Lab5 report

Detailed description of the implementation

Adder.v

- 同 lab4。

ALU_Ctrl.v

- 多數同 lab4。
- 新增
    - less_i
        - `ALUOP` == 00 or 10
        - `instr` == 0010
        - `ALU_Ctrl_o` = 0111 (less)
    - lw
        - `ALUOP` == 11
        - `instr` == 0010
        - `ALU_Ctrl_o` = 0010 (add)
    - shift_left_i
        - `ALUOP` == 00 or 10
        - `instr` == 0001
        - `ALU_Ctrl_o` = 0101 (shift left)

```
else if(ALUOp == 2'b00 & instr == 4'b0010)//less_i
    ALU_Ctrl_o = 4'b0111;
else if(ALUOp == 2'b11 & instr == 4'b0010)//lw
    ALU_Ctrl_o = 4'b0010;
else if(ALUOp == 2'b00 & instr == 4'b0001)//shift_left_i
    ALU_Ctrl_o = 4'b0101;
```

alu.v

- 同 lab4。

Decoder.v

	rtype	itype	lw	store	branch	jal	jalr
ALUSrc	0	1	1	1	0	0	0
MemtoReg	0	0	1	1	0	0	0
RegWrite	1	1	1	0	0	1	1
MemRead	0	0	1	0	0	0	0
MemWrite	0	0	0	1	0	0	0
Branch	0	0	0	0	1	0	0
ALUOp	10	00	11	00	01	11	11
Jump	0	0	0	0	0	1	1

note: 如果不是上述指令，要把所有 control bits 設成 0。

```
assign opcode = instr_i[6:0];
always@(*)begin
  if(opcode == 7'b0110011)begin//rtype
    ALUSrc = 1'b0;//rs2
    MemtoReg = 1'b0;
    RegWrite = 1'b1;
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b10;
    Jump = 1'b0;
  end
  else if(opcode == 7'b0010011)begin//itype
    ALUSrc = 1'b1;//imm
    MemtoReg = 1'b0;
    RegWrite = 1'b1;
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b00;
    Jump = 1'b0;
  end
  else if(opcode == 7'b0000011)begin//lw
    ALUSrc = 1'b1;//imm
    MemtoReg = 1'b1;
    RegWrite = 1'b1;
    MemRead = 1'b1;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b11;
    Jump = 1'b0;
  end
  else if(opcode == 7'b0100011)begin//store
    ALUSrc = 1'b1;//imm
    MemtoReg = 1'b1;
    RegWrite = 1'b0;
    MemRead = 1'b0;
    MemWrite = 1'b1;
    Branch = 1'b0;
    ALUOp = 2'b00;
    Jump = 1'b0;
  end
  else if(opcode == 7'b1100011)begin//branch
    ALUSrc = 1'b0;//rs2
    MemtoReg = 1'b0;
    RegWrite = 1'b0;
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b1;
    ALUOp = 2'b01;
    Jump = 1'b0;
  end
  else if(opcode == 7'b1101111)begin//jal
    ALUSrc = 1'b0;//rs2
    MemtoReg = 1'b0;
    RegWrite = 1'b1;//write pc+4
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b11;
    Jump = 1'b1;
  end
  else if(opcode == 7'b1100111)begin//jalr
    ALUSrc = 1'b0;//rs2
    MemtoReg = 1'b0;
    RegWrite = 1'b1;//write pc+4
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b11;
    Jump = 1'b1;
  end
  else begin // set zero
    ALUSrc = 1'b0;//rs2
    MemtoReg = 1'b0;
    RegWrite = 1'b0;//write pc+4
    MemRead = 1'b0;
    MemWrite = 1'b0;
    Branch = 1'b0;
    ALUOp = 2'b11;
    Jump = 1'b0;
  end
end
```

