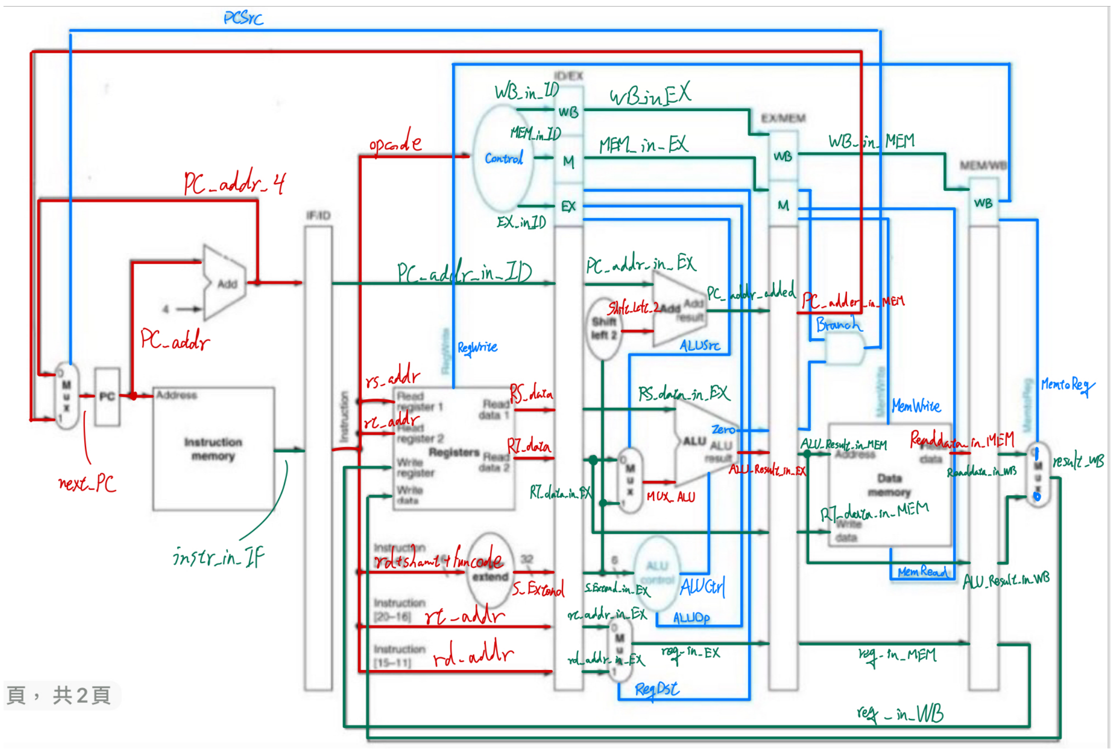
Computer Organization

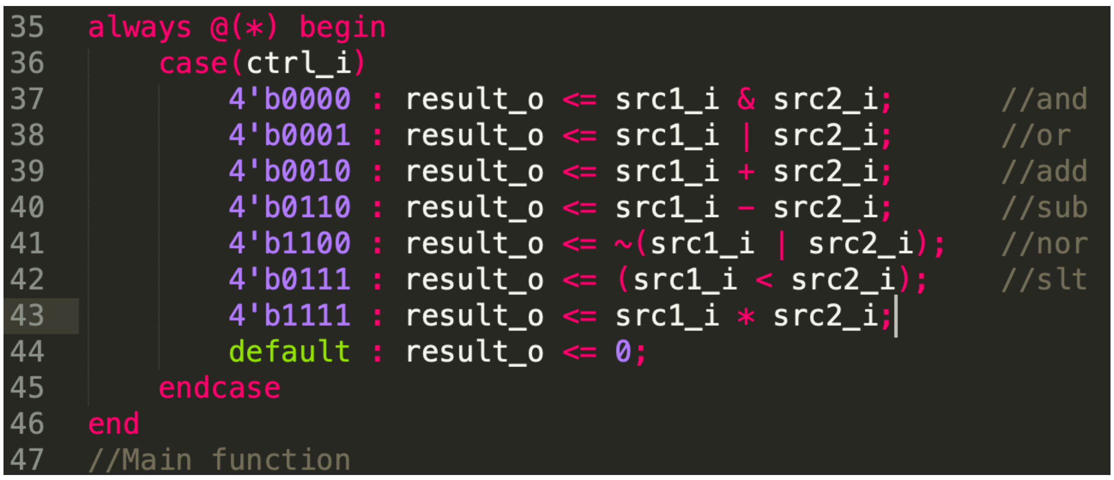
Architecture diagrams:



