Computer Organization

The input fields of each pipeline register:

Adder.v
ALU.v
ALU_Ctrl.v
Data_Memory.v
Decoder.v
Instr_Memory.v
Mux2to1.v
Mux3to1.v
Program_Counter.v
Reg_File.v
Shifter.v
Sign_Extend.v
Zero_Filled.v
REG.v

Pipeline_CPU.v // 這個是 pipline CPU 的檔案，要被 Testbench include

Compared with lab3, the extra modules:

我把新增的 module 全部放在 REG.V 這個檔案裏面。
分別為四個佔存器
1.

```
module IF_ID_REG(clk_i, rst_n, instruction_in, instruction_out);
input clk_i;
input rst_n;
input [31:0] instruction_in;

output [31:0] instruction_out;
reg [31:0] instruction_out;

always @(posedge clk_i or negedge rst_n) begin
	if(~rst_n) begin
		instruction_out <= 32'b0;
	end

	else
		begin
			instruction_out <= instruction_in;
		end

end
endmodule
```

2.

```
module ID_EX_REG(clk_i, rst_n, WD_in, WD_out, M_in, M_out, EXE_in,
EXE_out, RS_data_in, RS_data_out, RT_data_in, RT_data_out, SE_in, SE_out,
Zero_in, Zero_out, instr_s_in, instr_s_out, alu_ctr_in, alu_ctr_out, RT_dest_in,
RT_dest_out, RD_dest_in, RD_dest_out);
input clk_i;
input rst_n;
input [1:0] WD_in;
input [1:0] M_in;
input [4:0]    instr_s_in, RD_dest_in, RT_dest_in;
input [5:0] alu_ctr_in;
input [4:0] EXE_in;
input [31:0] RS_data_in, RT_data_in, SE_in, Zero_in;


output [1:0] WD_out;
output [1:0] M_out;
output [4:0] instr_s_out, RD_dest_out, RT_dest_out;
output [5:0] alu_ctr_out;
output [4:0] EXE_out;
output [31:0] RS_data_out, RT_data_out, SE_out, Zero_out;

reg [1:0] WD_out;
reg [1:0] M_out;
reg [4:0] instr_s_out, RD_dest_out, RT_dest_out;
reg [5:0] alu_ctr_out;
reg [4:0] EXE_out;
reg [31:0] RS_data_out, RT_data_out, SE_out, Zero_out;

always @(posedge clk_i or negedge rst_n) begin
      if(~rst_n) begin
            WD_out <= 3'b0;
            M_out <= 3'b0;
            EXE_out <= 7'b0;
            instr_s_out <= 5'b0;
            RD_dest_out <= 5'b0;
            RT_dest_out <= 5'b0;
            alu_ctr_out <= 6'b0;
            RS_data_out <= 32'b0;
            RT_data_out <= 32'b0;
            SE_out <= 32'b0;
            Zero_out <= 32'b0;
      end
      else begin
            WD_out <= WD_in;
            M_out <= M_in;
            EXE_out <= EXE_in;
            instr_s_out <= instr_s_in;
            RD_dest_out <= RD_dest_in;
            RT_dest_out <= RT_dest_in;
```

```
            alu_ctr_out <= alu_ctr_in;
            RS_data_out <= RS_data_in;
            RT_data_out <= RT_data_in;
            SE_out <= SE_in;
            Zero_out <= Zero_in;
        end

end
endmodule
```

