111-2 計算機組織 Final Project: Pipelined CPU Design 報告

111 學年度第 2 學期

教授：朱守禮 教授

學生/組員：

資訊二甲 10727120 洪錦彤

電資二 11020107 蘇伯勳

電資二 11020137 關翔謙

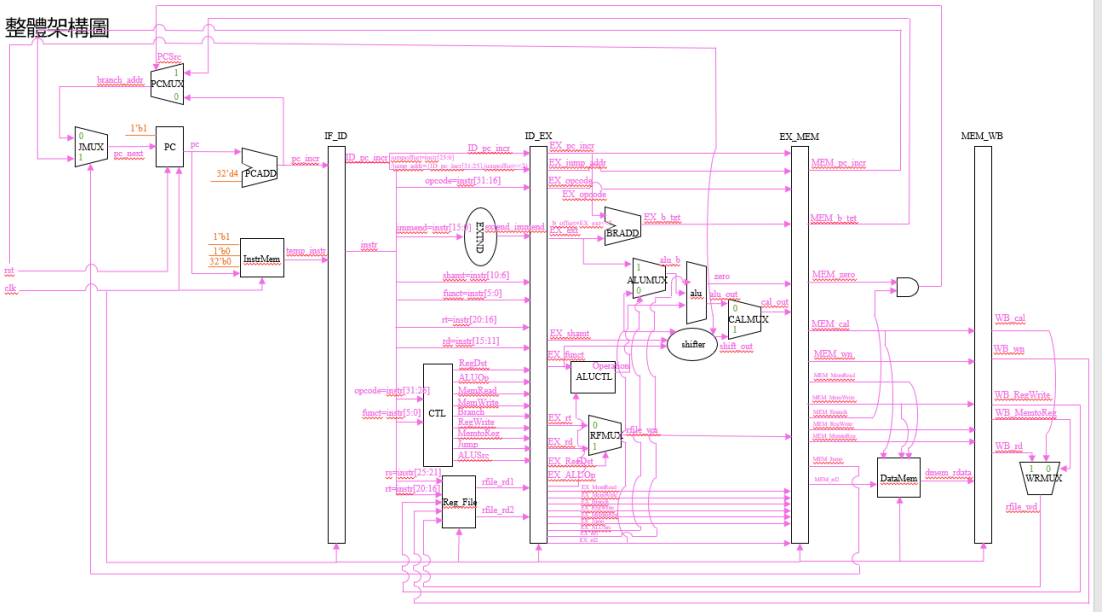
電資二 11020140 葉柏榆

一、背景

此 Final project 為 Midterm project 的拓展，將 ALU 擴增成 Pipelined MIPS-Lite CPU。要求是"add、sub、and、or、srl、slt、addiu、lw、sw、beq、j、multu、maddu、mfhi、mflo、nop"這 16 道指令，最終 multu、maddu 沒做，連帶地也用不到 nfhi、mflo，所以只有實現 12 道指令。ALU 及 Shifter 都請參照期中報告的設計，五個階段若有過多的接線宣告也顯得冗餘，就只在總架構說明一次，在 module 中不再重複。工作分配：洪錦彤繪製最終 datapath，蘇伯勳畫 datapath 初稿與撰寫報告，關翔謙跑 ModelSim 測試、debug、產測資、觀測波型，葉柏榆實作。

二、方法

(1) 總架構：請參照 mips_pipeline.v。



參照 datapath，input 為 clk 與 rst，接著宣告一大堆接線，並開始按照流程執行：

1. PC：
   Input: clk, rst, 1'b1, pc_next
   Output: pc
2. PCMUX：
   Input: PCSrc, pc_incr, MEM_b_tgt
   Output: branch_addr
3. JMUX：
   Input: MEM_Jump, branch_addr, MEM_jump_addr
   Output: pc_next
4. PCADD：
   Input: pc, 32' 4
   Output: pc_incr
5. InstrMem：
   Input: clk, 1'b1, 1'b0, 32'b0, pc
   Output: temp_instr
6. IF_ID：
   Input: clk, pc_incr, temp_instr
   Output: ID_pc_incr, instr
7. CTL：
   Input: opcode, funct
   Output: RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump, ALUOp
8. Reg_File：
   Input: clk, WB_RegWrite, rs, rt, WB_wn, rfile_wd
   Output: rfile_rd1, rfile_rd2
9. SignExt：
   Input: immed
   Output: extend_immed
10. ID_EX：
    Input: clk, ID_pc_incr, rfile_rd1, rfile_rd2, extend_immed, jump_addr, shamt, rt, rd, ALUOp, RegDst, ALUSrc, MemRead, MemWrite, Branch, RegWrite, MemtoReg, Jump, funct, opcode
    Output: EX_pc_incr, EX_rd1, EX_rd2, EX_ext, EX_jump_addr, EX_shamt, EX_rt, EX_rd, EX_RegDst, EX_ALUSrc, EX_MemRead, EX_MemWrite, EX_Branch, EX_RegWrite, EX_MemtoReg, EX_Jump, EX_ALUOp, EX_funct, EX_opcode