ForwardingUnit.v

根據 input，控制 output `ForwardA`, `ForwardB`，以決定要不要 forward。

- input
  - IDEXE_RS1
  - IDEXE_RS2
  - EXEMEM_RD
  - MEMWB_RD
  - EXEMEM_RegWrite
  - MEMWB_RegWrite
- output
  - ForwardA
  - ForwardB
- 會發生 data hazard 的狀況為
  1. 需要寫入
     - RegWrite == 1
  2. rd 不為 0
  3. 後面的 rd 跟前面的 rs 相同
- 統整
  - EX Hazard
    - `IDEXE_RS1 == EXEMEM_RD && EXEMEM_RegWrite == 1'b1 && EXEMEM_RD != 1'b0`
    - `IDEXE_RS2 == EXEMEM_RD && EXEMEM_RegWrite == 1'b1 && EXEMEM_RD != 1'b0`
    - `ForwardA` = `ForwardB` = 01
  - MEM Hazard
    - `IDEXE_RS1 == MEMWB_RD && MEMWB_RegWrite == 1'b1 && MEMWB_RD != 1'b0`
    - `IDEXE_RS2 == MEMWB_RD && MEMWB_RegWrite == 1'b1 && MEMWB_RD != 1'b0`
    - `ForwardA` = `ForwardB` = 10
  - else
    - `ForwardA` = `ForwardB` = 00

```
always@(*)begin
  if(IDEXE_RS1 == EXEMEM_RD && EXEMEM_RegWrite == 1'b1 && EXEMEM_RD != 1'b0)
```

```
    ForwardA = 2'b01;
  else if(IDEXE_RS1 == MEMWB_RD && MEMWB_RegWrite == 1'b1 && MEMWB_RD != 1'b0)
    ForwardA = 2'b10;
  else
    ForwardA = 2'b00;
end

always@(*)begin
  if(IDEXE_RS2 == EXEMEM_RD && EXEMEM_RegWrite == 1'b1 && EXEMEM_RD != 1'b0)
    ForwardB = 2'b01;
  else if(IDEXE_RS2 == MEMWB_RD && MEMWB_RegWrite == 1'b1 && MEMWB_RD != 1'b0)
    ForwardB = 2'b10;
  else
    ForwardB = 2'b00;
end
```

Hazard_detection.v

- input
  - IFID_regRs
  - IFID_regRt
  - IDEXE_regRd
  - IDEXE_memRead
- output
  - PC_write
  - IFID_write
  - control_output_select
- load use hazard 發生條件為
  - `IDEXE_memRead == 1 && IDEXE_regRd != 0`
    - 有 load，且有寫到 rd
  - `(IDEXE_regRd == IFID_regRs) || (IDEXE_regRd == IFID_regRt)`
    - load 的 rd 是後面人的 rs 或 rt
  - 成立則 output 都設為 0
  - 否則 output 都設為 1

```
always@(*)begin
  if(IDEXE_memRead && ((IDEXE_regRd == IFID_regRs) || (IDEXE_regRd == IFID_regRt)) && IDEXE_regRd != 0)
  begin
    PC_write = 1'b0;
    IFID_write = 1'b0;
    control_output_select = 1'b0;//??
  end
  else
  begin
    PC_write = 1'b1;
    IFID_write = 1'b1;
    control_output_select = 1'b1;
  end
end
```

Imm_Gen.v

- 同 lab4。

MUX_2to1.v

```
always@(*)begin
  if(select_i == 1'b1)
    data_o = data1_i;
  else
    data_o = data0_i;
end
```