3.

```
module EXE_MEM_REG (clk_i, rst_n, WB_in, WB_out, M_in, M_out, ALU_res_in,
ALU_res_out, RT_data_in, RT_data_out, REG_dest_in, REG_dest_out);
input clk_i;
input rst_n;
input [1:0] WB_in;
input [1:0] M_in;
input [31:0] ALU_res_in, RT_data_in;
input [4:0] REG_dest_in;


output [1:0] WB_out;
output [1:0] M_out;
output [31:0] ALU_res_out, RT_data_out;
output [4:0] REG_dest_out;

reg [1:0] WB_out;
reg [1:0] M_out;
reg [31:0] ALU_res_out, RT_data_out;
reg [4:0] REG_dest_out;

always @(posedge clk_i or negedge rst_n) begin
    if(~rst_n) begin
            WB_out <= 3'b0;
            M_out <= 3'b0;
            ALU_res_out <= 32'b0;
            RT_data_out <= 32'b0;
            REG_dest_out <= 5'b0;
    end
    else begin
            WB_out <= WB_in;
            M_out <= M_in;
            ALU_res_out <= ALU_res_in;
            RT_data_out <= RT_data_in;
            REG_dest_out <= REG_dest_in;
    end

end
endmodule
```

4.

```
module M_WB_REG(clk_i, rst_n, WB_in, WB_out, MEM_in, MEM_out,
ALU_res_in, ALU_res_out, REG_dest_in, REG_dest_out);
input clk_i;
input rst_n;
input [1:0] WB_in;
input [31:0] MEM_in;
input [31:0] ALU_res_in;
input [4:0]REG_dest_in;



output [1:0] WB_out;
output [31:0] MEM_out;
output [31:0] ALU_res_out;
output [4:0] REG_dest_out;
reg [1:0] WB_out;
reg [31:0] MEM_out;
reg [31:0] ALU_res_out;
reg [4:0] REG_dest_out;

always @(posedge clk_i or negedge rst_n) begin
	if(~rst_n) begin
		WB_out <= 3'b0;
		MEM_out <= 32'b0;
		ALU_res_out <= 32'b0;
		REG_dest_out <= 32'b0;
	end
	else begin
		WB_out <= WB_in;
		MEM_out <= MEM_in;
		ALU_res_out <= ALU_res_in;
		REG_dest_out <= REG_dest_in;
	end

end



endmodule
```

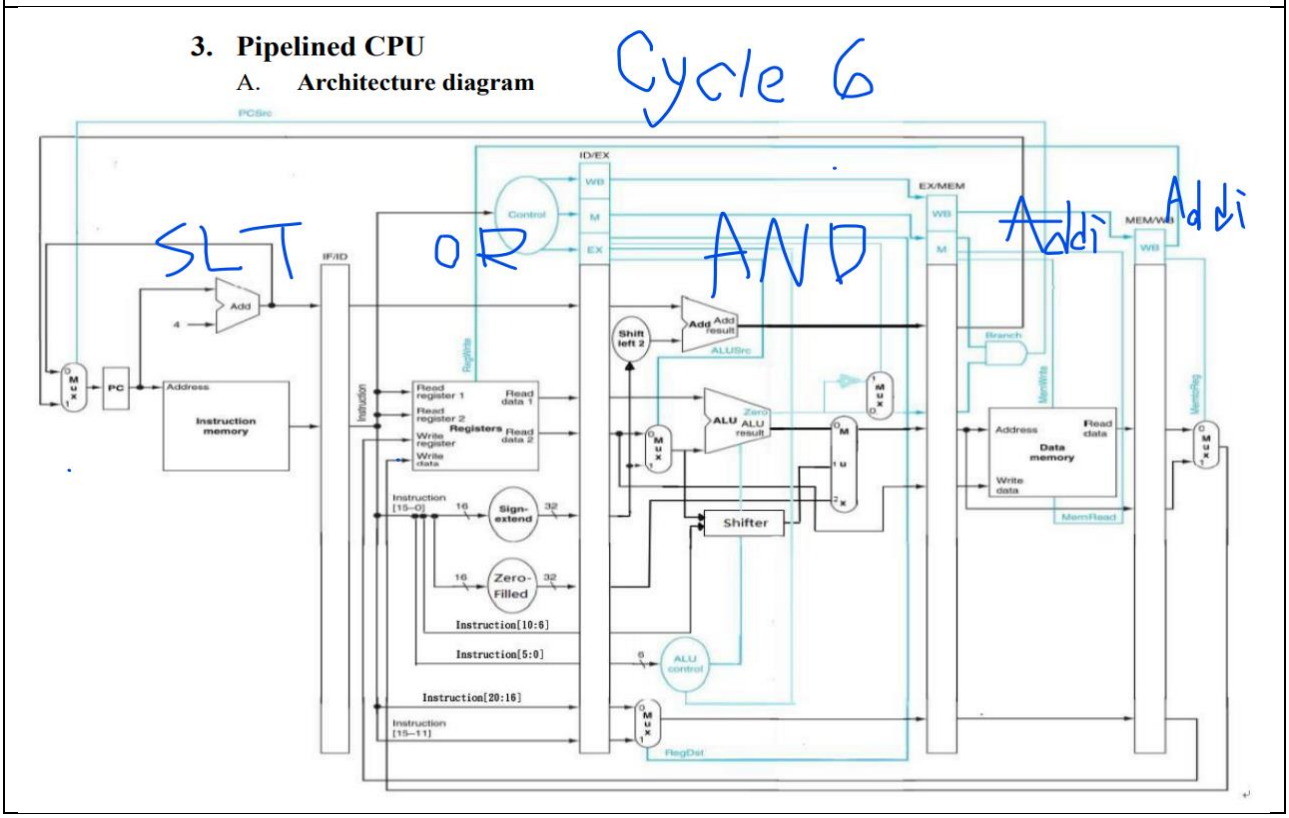
Explain your control signals in sixth cycle (both test data CO_P4_test_data1 and CO_P4_test_data2 are needed)