11. BRADD：
    Input: EX_pc_incr, b_offset
    Output: EX_b_tgt
12. ALUMUX：
    Input: EX_ALUSrc, EX_rd2, EX_ext
    Output: alu_b
13. RFMUX：
    Input: EX_RegDst, EX_rt, EX_rd
    Output: rfile_wn
14. ALUCTL：
    Input: EX_ALUOp, EX_funct
    Output: Operation
15. alu：
    Input: EX_rd1, alu_b, Operation
    Output: alu_out, Zero
16. shifter：
    Input: EX_rd2, EX_shamt, EX_funct, rst
    Output: shift_out
17. CALMUX：
    Input: EX_funct, alu_out, shift_out
    Output: cal_out
18. EX_MEM：
    Input: clk, EX_pc_incr, EX_b_tgt, cal_out, EX_rd2, rfile_wn, Zero, EX_MemRead, EX_MemWrite, EX_Branch, EX_RegWrite, EX_MemtoReg, EX_Jump, EX_jump_addr, EX_opcode
    Output: MEM_pc_incr, MEM_b_tgt, MEM_cal, MEM_rd2, MEM_wn, MEM_zero, MEM_MemRead, MEM_MemWrite, MEM_Branch, MEM_RegWrite, MEM_MemtoReg, MEM_Jump, MEM_jump_addr, MEM_opcode
19. and：
    Input: MEM_Branch, MEM_zero
    Output: PCSrc
20. DataMem：
    Input: clk, MEM_MemRead, MEM_MemWrite, MEM_rd2, MEM_cal
    Output: dmem_rdata
21. MEM_WB：
    Input: clk, MEM_cal, dmem_rdata, MEM_wn, MEM_RegWrite, MEM_MemtoReg

Output: WB_cal, WB_rd, WB_wn, WB_RegWrite, WB_MemtoReg

22. WRMUX：

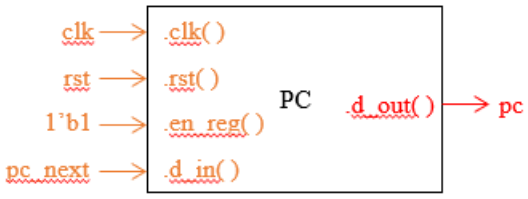
Input: WB_MemtoReg, WB_cal, WB_rd

Output: rfile_wd

接著開始講解設計。

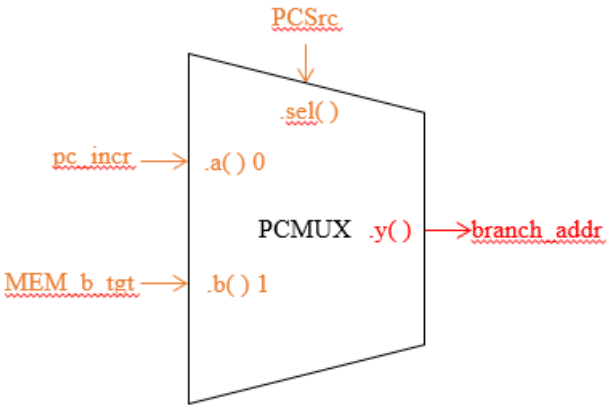
(2) PC：請參照 reg32.v。

en_reg 直接塞 1，當 clk 敲起，若 reset 則 0，不然就寫入（所以一直 enable，塞 1 代表）enable。



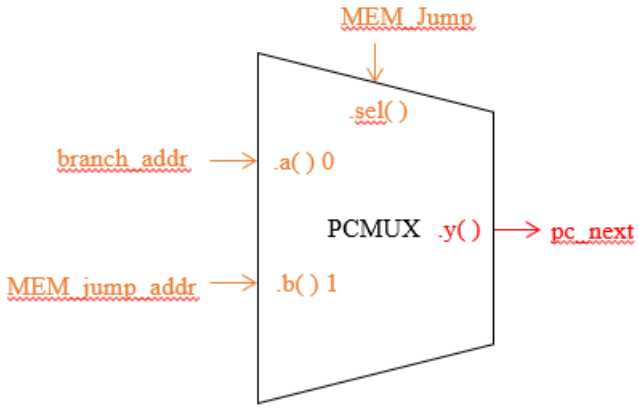
(3) PCMUX：請參照 mux2.v。

二對一多工器，選擇訊號為 PCSrc，輸出為 branch_addr。當訊號為 x 時輸出 pc_incr，else 訊號為 1 時輸出 MEM_b_tgt，else 輸出 pc_incr。



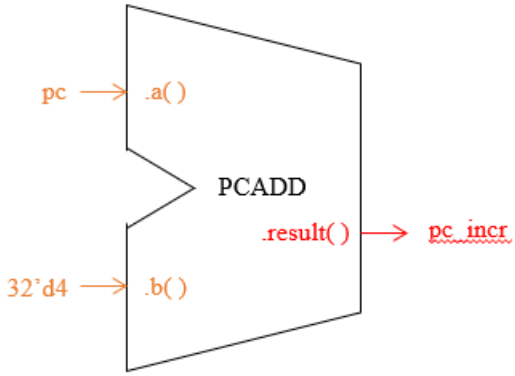
(4) JMUX：請參照 mux2.v。

二對一多工器，選擇訊號為 MEM_Jump，輸出為 pc_next。當訊號為 x 時輸出 branch_addr，else 訊號為 1 時輸出 MEM_jump_addr，else 輸出 branch_addr。



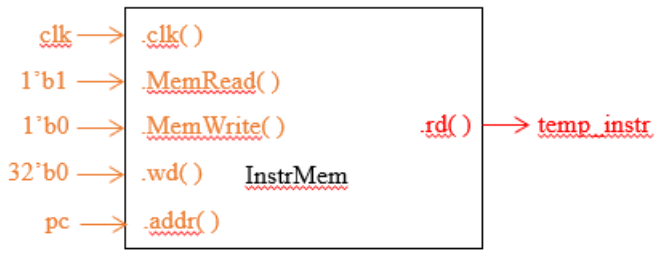
(5) PCADD：請參照 add32.v。

用於實現 PC+4，輸入就是 pc 和 4，輸出 pc+4 到 pc_incr。



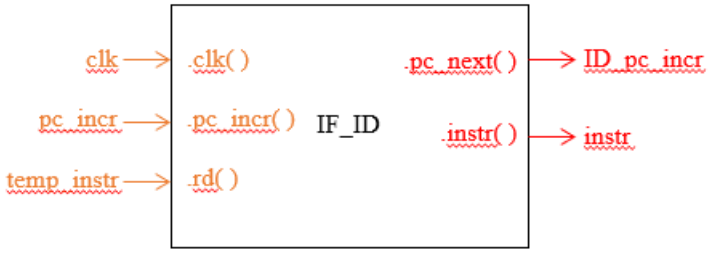
(6) InstrMem：請參照 memory.v。

輸入為 clk，永遠可讀永不可寫所以直接塞 1 和 0，並且用不到 wd 所以塞 0；addr 用 pc，輸出的 rd 為 temp_instr。當 read enable 或 mem_array 的 addr~addr+4 有東西時，若 read enable 就依照指定位址抓指令，不然輸出 x；由於不可寫所以不用管 MemWrite == 1。