MUX_3to1.v

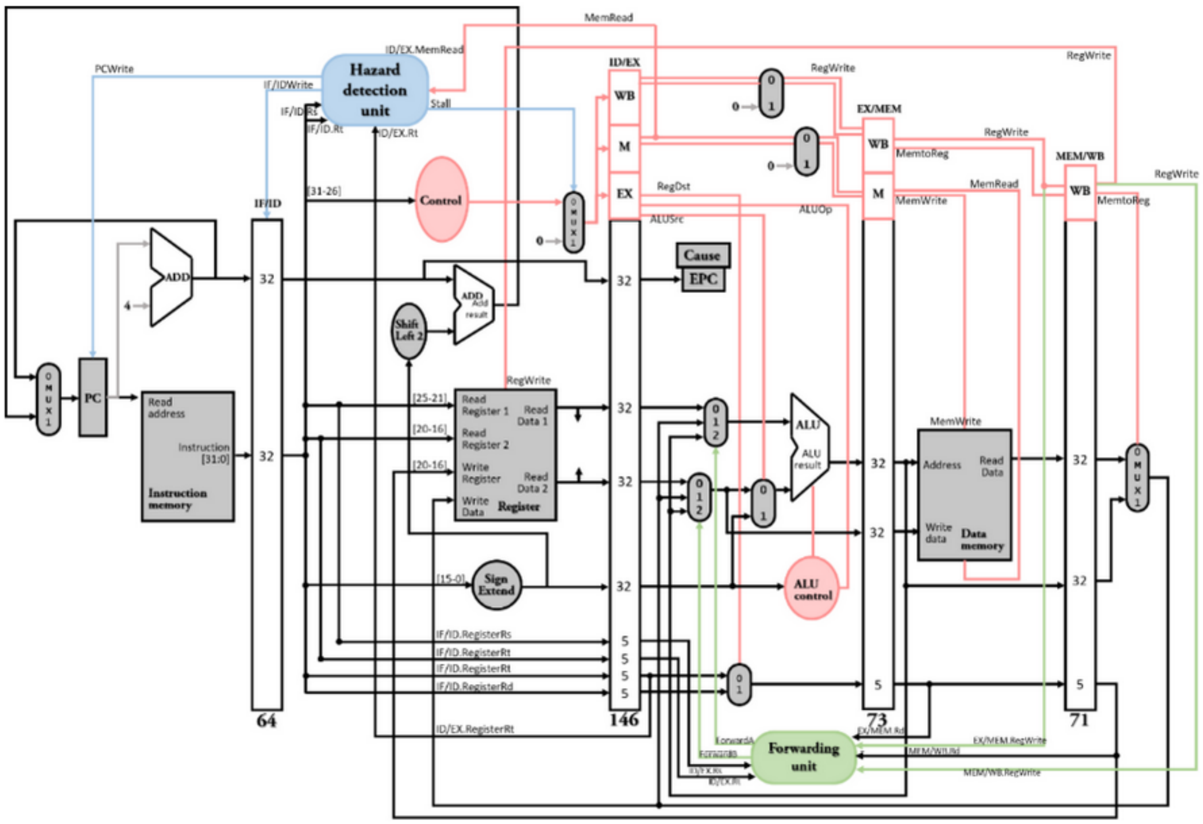
```
always@(*)begin
  if(select_i == 2'b01)
    data_o = data1_i;
  else if(select_i == 2'b10)
    data_o = data2_i;
  else
    data_o = data0_i;
end
```

Shift_Left_1.v

- Shift left by 1

```
always@(*)begin
  data_o = data_i<<1;
end
```

Pipeline_CPU.v



使用到的 wire 與 assign 其值：

```
assign IFID_Flush = (Branch && (RSdata_o == RTdata_o)) || Jump;//#

wire [31:0] instr;//#
wire control_output_select;//#
wire [31:0] MUX_control_o;//#
wire [31:0] MUX_control_1i;//#
wire [3:0] alu_ctrl_instr;
wire jalr_select;
wire [31:0] branch_src_o;
wire imm_or_src;
wire [31:0] ALU_src_2to1_i;
//decoder
wire [31:0] shift_left_o;//#


wire [1:0] MUX_MemtoReg_select;
assign MUX_MemtoReg_select[1] = MEMWB_WB_o[0];//??
assign MUX_MemtoReg_select[0] = MEMWB_WB_o[1];//??

assign MUX_control_1i[31:30] = ALUOp;//EX//[2:1]
assign MUX_control_1i[29] = ALUSrc;//EX//[0]
assign MUX_control_1i[28] = MemRead;//M[1]
assign MUX_control_1i[27] = MemWrite;//M[0]
assign MUX_control_1i[26] = RegWrite;//WB[2]
assign MUX_control_1i[25] = MemtoReg;//WB[1]
assign MUX_control_1i[24] = Jump;//WB[0]
assign MUX_control_1i[23:0] = 24'd0;

wire [2:0] EX;
wire [1:0] M;
wire [2:0] WB;
assign EX[2:1] = ALUOp;//(not_stall == 1'b1)? ALUOp:2'b00;
assign EX[0] = ALUSrc;//(not_stall == 1'b1)? ALUSrc:1'b0;
assign M[1] = MemRead;//(not_stall == 1'b1)? MemRead:1'b0;
assign M[0] = MemWrite;//(not_stall == 1'b1)? MemWrite:1'b0;
assign WB[2] = RegWrite;//(not_stall == 1'b1)? RegWrite:1'b0;
assign WB[1] = MemtoReg;//(not_stall == 1'b1)? MemtoReg:1'b0;
assign WB[0] = Jump;//(not_stall == 1'b1)? Jump:1'b0;

assign alu_ctrl_instr[3] = IFID_Instr_o[30];
assign alu_ctrl_instr[2:0] = IFID_Instr_o[14:12];

assign jalr_select = (Jump && (IFID_Instr_o[6:0] == 7'b1100111));
assign imm_or_src = (IDEXE_Instr_o[6:0] == 7'b0010011) || (IDEXE_Instr_o[6:0] == 7'b0000011) || (IDEXE_Instr_o[6:0] == 7'b0100011);

wire [2:0] EXMEM_mem_i;
assign EXMEM_mem_i[2:1] = IDEXE_Mem_o;
assign EXMEM_mem_i[0] = 1'b0;
```

