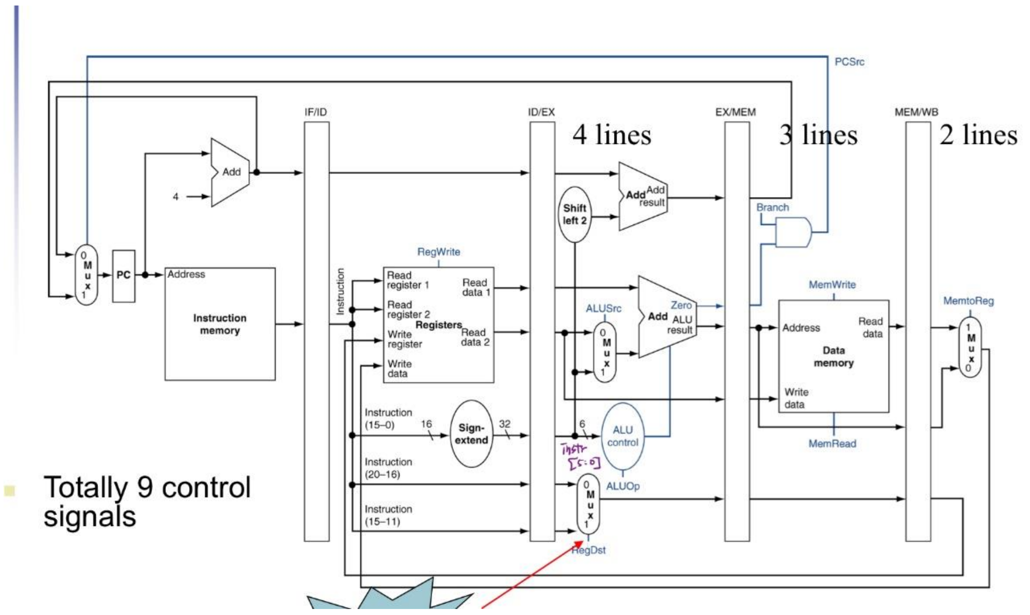
Computer Organization Lab4

Name:梁詠晴

ID:109550134

Architecture diagrams:



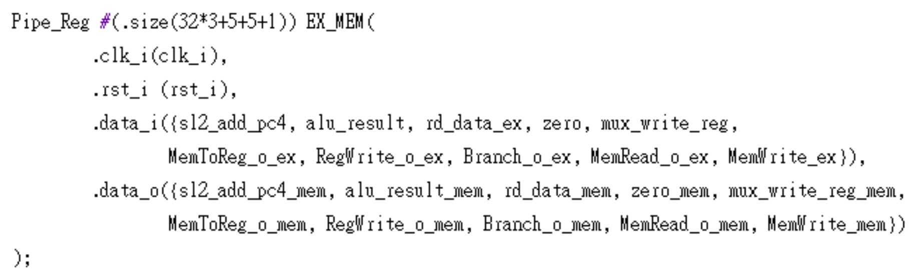
Hardware module analysis:

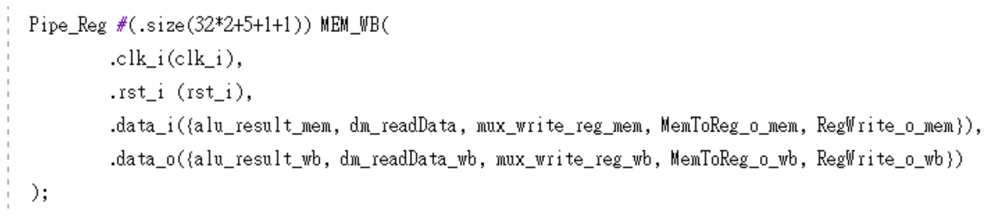
比起 single cycle cpu，在 stage 之間新增 reg 以將會用到的資料傳到下個

stage : reg if/id、reg id/ex、reg ex/mem、reg mem/wb

```
Pipe_Reg #(.size(32*2)) IF_ID(        //N is the total length of input/output
        .clk_i(clk_i),
        .rst_i (rst_i),
        .data_i({instr_o, pc_add_four}),
        .data_o({instr_o_id, pc_add_four_id})
);
```