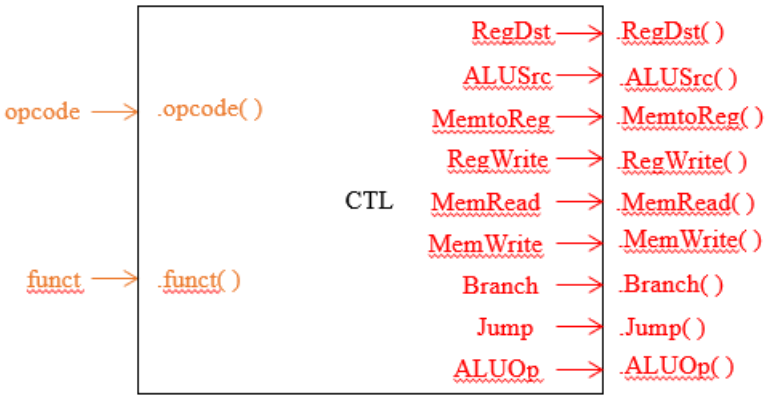
(7) IF_ID：請參照 IF_ID.v。

當 clk 敲起時把 pc+4 和指令丟出去。



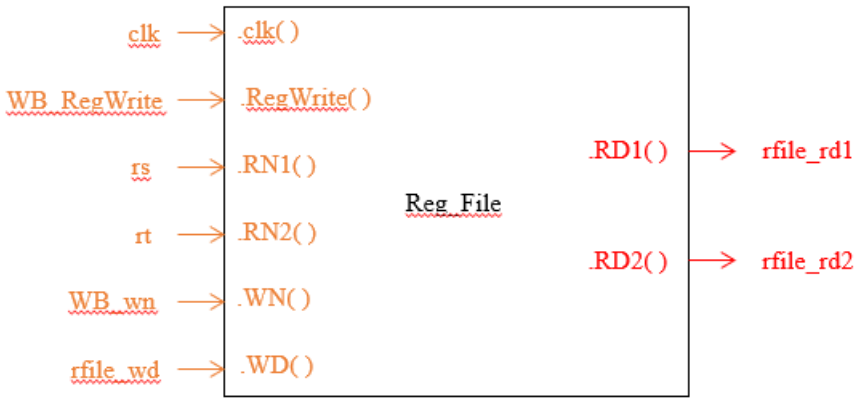
(8) CTL：請參照 control_pipeline.v。

輸入為從 instr[31:26]抓的 opcode 和 instr[5:0]的 function，並且當其中一個有東西時進入 switch-case 分成以下 cases：R_FORMAT(if function!=0, else NOP)、I_ADDIU、LW、SW、BEQ、J，default 沒實作的的指令，其中的輸出值就照 code 走不贅述。



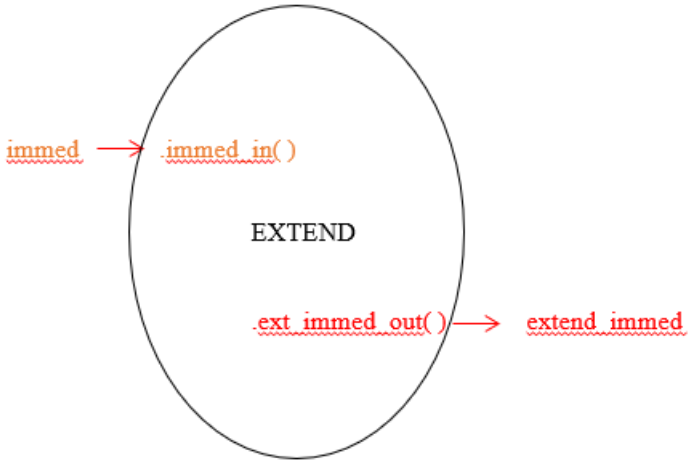
(9) Reg_File：請參照 reg_file.v。

當 clk 敲起時若 write enable 且 WN 不為 0 則將 WD 寫入 file_array[WN]；並在 RN1、RN@或 file_array[RN1]、file_array[RN2]觸發時若 RN1、RN2==0 則其對應的 RD 寫入 0，else 對應的 RD 寫入 file_array[對映的 RN]。



