .select_i(ALUSrc_idex),
                .data_o(Mux_ALU_o)
               );
MUX_2to1 #(.size(5)) Mux_Write_Reg(
                .data0_i(RT_idex),
                .data1_i(RD_idex),
                .select_i(RegDst_idex),
                .data_o(mux_writereg)
               );
Pipe_Reg #(.size(208)) EX_MEM(
                .clk_i(clk_i),
                .rst_i(rst_i),
                .data_i({mux_writereg, Jr_o, SignExt_idex, RTdata_idex, ALU_res, jr_addr, zero_o,
                add_2_o, jump_addr_idex, RegWrite_idex, MemToReg_idex, Jump_idex,
                Branch_idex, BranchType_idex,  MemRead_idex, MemWrite_idex}),
                .data_o({writereg_exmem, Jr_exmem, SignExt_exmem, RTdata_exmem,
                ALUres_exmem, jr_addr_exmem, zero_exmem, add_2_exmem,
                jump_addr_exmem, RegWrite_exmem, MemToReg_exmem, Jump_exmem,
                Branch_exmem, BranchType_exmem, MemRead_exmem, MemWrite_exmem})
                );

//Instantiate the components in MEM stage
Data_Memory DM(
                 .clk_i(clk_i),
                 .addr_i(ALUres_exmem),
                 .data_i(RTdata_exmem),
                 .MemRead_i(MemRead_exmem),
                 .MemWrite_i(MemWrite_exmem),
                 .data_o(ReadData_o)
               );
assign branch_mux=Branch_exmem & zero_exmem;
```

```
Pipe_Reg #(.size(104)) MEM_WB(
            .clk_i(clk_i),
            .rst_i(rst_i),
           .data_i({writereg_exmem, ALUres_exmem, SignExt_exmem, ReadData_o,
           RegWrite_exmem, MemToReg_exmem}),
            .data_o({writereg_memwb,ALUres_memwb,SignExt_memwb,ReadData_memwb,
           RegWrite_memwb,MemToReg_memwb})
            );

//Instantiate the components in WB stage
MUX_3to1 Mux3(
               .data0_i(ALUres_memwb),
               .data1_i(ReadData_memwb),
               .data2_i(SignExt_memwb),
               .select_i(MemToReg_memwb),
               .data_o(RegSource_o)
           );
```

B. Implement and hardware module analysis:



Basic 的部分就是照著上面的圖來接線，其中我們在 WB stage 的 mux 是用 3to1，是根據 LAB3 完成的部分所加上去的。在 ID stage 我們還加了一個 shifter，是為了 jump 指令所設計的。

C. Finished part

我們完成 basic 的部分。

D. Problems met and solutions:

做完 basic 後，我們發現我們需要的 clock cycle 大於 test bench 裡預訂的 30 ，至少需要 40 個 clock cycle 才可以執行完整個運算。

後來我們發現問題出在我們是直接用 lab3 添加 pipeline register 改成 lab4，由於 lab3 我們有做 bonus，不同 branch type 的部分，故多了一些 module 在 cpu 裡，使得無法在 30 cycle 裡及時完成。把多餘的 module 拿掉後就成功了。

E. Summary:

這次作業主要是實作 pipeline，參考老師的附圖可以很簡單的完成，只要加上四個 pipeline register (IF/ID, ID/EX, EX/MEM, MEM/WB)，要注意這些 register 的 output 要分別接到哪個階段(WB or M or EX)，便可正確控制其他 hardware。