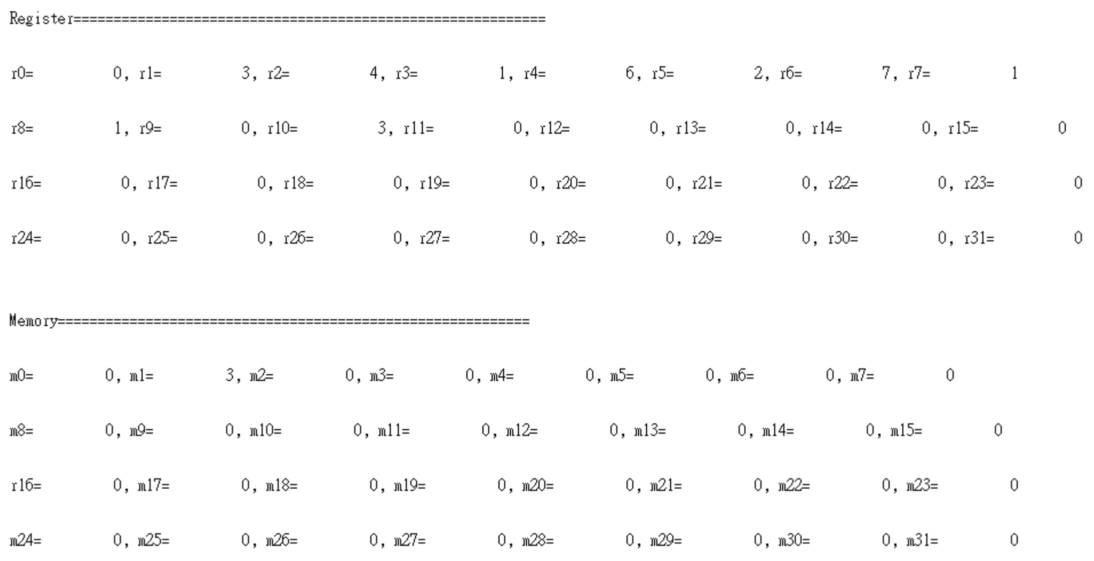
```
Pipe_Reg #(.size(32*4+3+2+5+21)) ID_EX(
        .clk_i(clk_i),
        .rst_i (rst_i),
        .data_i({rs_data, rd_data, se_o, pc_add_four_id,ALU_op_o,RegDst_o, MemToReg_o,
               RegWrite_o, ALUSrc_o, Branch_o, MemRead_o, MemWrite,instr_o_id[20:0]}),
        .data_o({rs_data_ex, rd_data_ex, se_o_ex, pc_add_four_ex,ALU_op_o_ex,RegDst_o_ex, MemToReg_o_ex,
               RegWrite_o_ex, ALUSrc_o_ex, Branch_o_ex, MemRead_o_ex, MemWrite_ex,instr_ex})
);
```