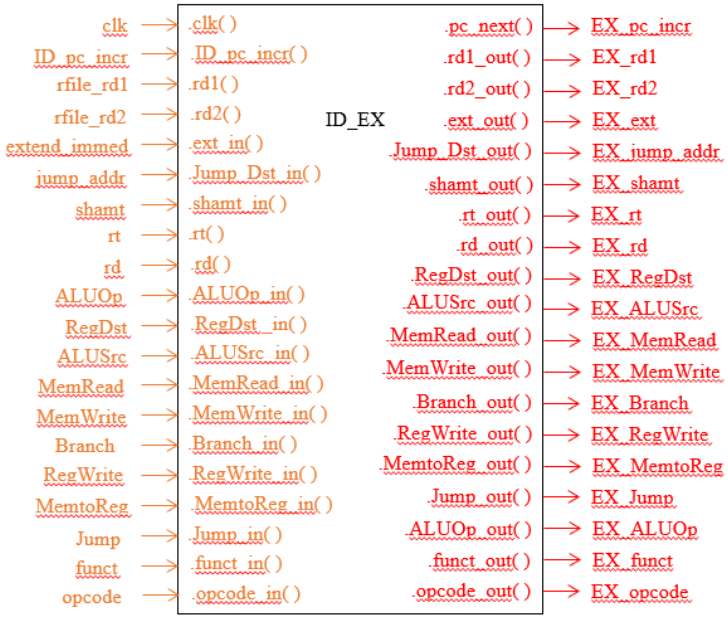
(10) SignExt：請參照 sign_extend.v。

將輸入 immed 的最高位（第 15 位元）複製 16 次銜接原本的 immed 以拓展成 32 位元輸出到 extend_immed。



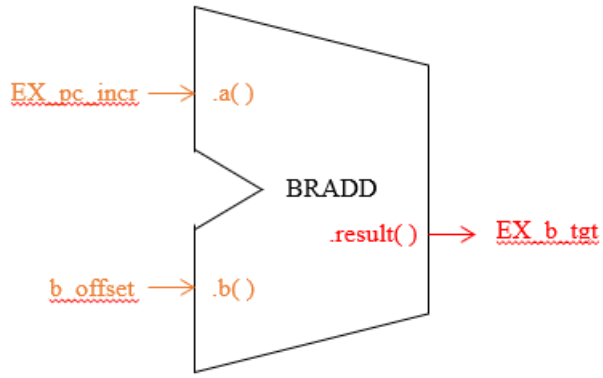
(11) ID_EX：請參照 ID_EX.v。

當 clk 敲起將輸入丟出去，詳見上面的總架構或直接看 code。



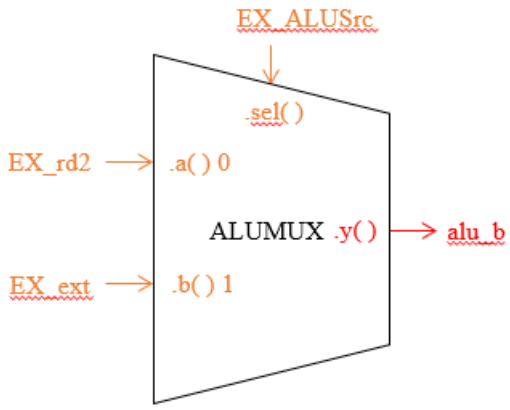
(12) BRADD：請參照 add32.v。

將輸入的 EX_pc_incr（即 PC+4）與由 SignExt 拓展的 32 位元 extend_immed 左移兩位元（即*4）所得之 b_offset 相加並輸出給 EX_b_tgt。



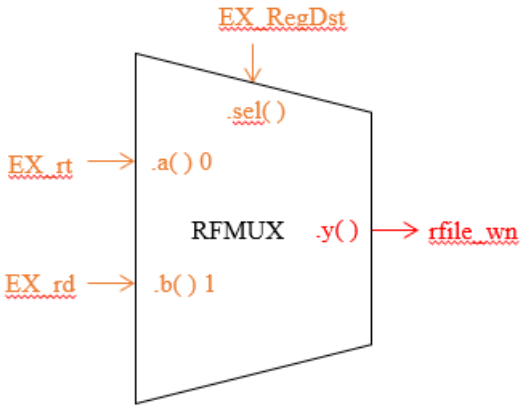
(13) ALUMUX：請參照 mux2.v。

二對一多工器，選擇訊號為 EX_ALUSrc，輸出為 alu_b。當訊號為 x 時輸出 EX_rd2，else 訊號為 1 時輸出 EX_ext，else 輸出 EX_rd2。



(14) RFMUX：請參照 mux2_5bit.v。

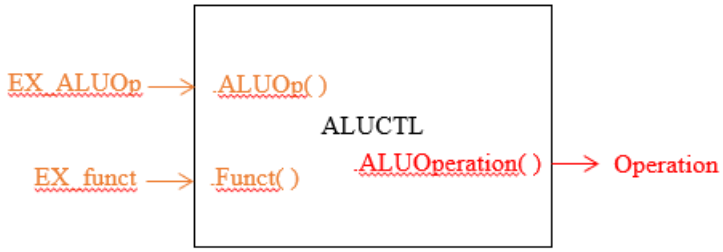
二對一多工器，選擇訊號為 EX_RegDst，輸出為 rfile_wn。當訊號為 1 時輸出 EX_rd，else 輸出 EX_rt。



(15) ALUCTL：請參照 alu_ctl.v。

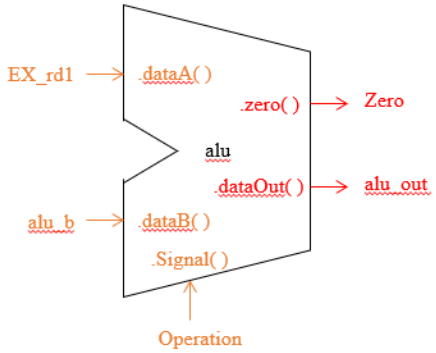
輸入 EX_ALUOp 與 EX_funct 觸發，以 EX_ALUOp 選擇 switch-case：00=>add, 01=>sub, 10=>以 EX_funct 選擇 switch-case：

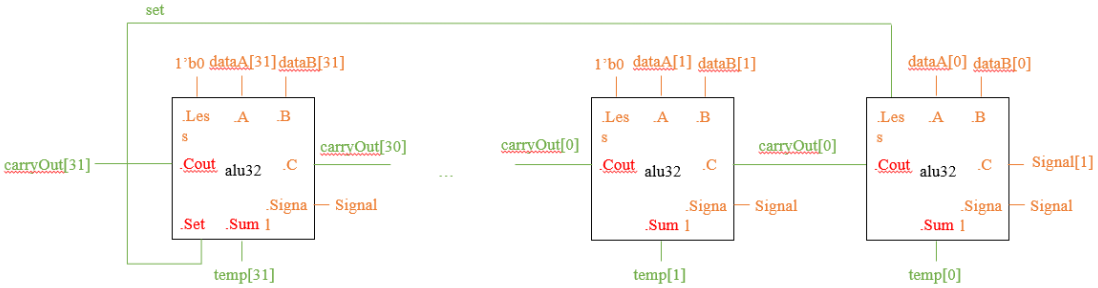
mfhi、mflo、add 為 add，sub 為 sub，and 為 and，or 為 or，slt 為 slt；兩個 switch-case 的 default 都為 xxx。輸出給 Operation。