CO_P4_test_data1:

指令轉換:

ADDI $1, $Zero, 31
ADDI $2, $Zero, 17
ADDI $4, $Zero, 14
AND $3, $5, $6
OR $5, $1, $Zero
SLT $6, $2, $1

Picture:

CO_P5_test_data1


Sixth cycle 的所有指令狀況如上

先看 IF 的部分，
IF 內容沒什麼變化，在 PC+4 的地方因為沒有 Branch 指令，因此我把它拿掉了。剩下 instructio memory 輸出的地方要改到 IF_ID_REG 內儲存。

IF_ID_REG
作為 cycle 與 cycle 之間的間隔佔存，因此必須在內部設 clock 為 posedge。且 Testbench 內 rst_n 一開始為 0，所以將 rst_n 設為 negedge 作為初始化所有值為 0。

```
module IF_ID_REG(clk_i, rst_n, instruction_in, instruction_out);
input clk_i;
input rst_n;
input [31:0] instruction_in;

output [31:0] instruction_out;
reg [31:0] instruction_out;

always @(posedge clk_i or negedge rst_n) begin
    if(~rst_n) begin
        instruction_out <= 32'b0;
    end

    else
        begin
            instruction_out <= instruction_in;
        end

end
endmodule

```

剩下就是指派問題而已，將輸出接上輸入就完成了

ID 的內容有許多變化
Controller 要接上 OR 的 Opcode，Register 接上相對應的輸入輸出:
其中因為此時要寫入 REG 的是前面 ADDI 的值，因此 RegWrite 為 1，且 Write data 為 ADDI 的結果 10001(17)，Write register 為 ADDI 所要寫入的 Addr 00010($2)。

剩下與 LAB3 無異，只是將輸出接在 ID_EX_REG 上面而已。

ID_EX_REG
(一樣將輸入接上輸出，用 Clk_i 控制)

EX:
此時指令為 AND，
ALUSrc 為 0
DataSource 為 0
ALU control 為 010
ALUoperation 為 0000

EX_MEM_REG
(一樣將輸入接上輸出，用 Clk_i 控制)