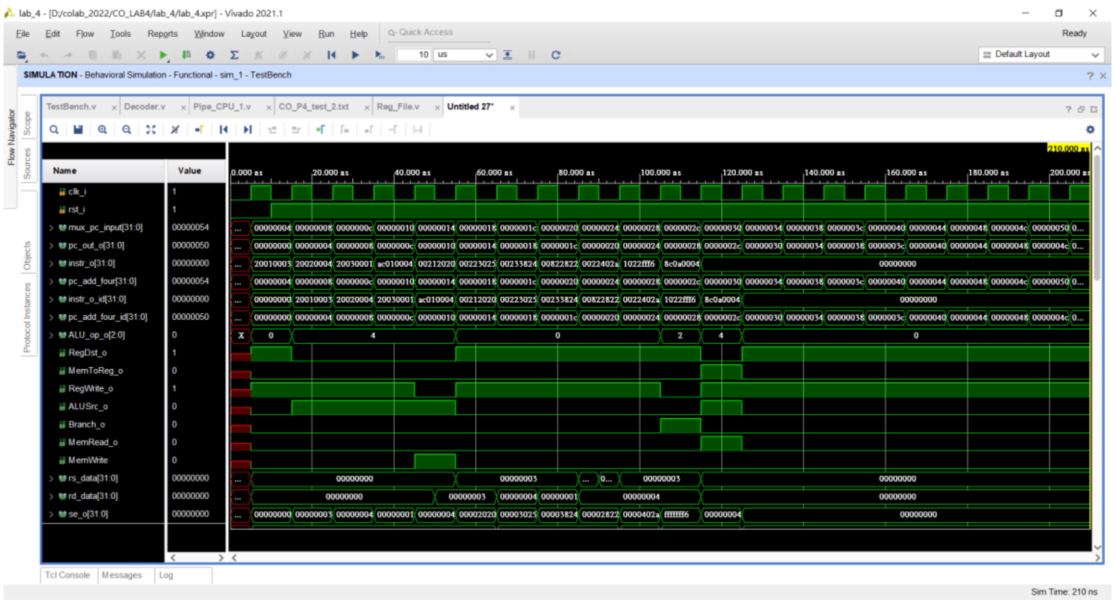

```
Pipe_Reg #(.size(32*3+5+5+1)) EX_MEM(
        .clk_i(clk_i),
        .rst_i (rst_i),
        .data_i({sl2_add_pc4, alu_result, rd_data_ex, zero, mux_write_reg,
               MemToReg_o_ex, RegWrite_o_ex, Branch_o_ex, MemRead_o_ex, MemWrite_ex}),
        .data_o({sl2_add_pc4_mem, alu_result_mem, rd_data_mem, zero_mem, mux_write_reg_mem,
               MemToReg_o_mem, RegWrite_o_mem, Branch_o_mem, MemRead_o_mem, MemWrite_mem})
);
```


```
Pipe_Reg #(.size(32*2+5+1+1)) MEM_WB(
        .clk_i(clk_i),
        .rst_i (rst_i),
        .data_i({alu_result_mem, dm_readData, mux_write_reg_mem, MemToReg_o_mem, RegWrite_o_mem}),
        .data_o({alu_result_wb, dm_readData_wb, mux_write_reg_wb, MemToReg_o_wb, RegWrite_o_wb})
);
```

Finished part:

執行結果和預期相符:


```
Register===========================================================
r0=          0, r1=          3, r2=          4, r3=          1, r4=          6, r5=          2, r6=          7, r7=          1
r8=          1, r9=          0, r10=          3, 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=          0, r30=          0, r31=          0

Memory===========================================================
m0=          0, m1=          3, m2=          0, m3=          0, m4=          0, m5=          0, m6=          0, m7=          0
m8=          0, m9=          0, m10=          0, m11=          0, m12=          0, m13=          0, m14=          0, m15=          0
r16=          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
```

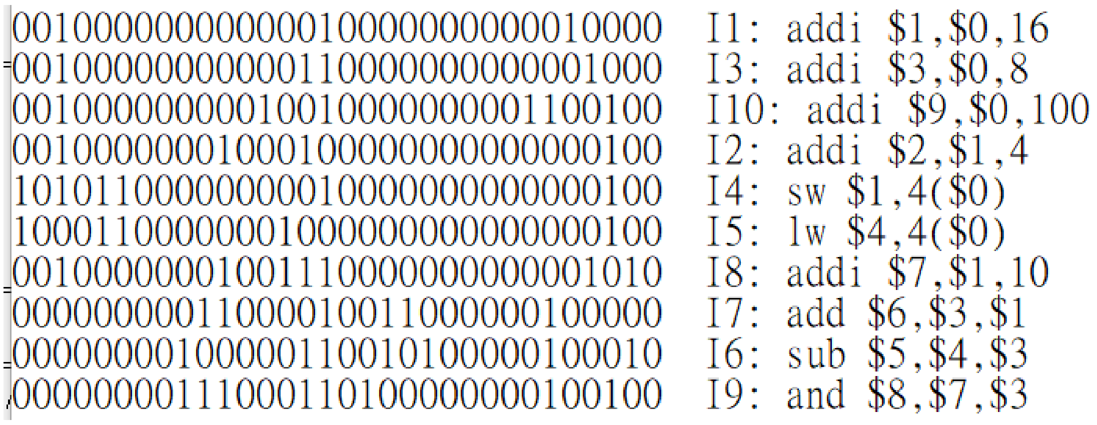


Problems you met and solutions:

在測試時發現有被使用、應該要跑出數值的 reg 顯示時都變成顯示 x。後來發現是因為在計算 ID/EX reg 時將 reg 的大小算錯了，使 reg 漏存部分資料導致。

Bonus (optional):