如上圖，接上各個 module，以組成 pipeline CPU。

- pipeline 的 `rst_i` 是來自 testbench 中的 `rst_n`，當各個 module 要 reset 時，`rst_i` input 應為 1，故在 pipeline.v 中，除了 `ref_file`，其餘的 `rst_i` input 為 `~rst_i`。
- IF
  - Mux for PC source

    ```
    MUX_2to1 MUX_PCSrc(
      .data0_i(PC_Add4),
      .data1_i(Imm_Gen_o),
      .select_i(IFID_Flush),
      .data_o(PC_i)
    );
    ```

  - mux for branch source

    ```
    //MUX for branch src//for jalr
    MUX_2to1 Branch_src(
      .data0_i(IFID_PC_o),
      .data1_i(RSdata_o),
      .select_i(jalr_select),
      .data_o(branch_src_o)
    );
    ```

  - PC

    ```
    ProgramCounter PC(
      .clk_i(clk_i),
      .rst_i(~rst_i),//?
      .pc_i(PC_i),
      .PC_write(PC_write),
      .pc_o(PC_o)
    );
    ```

  - adder for PC+4

    ```
    Adder PC_plus_4_Adder(
      .src1_i(PC_o),
      .src2_i(four_tmp),
      .sum_o(PC_Add4)
    );
    ```

  - instruction memory

```
Instr_Memory IM(
  .addr_i(PC_o),
  .instr_o(instr)
);
```

  - data transfer from IF to ID

```
IFID_register IFtoID(
  .clk_i(clk_i),
  .rst_i(~rst_i),
  .flush(IFID_Flush),//??
  .write(IFID_Write),
  .address_i(PC_o),//??
  .instr_i(instr),
  .pc_add4_i(PC_Add4),//for jump instructions
  .address_o(IFID_PC_o),
  .instr_o(IFID_Instr_o),
  .pc_add4_o(IFID_PC_Add4_o)
);
```

- ID
  - hazard detection unit

```
Hazard_detection Hazard_detection_obj(
  .IFID_regRs(IFID_Instr_o[19:15]),
  .IFID_regRt(IFID_Instr_o[24:20]),
  .IDEXE_regRd(IDEXE_Instr_11_7_o),
  .IDEXE_memRead(IDEXE_Mem_o[1]),//??
  .PC_write(PC_write),
  .IFID_write(IFID_Write),
  .control_output_select(control_output_select)//??
);
```

  - mux for control signals

```
MUX_2to1 MUX_control(
  .data0_i(zero_tmp),
  .data1_i(MUX_control_1i),
  .select_i(control_output_select),
  .data_o(MUX_control_o)//the same as MUX_control_1i or all 0
);
```

  - decoder

```
Decoder Decoder(
  .instr_i(IFID_Instr_o),
  .Branch(Branch),
  .ALUSrc(ALUSrc),
  .RegWrite(RegWrite),
  .ALUOp(ALUOp),
  .MemRead(MemRead),
  .MemWrite(MemWrite),
  .MemtoReg(MemtoReg),
  .Jump(Jump)
);
```

  - register file

```
Reg_File RF(
  .clk_i(clk_i),
  .rst_i(rst_i),
  .RSaddr_i(IFID_Instr_o[19:15]),
  .RTaddr_i(IFID_Instr_o[24:20]),
  .RDaddr_i(MEMWB_Instr_11_7_o),
  .RDdata_i(MUXMemtoReg_o),
  .RegWrite_i(MEMWB_WB_o[2]),//??
  .RSdata_o(RSdata_o),
  .RTdata_o(RTdata_o)
);
```

- immediate generator

```
Imm_Gen ImmGen(
  .instr_i(IFID_Instr_o),
  .Imm_Gen_o(Imm_Gen_o)
);
```

- shifit left for immediate

```
Shift_Left_1 SL1(
  .data_i(Imm_Gen_o),
  .data_o(shift_left_o)
);
```