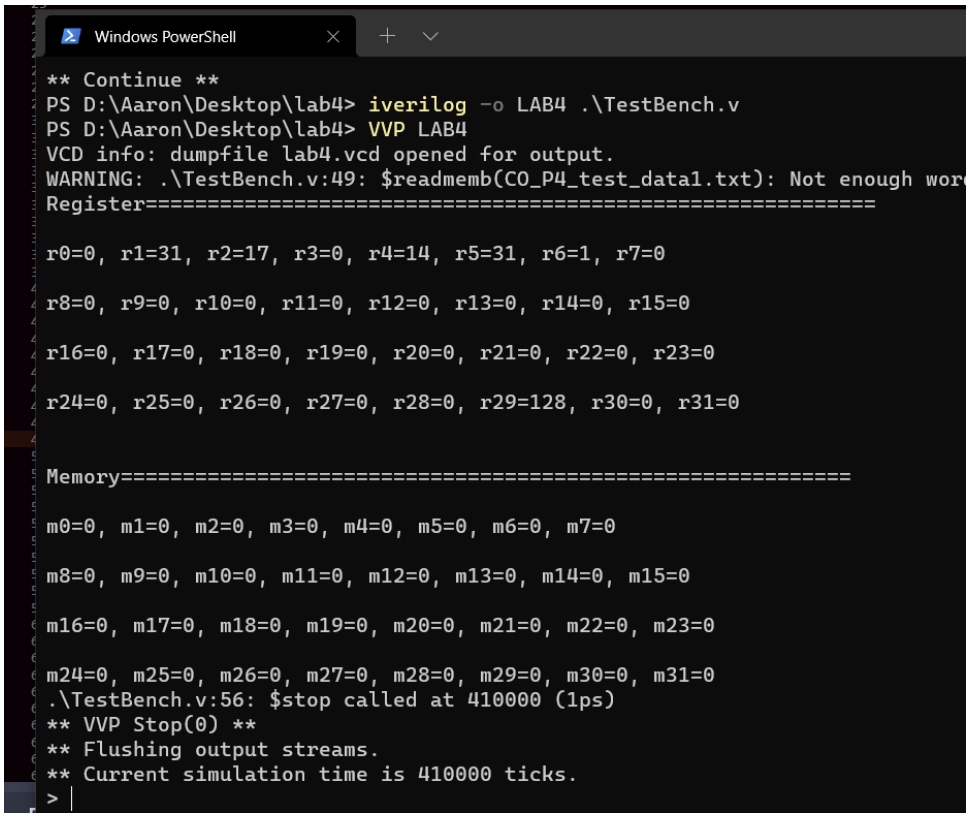
MEM
此時指令為 ADDI，
因為不須讀取記憶體，因此 MemRead、MemWrite 接為 don’t care

MEM_WB_REG
(一樣將輸入接上輸出，用 Clk_i 控制)