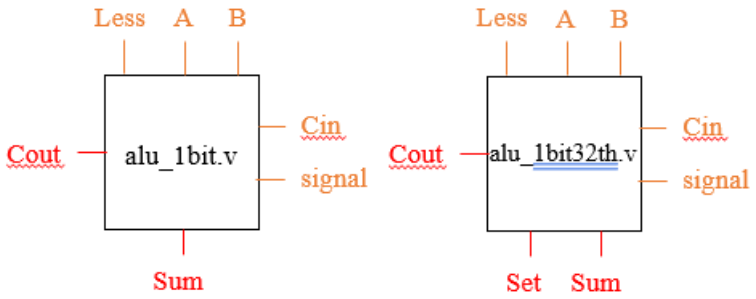


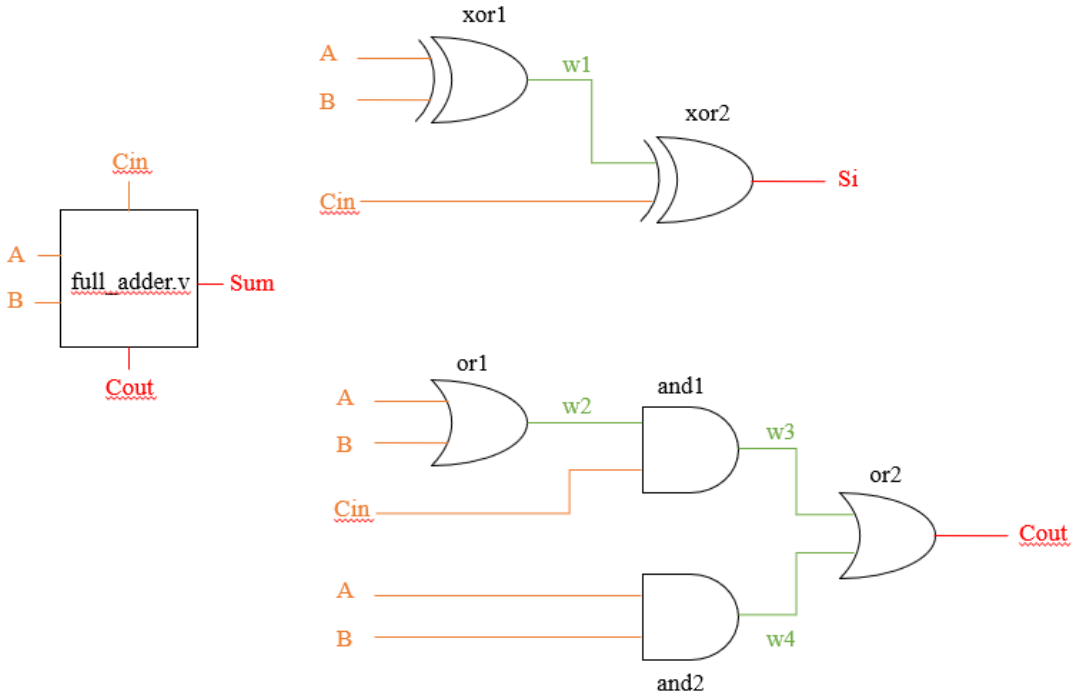
(16) alu：請參考 ALU.v、full_adder.v、alu_1bit.v、alu_1bit32th.v、期中報告的這些 module 說明。基本和期中的雷同，都是 1bit full adder => 1bit ALU =>

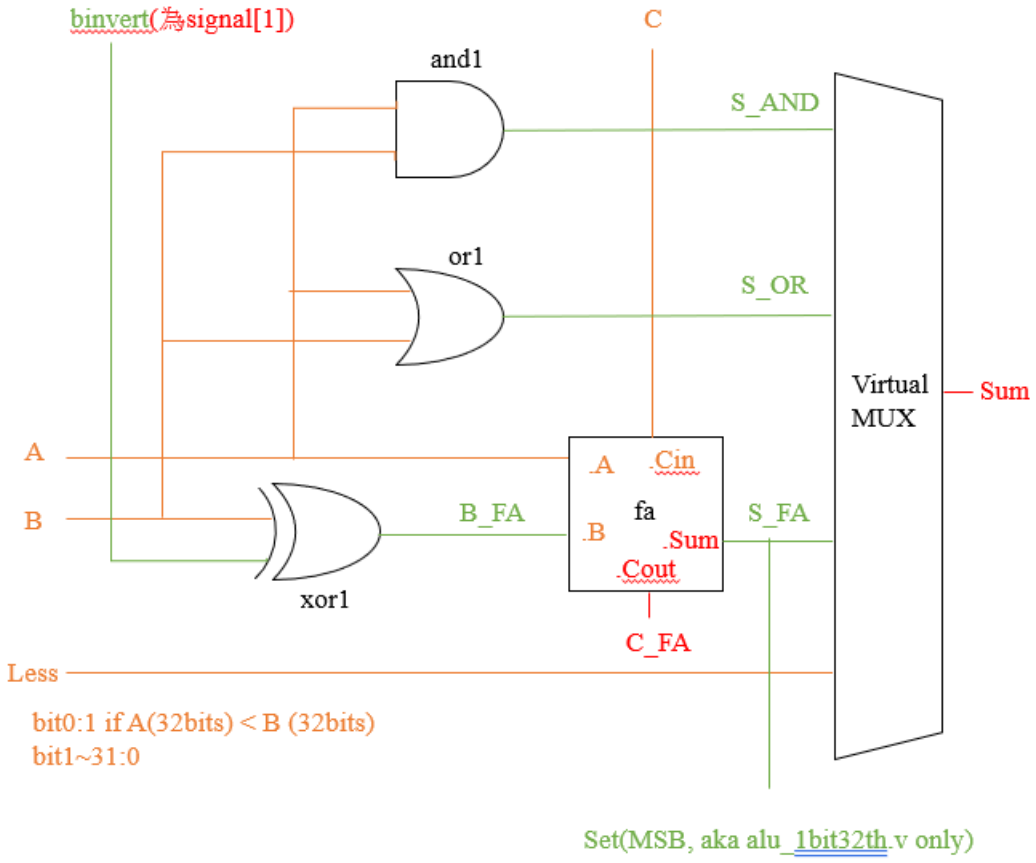
32bits ALU；不同的地方是多了 Zero，若結果是 0 則 Zero 為 true，設為 1，else 設為 0。輸入為 EX_rd1、alu_b、訊號 Operation，輸出 alu_out（計算結果）和 Zero。