- adder for branch PC

```
Adder Branch_Adder(
  .src1_i(branch_src_o),
  .src2_i(shift_left_o),
  .sum_o(PC_Add_Immediate)
);
```

- data transfer from ID to EXE

```
IDEXE_register IDtoEXE(
  .clk_i(clk_i),
  .rst_i(~rst_i),
  .instr_i(IFID_Instr_o),
  .WB_i(WB),//MUX_control_o[26:24]),
  .Mem_i(M),//MUX_control_o[28:27]),
  .Exe_i(EX),//MUX_control_o[31:29]),
  .data1_i(RSdata_o),
  .data2_i(RTdata_o),
  .immgen_i(Imm_Gen_o),
  .alu_ctrl_instr(alu_ctrl_instr),
  .WBreg_i(IFID_Instr_o[11:7]),//rd address
  .pc_add4_i(IFID_PC_Add4_o),

  .instr_o(IDEXE_Instr_o),
  .WB_o(IDEXE_WB_o),
  .Mem_o(IDEXE_Mem_o),//2bit
  .Exe_o(IDEXE_Exe_o),
  .data1_o(IDEXE_RSdata_o),
  .data2_o(IDEXE_RTdata_o),
  .immgen_o(IDEXE_ImmGen_o),
  .alu_ctrl_input(IDEXE_Instr_30_14_12_o),
  .WBreg_o(IDEXE_Instr_11_7_o),
  .pc_add4_o(IDEXE_PC_add4_o)
);
```

- EXE
  - mux for alu source 2

```
MUX_2to1 MUX_ALUSrc(//imm or src2?
  .data0_i(ALU_src_2to1_i),//src2
  .data1_i(IDEXE_ImmGen_o),//imm
  .select_i(IDEXE_Exe_o[0]),//ALUSrc
  .data_o(ALUSrc2_o)
);
```

- forwarding unit

```
ForwardingUnit FWUnit(
  .IDEXE_RS1(IDEXE_Instr_o[19:15]),
  .IDEXE_RS2(IDEXE_Instr_o[24:20]),
  .EXEMEM_RD(EXEMEM_Instr_11_7_o),
  .MEMWB_RD(MEMWB_Instr_11_7_o),
  .EXEMEM_RegWrite(EXEMEM_WB_o[2]),
  .MEMWB_RegWrite(MEMWB_WB_o[2]),
  .ForwardA(ForwardA),
  .ForwardB(ForwardB)
);
```

- mux for alu source 1

```
MUX_3to1 MUX_ALU_src1(
  .data0_i(IDEXE_RSdata_o),
  .data1_i(EXEMEM_ALUResult_o),
  .data2_i(MUXMemtoReg_o),
  .select_i(ForwardA),
  .data_o(ALUSrc1_o)
);
```

- mux for “mux for alu source 2”

```
MUX_3to1 MUX_ALU_src2(
  .data0_i(IDEXE_RTdata_o),
  .data1_i(EXEMEM_ALUResult_o),
  .data2_i(MUXMemtoReg_o),
  .select_i(ForwardB),
  .data_o(ALU_src_2to1_i)
);
```

- alu control

```
ALU_Ctrl ALU_Ctrl(
  .instr(IDEXE_Instr_30_14_12_o),
  .ALUOp(IDEXE_Exe_o[2:1]),
  .ALU_Ctrl_o(ALU_Ctrl_o)
);
```

- alu

```
alu alu(
  .rst_n(~rst_i),
  .src1(ALUSrc1_o),
  .src2(ALUSrc2_o),
  .ALU_control(ALU_Ctrl_o),
  .result(ALUResult),
  .zero(ALU_zero)
);
```

- data transfer from EXE to MEM

```
EXEMEM_register EXEtoMEM(
  .clk_i(clk_i),
  .rst_i(~rst_i),
  .instr_i(IDEXE_Instr_o),
  .WB_i(IDEXE_WB_o),
  .Mem_i(EXMEM_mem_i),//0 bit is 0
  .zero_i(ALU_zero),
  .alu_ans_i(ALUResult),
  .rtdata_i(IDEXE_RTdata_o),
  .WBreg_i(IDEXE_Instr_11_7_o),
  .pc_add4_i(IDEXE_PC_add4_o),

  .instr_o(EXEMEM_Instr_o),
  .WB_o(EXEMEM_WB_o),
  .Mem_o(EXEMEM_Mem_o),
  .zero_o(EXEMEM_Zero_o),
  .alu_ans_o(EXEMEM_ALUResult_o),
  .rtdata_o(EXEMEM_RTdata_o),
  .WBreg_o(EXEMEM_Instr_11_7_o),
  .pc_add4_o(EXEMEM_PC_Add4_o)
);
```