上圖是此次作業的架構，主要是參照上圖做接線，藍線為control signal，綠線為儲存在各stage的電路。

Hardware module analysis:

首先是mult指令的部分，原本把這東西想得超複雜，需要HI/LO來做，好在有同學問了問題才發現原來這麼輕鬆，至於ALUCtrl不知道要用多少就寫1111了。



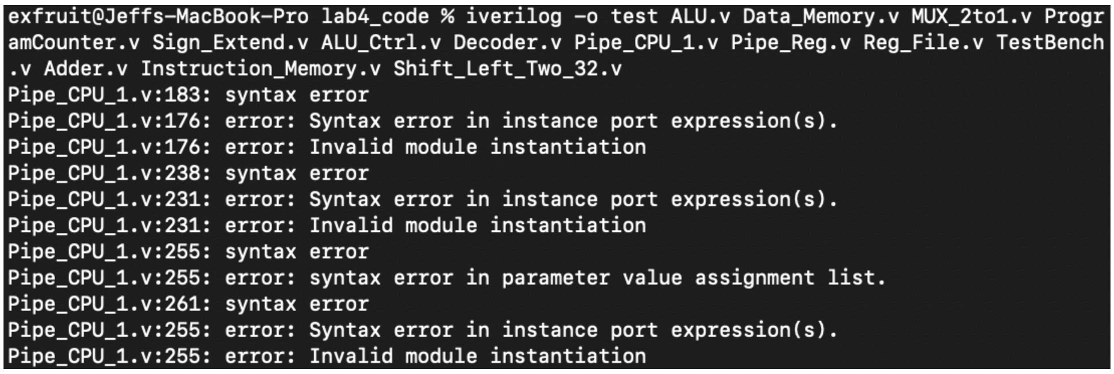
```
always @(*) begin
    case(ctrl_i)
        4'b0000 : result_o <= src1_i & src2_i;      //and
        4'b0001 : result_o <= src1_i | src2_i;      //or
        4'b0010 : result_o <= src1_i + src2_i;      //add
        4'b0110 : result_o <= src1_i - src2_i;      //sub
        4'b1100 : result_o <= ~(src1_i | src2_i);   //nor
        4'b0111 : result_o <= (src1_i < src2_i);    //slt
        4'b1111 : result_o <= src1_i * src2_i;
        default : result_o <= 0;
    endcase
end
//Main function
```

其他主要沒什麼需要特別描述的，所以這part就這樣。

Problem I met and solutions:

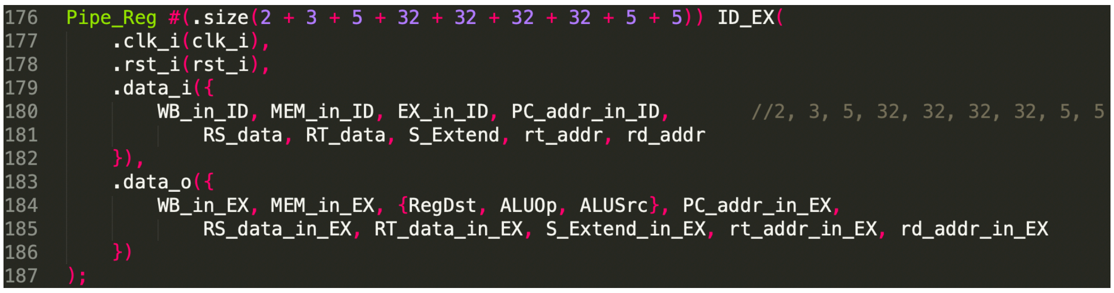
在處理control signal在各個stage的接線時一開始把bus的接線反了過來，在接ALUCtrl時發現這樣的輸入好像會整個反過來，看了前次功課才確認並改正。