WB
RegWrite 要接回 REGISTER
RegtoMEM 為 0

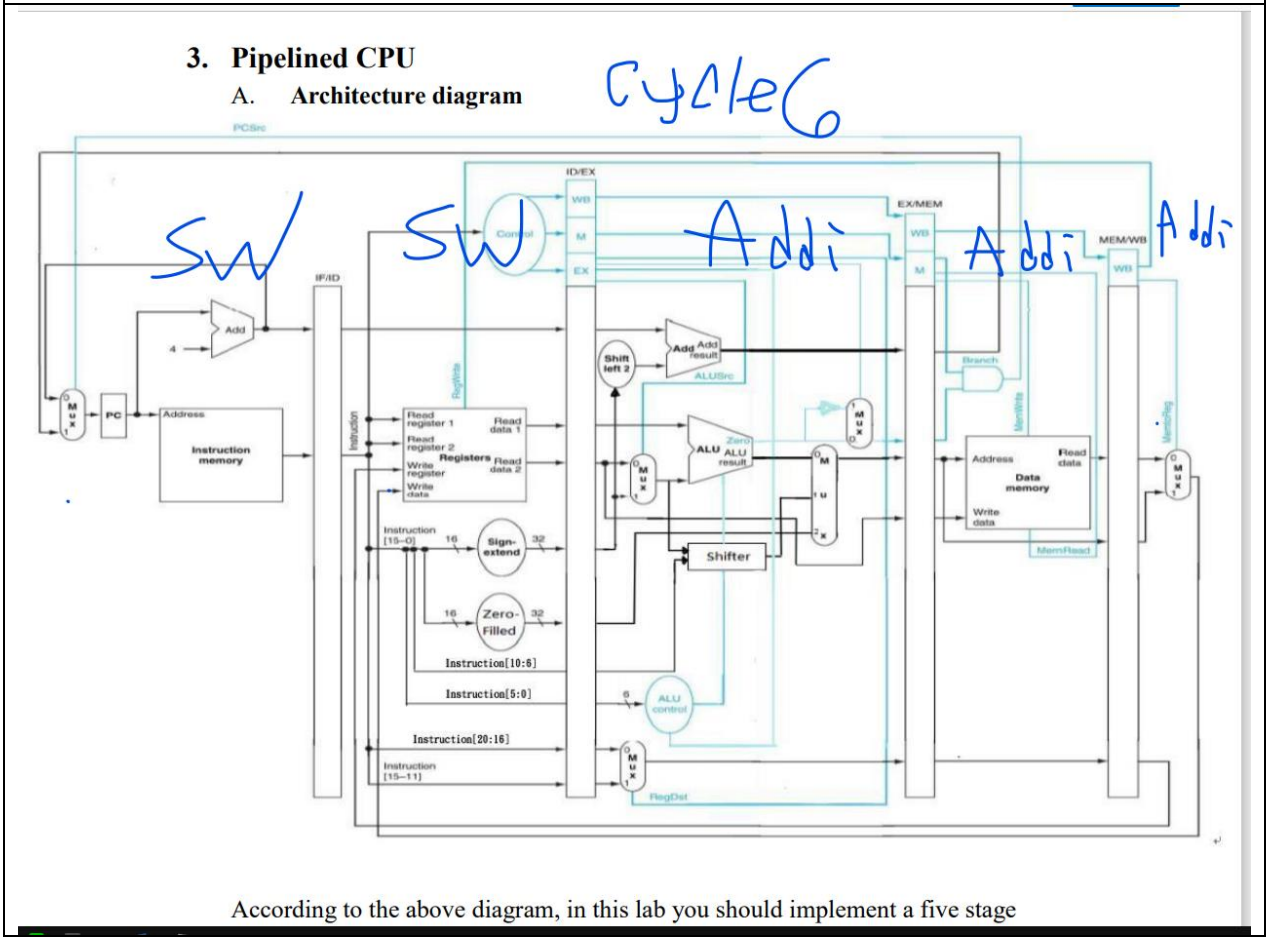
以上為第六個 cycle 的 Control signal 狀況

最後輸出結果



Picture:

CO_P5_test_data2



CO_P4_test_data2:

指令轉換:

ADDI $2, $Zero, 7
ADDI $3, $Zero, 8
ADDI $4, $Zero, 9
ADDI $5, $Zero, 10
SW $2, 4($Zero)
SW $3, 8($Zero)
SUB $5, $3, $2

Sixth cycle 的所有指令狀況如上

先看 IF 的部分，
此時進 instruction 為 SW，其餘與 data 一樣因此不多做解釋。

IF_ID_REG

作為 cycle 與 cycle 之間的間隔佔存，因此必須在內部設 clock 為 posedge。且 Testbench 內 rst_n 一開始為 0，所以將 rst_n 設為 negedge 作為初始化所有值為 0。

```
module IF_ID_REG(clk_i, rst_n, instruction_in, instruction_out);
input clk_i;
input rst_n;
input [31:0] instruction_in;

output [31:0] instruction_out;
reg [31:0] instruction_out;

always @(posedge clk_i or negedge rst_n) begin
    if(~rst_n) begin
        instruction_out <= 32'b0;
    end

    else
        begin
            instruction_out <= instruction_in;
        end

end
endmodule