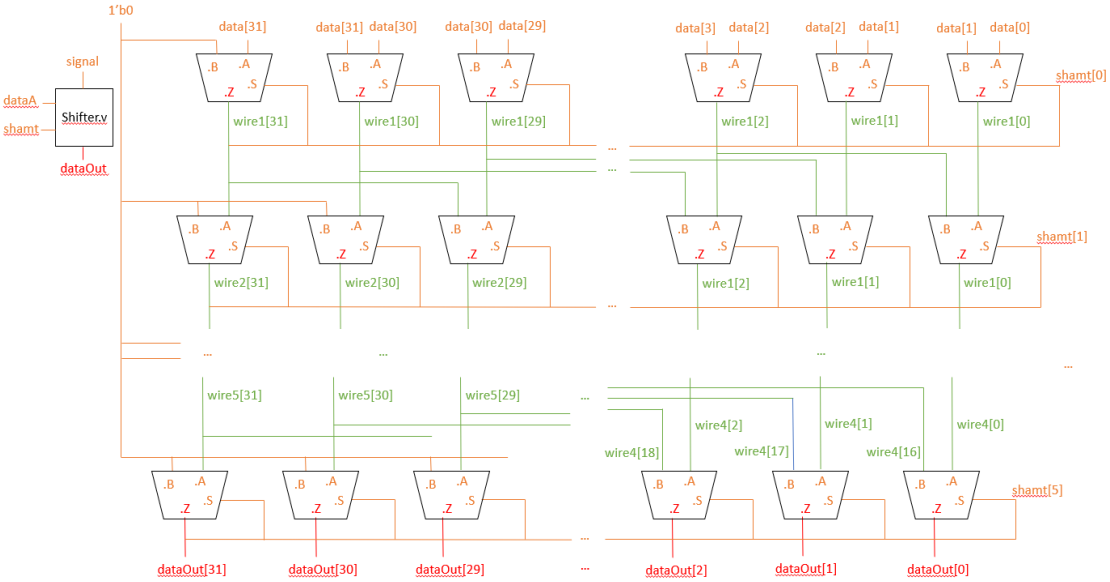






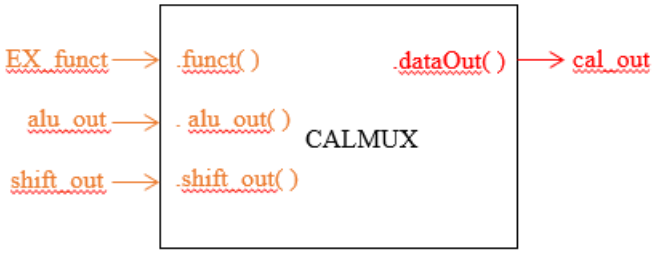
(17) shifter：請參照 Shifter.v、期中報告的 module Shifter.v 說明。