在全部接完線後出現以下訊息：



```
exfruit@Jeffs-MacBook-Pro lab4_code % iverilog -o test ALU.v Data_Memory.v MUX_2to1.v Progr
amCounter.v Sign_Extend.v ALU_Ctrl.v Decoder.v Pipe_CPU_1.v Pipe_Reg.v Reg_File.v TestBench
.v Adder.v Instruction_Memory.v Shift_Left_Two_32.v
Pipe_CPU_1.v:183: syntax error
Pipe_CPU_1.v:176: error: Syntax error in instance port expression(s).
Pipe_CPU_1.v:176: error: Invalid module instantiation
Pipe_CPU_1.v:238: syntax error
Pipe_CPU_1.v:231: error: Syntax error in instance port expression(s).
Pipe_CPU_1.v:231: error: Invalid module instantiation
Pipe_CPU_1.v:255: syntax error
Pipe_CPU_1.v:255: error: syntax error in parameter value assignment list.
Pipe_CPU_1.v:261: syntax error
Pipe_CPU_1.v:255: error: Syntax error in instance port expression(s).
Pipe_CPU_1.v:255: error: Invalid module instantiation
```

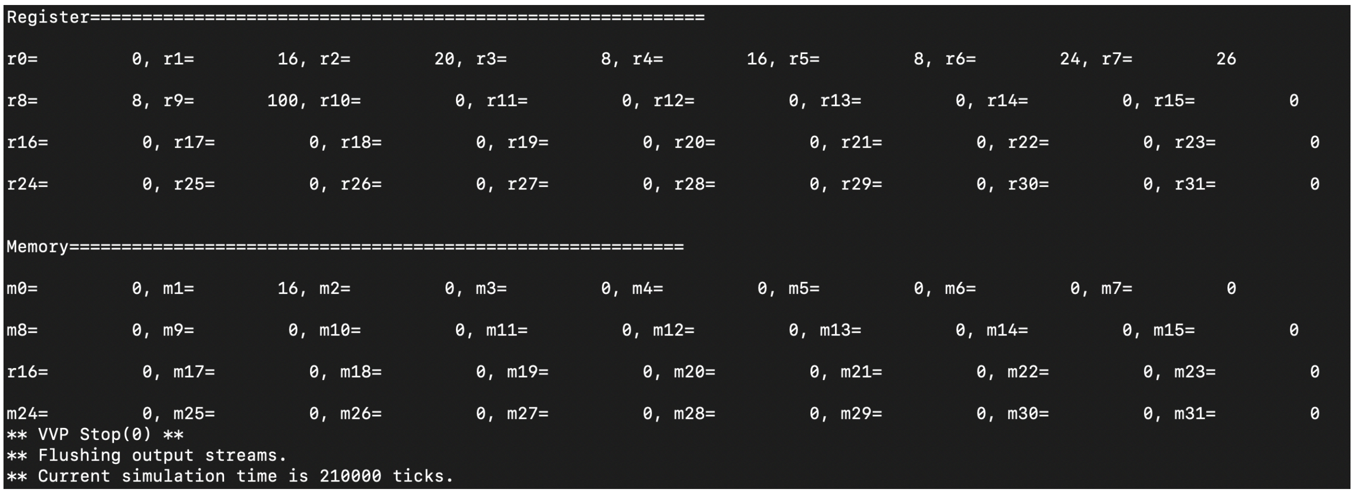
然後這是程式碼：



```
176  Pipe_Reg #(.size(2 + 3 + 5 + 32 + 32 + 32 + 32 + 5 + 5)) ID_EX(
177      .clk_i(clk_i),
178      .rst_i(rst_i),
179      .data_i({
180          WB_in_ID, MEM_in_ID, EX_in_ID, PC_addr_in_ID,       //2, 3, 5, 32, 32, 32, 32, 5, 5
181              RS_data, RT_data, S_Extend, rt_addr, rd_addr
182      }),
183      .data_o({
184          WB_in_EX, MEM_in_EX, {RegDst, ALUOp, ALUSrc}, PC_addr_in_EX,
185              RS_data_in_EX, RT_data_in_EX, S_Extend_in_EX, rt_addr_in_EX, rd_addr_in_EX
186      })
187  );
```