```

剩下就是指派問題而已，將輸出接上輸入就完成了

ID
Controller 要接上 SW 的 Opcode，Register 接上相對應的輸入輸出:
其中因為此時要寫入 REG 的是前面 ADDI 的值，因此 RegWrite 為 1，且 Write data 為 ADDI 的結果 1000(8)，Write register 為 ADDI 所要寫入的 Addr 00011($3)。

剩下與 LAB3 無異，只是將輸出接在 ID_EX_REG 上面而已。

ID_EX_REG
(一樣將輸入接上輸出，用 Clk_i 控制)

EX:
此時指令為 ADDI，
ALUSrc 為 1
DataSource 為 0
ALU control 為 011
ALUoperation 為 0010

EX_MEM_REG
(一樣將輸入接上輸出，用 Clk_i 控制)

MEM
此時指令為 ADDI，
因為不須讀取記憶體，因此 MemRead、MemWrite 接為 don't care

MEM_WB_REG

(一樣將輸入接上輸出，用 Clk_i 控制)

WB
RegWrite 要接回 REGISTER
RegtoMEM 為 0

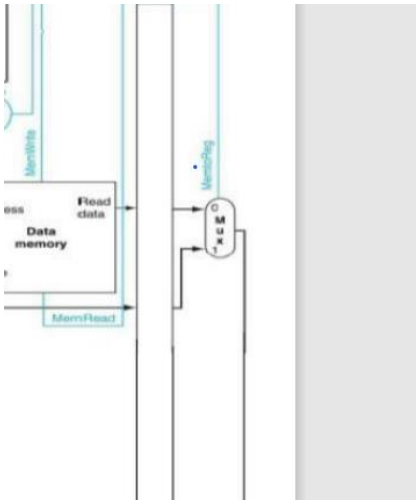
以上為第六個 cycle 的 Control signal 狀況

最後輸出結果

```
** Continue **
PS D:\Aaron\Desktop\lab4> iverilog -o LAB4 .\TestBench.v
PS D:\Aaron\Desktop\lab4> VVP LAB4
VCD info: dumpfile lab4.vcd opened for output.
WARNING: .\TestBench.v:49: $readmemb(CO_P4_test_data2.txt): Not enou
Register=============================================================

r0=0, r1=0, r2=7, r3=8, r4=9, r5=1, r6=0, r7=0

r8=0, r9=0, r10=0, r11=0, r12=0, r13=0, r14=0, r15=0

r16=0, r17=0, r18=0, r19=0, r20=0, r21=0, r22=0, r23=0

r24=0, r25=0, r26=0, r27=0, r28=0, r29=128, r30=0, r31=0


Memory===============================================================

m0=0, m1=7, m2=0, m3=0, m4=8, m5=0, m6=0, m7=0

m8=0, m9=0, m10=0, m11=0, m12=0, m13=0, m14=0, m15=0

m16=0, m17=0, m18=0, m19=0, m20=0, m21=0, m22=0, m23=0

m24=0, m25=0, m26=0, m27=0, m28=0, m29=0, m30=0, m31=0
.\TestBench.v:56: $stop called at 410000 (1ps)
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 410000 ticks.
> |
```

Problems you met and solutions:

像這種輸入輸出很多的大專案一定....一定要把所有變數名稱取好。我因為很懶所以變數名稱都取很像，導致最後在 Debub 很痛苦。
最後的解決辦法只能用 gtkwave 模擬器來抓錯，抓了好久才抓到只是一個地方打錯字(因為變數名稱太相像而接錯 signal)QQ
P.S.



題目給的圖片最後 MUX 與 Reference 有出入，我一開始覺得怪怪的所以有特別注意。不知道是否有人因為這個而出錯 De 不出 BUG。

Summary:

這次 Homework 看起來有些難度，其實很簡單(但我做了好久)。大部分時間都在 Debug 那個很白癡的小錯。Debug 佔我做的時間 80%以上吧。

這次只是加上四個 module 且把每個 module 輸入接上輸出就結束了，但 Homework4 卻是我做最久的一次作業。