設計內容和期中一模一樣不贅述，輸入 EX_rd2、EX_shamt、EX_funct、rst，輸出到 shift_out。



(18) CALMUX：請參照 mux2_cal.v。

二對一多工器，選擇訊號為 EX_funct，輸出為 cal_out。當訊號為 2 時輸出 shift_out，else 輸出 alu_out。

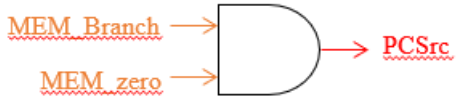


(19) EX_MEM：請參照 EX_MEM.v。

當 clk 敲起把輸入丟出去，詳細請見上面的總架構或直接看 code 不贅述。

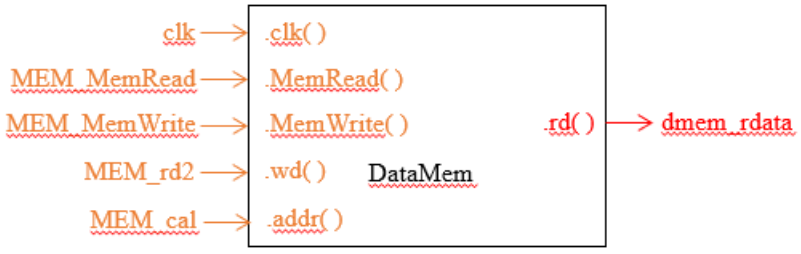
(20) and：

把 MEM_Branch 和 MEM_zero 做 and 運算，丟出到 PCSrc，以實現 BEQ 之操作。



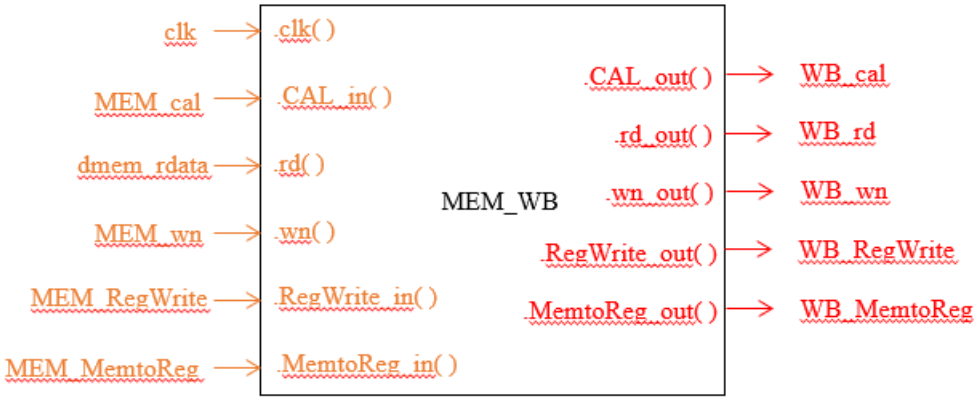
(21) DataMem：請參照 memory.v。

wd 為 MEM_rd2，addr 為 MEM_cal，輸出 rd 為 dmem_rdara，當 clk 敲起且 write enable 時依照位址將 MEM_rd2 寫入，並在 read enable 時依照位址讀入到 dmem_rdata，else 設為 x。



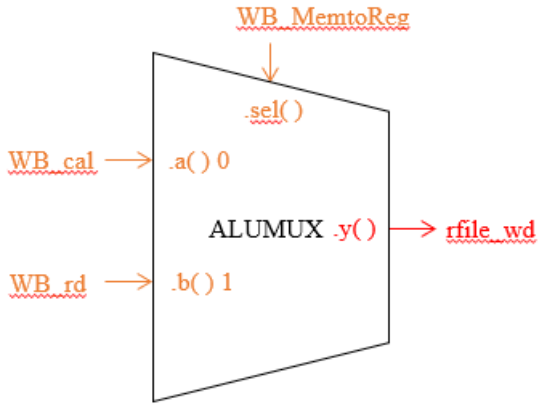
(22) MEM_WB：請參照 MEM_WB.v。

當 clk 敲起時，將輸入丟出去。詳見上面的總架構或直接看 code。



(23) WRMUX：請參照 mux2.v。

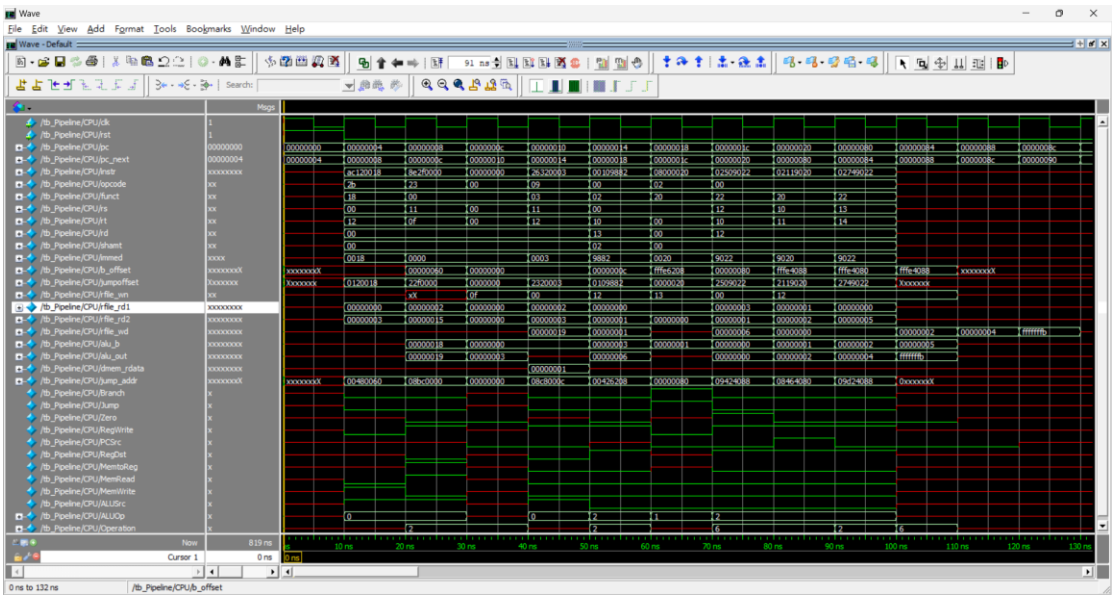
二對一多工器，選擇訊號為 WB_MemtoReg，輸出為 rfile_wd。當訊號為 x 時輸出 WB_cal，else 訊號為 1 時輸出 WB_rd，else 輸出 WB_cal。



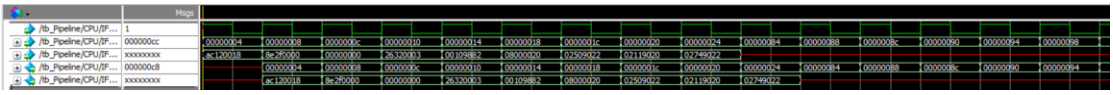
(24) TestBench：請參照 tb_Pipeline.v。

timescale 設為 1ns/1ns，clk 最初為 1，每隔 5 延遲反轉；rst 最初為 1，在讀入指令記憶體、資料記憶體、暫存器與 hilo 後延遲 10 再設為 0。當 clk 正緣觸發時，顯示要執行的指令。整支程式透過實例化 mips_pipeline（稱做 CPU）傳入 clk 與 rst 執行指令並輸出結果。