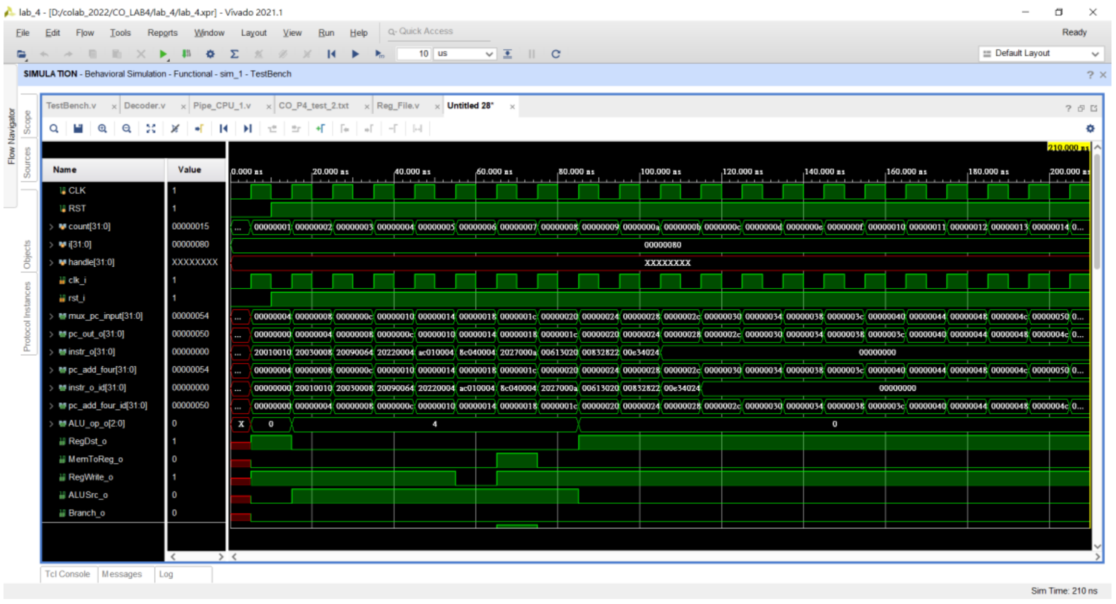
將 I3 和 I10 提前到 I2 前執行，I8 和 I6 交換順序，便可以解決 data hazard 的問題。下圖為調整過後的順序:



```
00100000000000010000000000010000   I1:  addi $1,$0,16
00100000000000110000000000001000   I3:  addi $3,$0,8
00100000000010010000000001100100   I10: addi $9,$0,100
00100000001000100000000000000100   I2:  addi $2,$1,4
10101100000000010000000000000100   I4:  sw $1,4($0)
10001100000001000000000000000100   I5:  lw $4,4($0)
00100000001001110000000000001010   I8:  addi $7,$1,10
00000000011000010011000000100000   I7:  add $6,$3,$1
00000000100000110010100000100010   I6:  sub $5,$4,$3
00000000111000110100000000100100   I9:  and $8,$7,$3
```

執行結果:

```
Register=============================================================

r0=          0, r1=         16, r2=         20, r3=          8, r4=         16, r5=          8, r6=         24, r7=         26

r8=          8, r9=        100, 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=         0, r30=         0, r31=          0


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

m0=          0, m1=         16, m2=          0, m3=          0, m4=          0, m5=          0, m6=          0, m7=          0

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

r16=         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
```


lab_4 - [D:/colab_2022/CO_LAB4/lab_4/lab_4.xpr] - Vivado 2021.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - TestBench

Name	Value
CLK	1
RST	1
count[31:0]	00000015
i[31:0]	00000080
handle[31:0]	XXXXXXXX
clk_i	1
rst_i	1
mux_pc_input[31:0]	00000054
pc_out_o[31:0]	00000050
instr_o[31:0]	00000000
pc_add_four[31:0]	00000054
instr_o_id[31:0]	00000000
pc_add_four_id[31:0]	00000050
ALU_op_o[2:0]	0
RegDst_o	1
MemToReg_o	0
RegWrite_o	1
ALUSrc_o	0
Branch_o	0

Sim Time: 210 ns

Summary:

這次 lab 讓我深刻體驗到細心的重要，為了找出 bug，我重複檢查了好幾次 cpu module 中的接線，卻一直假設 reg 大小沒有錯而略過檢查。若是能更仔細檢查，便能大幅減少找出問題的時間。