- MEM
  - data memory

```
Data_Memory Data_Memory(
  .clk_i(clk_i),
  .addr_i(EXEMEM_ALUResult_o),
  .data_i(EXEMEM_RTdata_o),
  .MemRead_i(EXEMEM_Mem_o[2]),
  .MemWrite_i(EXEMEM_Mem_o[1]),
  .data_o(DM_o)
);
```

  - data transfer from MEM to WB

```
MEMWB_register MEMtoWB(
  .clk_i(clk_i),
  .rst_i(~rst_i),
  .WB_i(EXEMEM_WB_o),
  .DM_i(DM_o),
  .alu_ans_i(EXEMEM_ALUResult_o),
  .WBreg_i(EXEMEM_Instr_11_7_o),
  .pc_add4_i(EXEMEM_PC_Add4_o),

  .WB_o(MEMWB_WB_o),
  .DM_o(MEMWB_DM_o),
  .alu_ans_o(MEMWB_ALUresult_o),
  .WBreg_o(MEMWB_Instr_11_7_o),
  .pc_add4_o(MEMWB_PC_Add4_o)
);
```

- WB
  - mux for mem to reg

```
MUX_3to1 MUX_MemtoReg(//jump: write pc's address?
  .data0_i(MEMWB_ALUresult_o),//alu result
  .data1_i(MEMWB_DM_o),//data from mem
  .data2_i(MEMWB_PC_Add4_o),//pc + 4
  .select_i(MUX_MemtoReg_select),
  .data_o(MUXMemtoReg_o)
);
```

IFID_register.v

把前面的資料通通搬過來，包括：

- input from pipeline
    - IFID_Flush
    - IFID_Write
    - PC_o
    - instr
    - PC_Add4
- output
    - address_o
    - instr_o
    - pc_add4_o

```
always@(posedge clk_i)begin
  if(~rst_i)begin
    if(~flush)begin
      if(write)begin
        address_o <= address_i;
        instr_o <= instr_i;
        pc_add4_o <= pc_add4_i;
      end
      else begin
        address_o <= address_o;
        instr_o <= instr_o;
        pc_add4_o <= pc_add4_o;
      end

    end
    else begin
      address_o <= 0;
      instr_o <= 0;
      pc_add4_o <= 0;
    end
  end
  else begin
    address_o <= 0;
    instr_o <= 0;
    pc_add4_o <= 0;
  end
end
```

IDEXE_register.v

把前面的資料通通搬過來，包括：

- input from pipeline
    - IFID_Instr_o
    - WB //MUX_control_o[26:24]
    - M //MUX_control_o[28:27]
    - EX //MUX_control_o[31:29]
    - RSdata_o

    - RTdata_o
    - Imm_Gen_o
    - alu_ctrl_instr
    - IFID_Instr_o[11:7]
    - IFID_PC_Add4_o
- output
    - IDEXE_Instr_o
    - IDEXE_WB_o
    - IDEXE_Mem_o
    - IDEXE_Exe_o
    - IDEXE_RSdata_o
    - IDEXE_RTdata_o
    - IDEXE_ImmGen_o
    - IDEXE_Instr_30_14_12_o
    - IDEXE_Instr_11_7_o
    - IDEXE_PC_add4_o

```
always@(posedge clk_i)begin
  if(~rst_i)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;
  end
  else begin
    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;
  end
end
```

EXEMEM_register.v

把前面的資料通通搬過來，包括：

- input from pipeline
    - IDEXE_Instr_o
    - IDEXE_WB_o
    - EXMEM_mem_i

    - ALU_zero
    - ALUResult
    - IDEXE_RTdata_o
    - IDEXE_Instr_11_7_o
    - IDEXE_PC_add4_o
- output
    - instr_o
    - WB_o
    - Mem_o
    - zero_o
    - alu_ans_o
    - rtdata_o
    - WBreg_o
    - pc_add4_o

```
always@(posedge clk_i)begin
  if(~rst_i)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
  else begin
    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
end
```