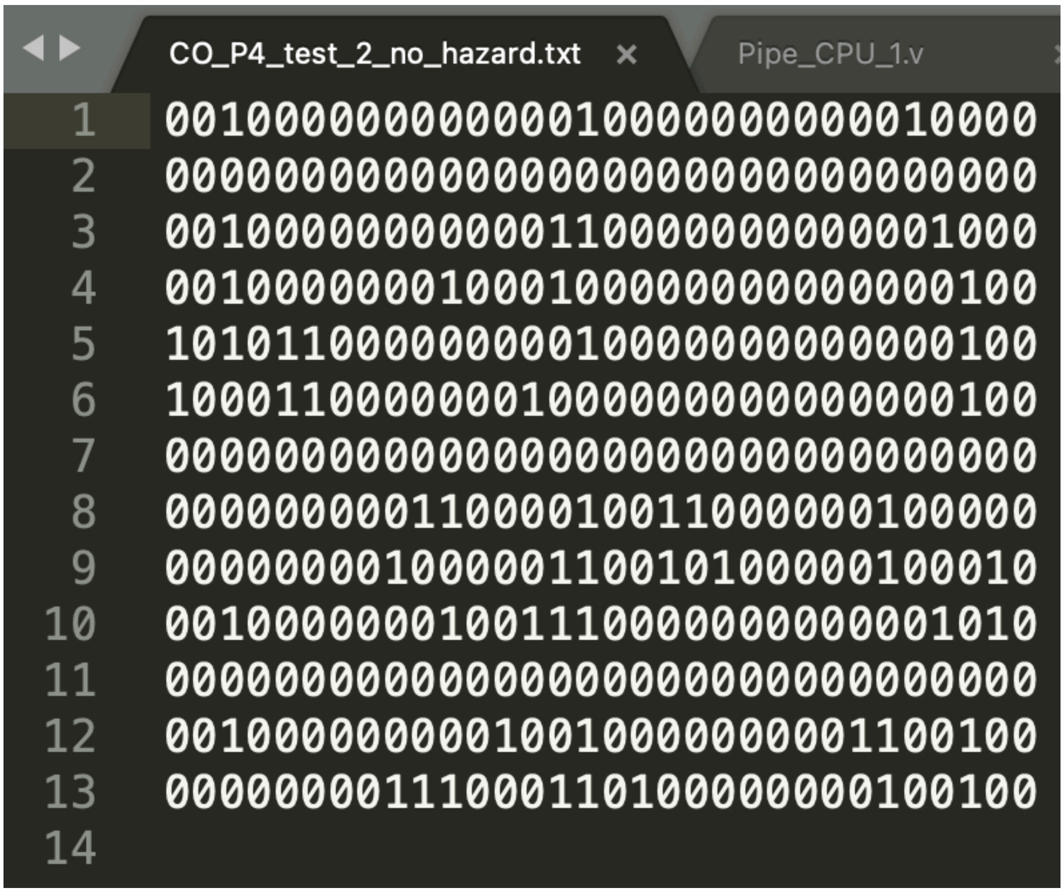
我想不透到底哪裡錯，結果去看其他人的code在.data_i({....這裡，是寫.data_i({...，沒錯，就是有個空格在(和{中間，我就不信邪的加了空格，然後就對了？？？？？？？？我發誓我沒唬爛，重點是，我把它改回來，結果他還是能跑...，我就在這裡浪費了半小時，也不知道這算不算解決方法。

Result:



```
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
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 210000 ticks.
```

以上是測資2的結果，處理data hazard的方法是，因為無法forwarding，所以在兩個指令內有data dependence都會產生data hazard，所以我將I2,I3調換，並在I1下一行加入Bubble；I6,I7調換，在I5下一行加入Bubble；將I9,I10調換，在I8下一行加入Bubble。以下是修改結果：



```
00100000000000010000000000010000
00000000000000000000000000000000
00100000000000110000000000001000
00100000001000100000000000000100
10101100000000010000000000000100
10001100000001000000000000000100
00000000000000000000000000000000
00000000011000010011000000100000
00000000100000110010100000100010
00100000001001110000000000001010
00000000000000000000000000000000
00100000000010010000000001100100
00000000111000110100000000100100

```

Summary:

　比起上次，這次功課顯得友善許多，雖然還是因為一些奇奇怪怪而且很莫名其妙的問題花了點時間，最後都還是成功解決了，在bouns problem中也更了解了pipeline的運作。