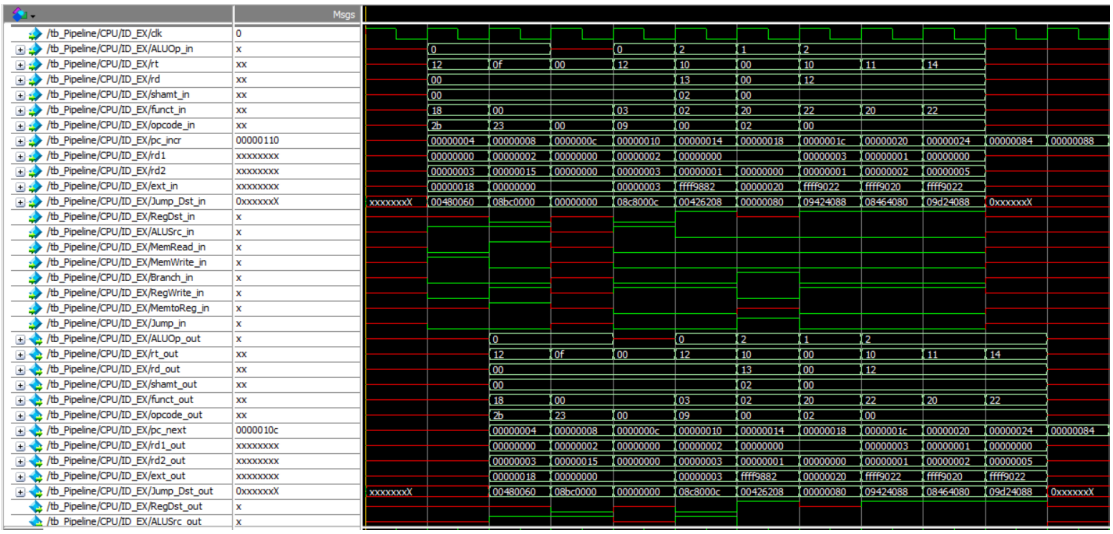
三、結果



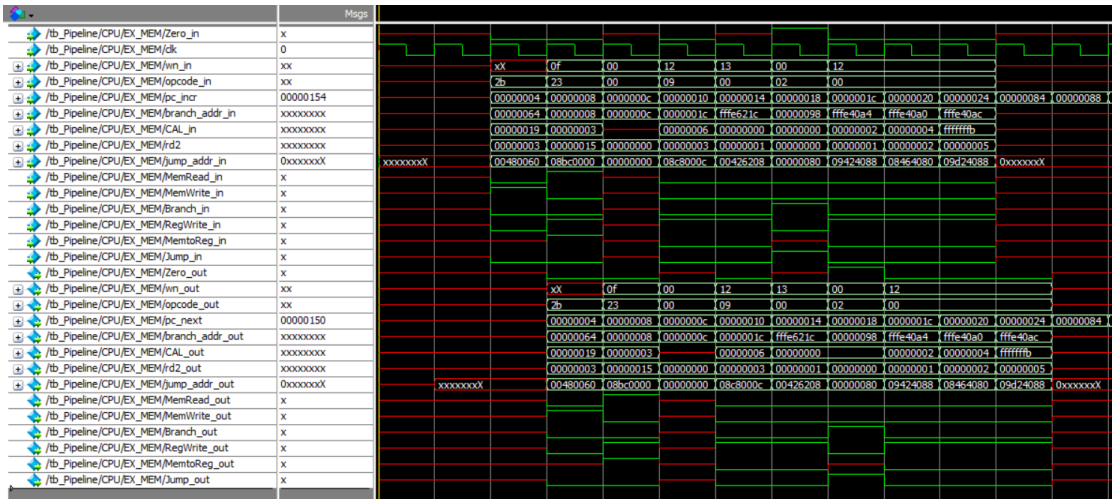
CPU 各接口波形圖



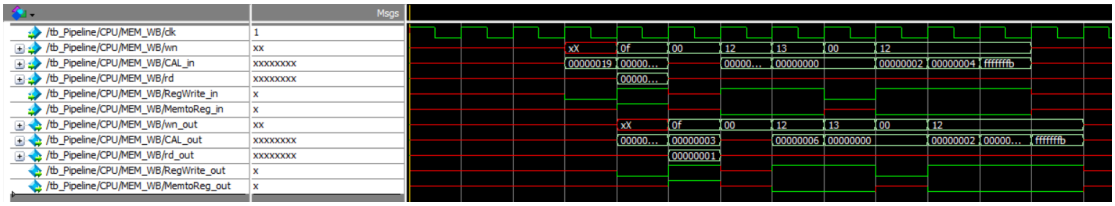
IF/ID Pipeline register



ID/EX Pipeline register



EX/MEM Pipeline register



MEM/WB Pipeline register

四、討論

比起期中，期末的自由度雖然放寬，但是難度卻提升了好幾倍。我們一致認為最困難的地方反而是接線與 datapath 的設計，雖說最後也沒做完。在 module mips_pipeline 中，可以看到一大堆 wire，目前的版本透過 datapath 畫出來會發現有很多跳線，甚至有些傳到後面幾個階段根本沒有用到。若是設計得好，不僅架構圖好畫，接線明瞭，邏輯清晰，甚至能減少效能，將這個所謂的輕量級 CPU 更加減輕負載，module 也能更少，降低複雜度。

五、結論

透過這次 final project，我們大致了解了 MIPS 指令集架構的 CPU 運作邏輯，並且由 IF、ID、EX、MEM 與 WB 了解了所謂的”pipeline”到底是怎麼一回事。由於時間沒有掌握好，沒有完成 multu 與 maddu，連帶地 mfhi、mflo 也沒有實現，實屬可惜。並且，在實作的過程中，我們發現有很多重複的接線宣告與邏輯判斷。光是多工器，除去 mux2_mf.v 這個沒有實做 hilo 而無法用到的 module 以外，就有 4 個版本，且在後幾個階段都是等到 clk 敲起把傳入丟出去，那有沒有一個可能，用一個介面達到省略重複部分的效果？否則，在校譽的部分實在是稱不上表現好。

六、未來展望

第一個點就是補上 multu 與 maddu 與對應的 mfhi、mflo 的部分。其次，試著以上述提到的「簡化」之思路將 CPU 去除冗餘的部分（像是 Java 的 interface 這種概念）。再來，重新整理 datapath，避免過多的跳線。最後，試著設計一塊 PCB，將這個 MIPS-Lite CPU 真正地實體化，實現所謂工程師的浪漫。若有多餘的時間，還可以將所學整理成筆記發布到 HackMD 上，提供他人學習。