MEMWB_register.v

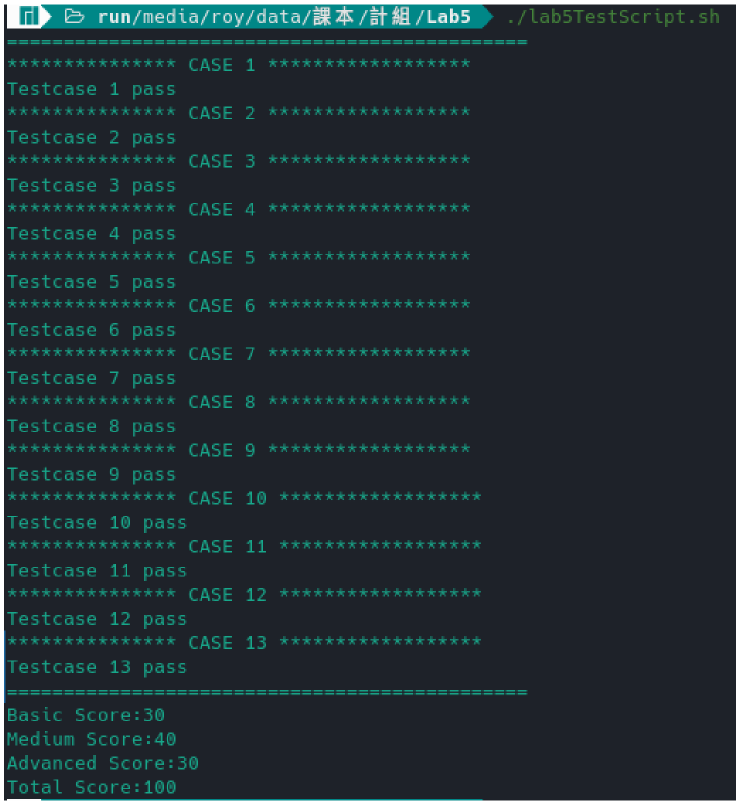
把前面的資料通通搬過來，包括：

- input from pipeline
    - EXEMEM_WB_o
    - DM_o
    - EXEMEM_ALUResult_o
    - EXEMEM_Instr_11_7_o
    - EXEMEM_PC_Add4_o
- output
    - WB_o

- DM_o
- alu_ans_o
- WBreg_o
- pc_add4_o

```
always@(posedge clk_i)begin
  if(~rst_i)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
  else begin
    WB_o <= 0;
    DM_o <= 0;
    alu_ans_o <= 0;
    WBreg_o <= 0;
    pc_add4_o <= 0;
  end
end
```

Implementation results



```
run/media/roy/data/課本/計組/Lab5  ./lab5TestScript.sh
==============================================
*************** CASE 1 ******************
Testcase 1 pass
*************** CASE 2 ******************
Testcase 2 pass
*************** CASE 3 ******************
Testcase 3 pass
*************** CASE 4 ******************
Testcase 4 pass
*************** CASE 5 ******************
Testcase 5 pass
*************** CASE 6 ******************
Testcase 6 pass
*************** CASE 7 ******************
Testcase 7 pass
*************** CASE 8 ******************
Testcase 8 pass
*************** CASE 9 ******************
Testcase 9 pass
*************** CASE 10 ******************
Testcase 10 pass
*************** CASE 11 ******************
Testcase 11 pass
*************** CASE 12 ******************
Testcase 12 pass
*************** CASE 13 ******************
Testcase 13 pass
==============================================
Basic Score:30
Medium Score:40
Advanced Score:30
Total Score:100
```

Problems encountered and solutions

1. 如果 reg_file 是 positive edge，會跟 IFID 同時改，導致使用到 IFID value 時，是用到舊的 IFID value，而不是更新過的，導致結果錯誤。

2. 接 module 時，rst_i 或 ~rst_i 要放對，一開始放反導致資料都讀不到。
3. decoder 在什麼指令都不是時，要歸零，不然之後 control bits 會錯
4. hazard stall 時，要保留原值
5. 助教一開始給的版本沒有 pc_write 跟 IFID_write，都是我們自己加的。

註：此次作業我們在舊版本時就已完成，後來有下載新版本，確認助教新增的 module input / output，我們都事先加了。已確認我們的 code 與新版本相容，如助教有疑慮請再告知，謝謝。

109550168 林慧旻，109550042 林律穎