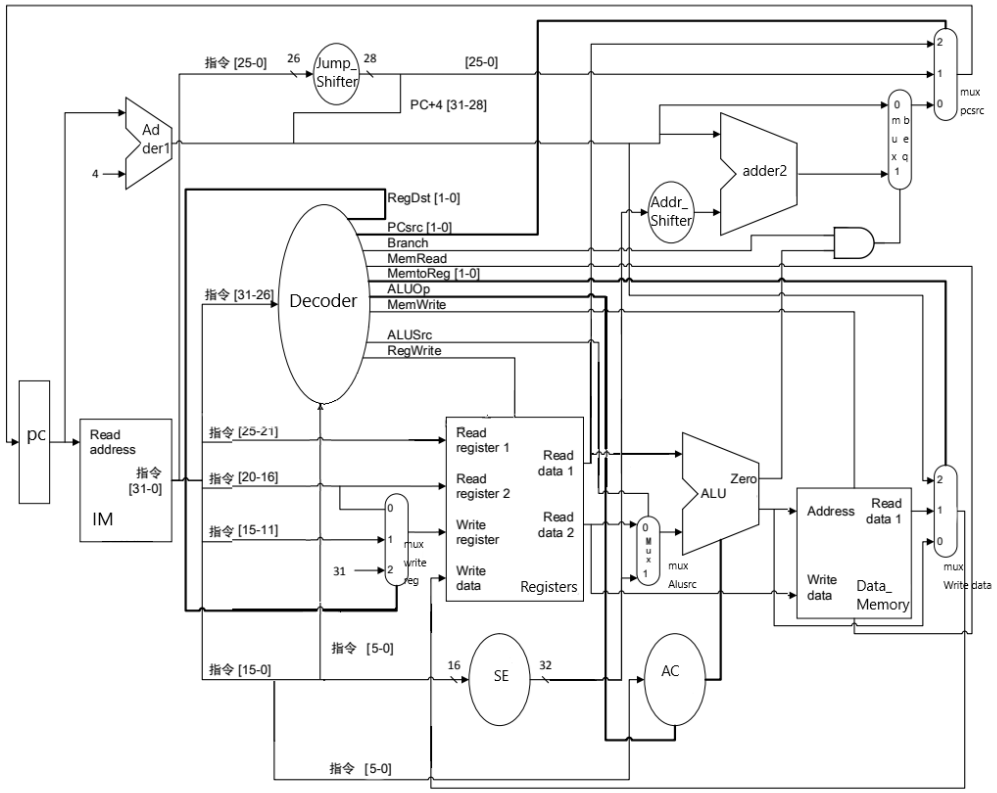
Computer Organization

Architecture Diagram



Hardware module analysis(簡短解釋一下每個 module 的功能，以及在 CPU 裡扮演的角色):

ProgramCounter.v

一開始設定起始地址為零，並隨著 clk 更新出下一個的地址。藉由 clk 更新機制確保一個指令執行完下一個指令才會被讀取。

Instr_Memory.v:

根據 PC 給出的地址從 txt(mem)中提取指令。

Adder1

把現在地址+4，以算出下一個地址候補。

Mux_PC_Source

根據 PCsrc 決定要用哪個地址。0 為 beq or pc+4；1 為 j、jal 指令計算的

地址值；2 為 jr 指令計算的地址值。

Reg_File.v:

根據 RF 給出的指令提取出指定 reg 的數值，並隨著 clk 跟 RegWrite 的真偽決定是否更新 reg 的數值

ALU_Ctrl.v

根據 decoder 產生的 ALUOp 跟指令的 function code 決定要輸入給 ALU 的 alu_option。

Mux_Write_Reg

根據 Decoder 產生的 Regdst 決定使用 rt 還是 rd 還是 31($ra)當作 write reg。不同指令的 write reg 在不同的位置。

Decoder.V

ID Stage。根據 instruction 的 opcode 決定 RegWrite、ALU_OP、ALUSrc、RegDst、Branch。他們分別會影響是否更新 reg、ALU_oprion 的數值、ALU 的 input2、write reg 的值、是否跳轉到另一個地址。

Sign_extend.v

根據輸入的數值的正負，決定補到 32 位時是要補 1 還是 0。避免 16bit 數值在進入 32bit alu 之前因為擴充位元而正負號改變。

ALU.v

根據輸入的的 alu_option，決定如何處裡輸入的兩個數值。輸出結果跟是否為 0 出來。

Mux_ALUSrc.v:

根據輸入的的 alu_src，決定輸入 ALU 的數值。因為 I、Rtype 指令需要處裡的數值分別來自 reg 或是 sign_extend。

Addr_shifter

因為 beq 讀取出的地址需要*4 得到真正的位移數值，所以左移 2。

Adder2

EXE stage。把現有地址+4 跟跳轉數值相加，決定候補地址 2。

Jump_shifter

因為 J、JAL 的地址[27:0]需要*4 得到真正的絕對地址，所以左移 2。

Data_Memory

根據 decoder 的 memread 及 memwrite 值，決定要從記憶體讀出 or 寫入從 write_data 進來的資料。

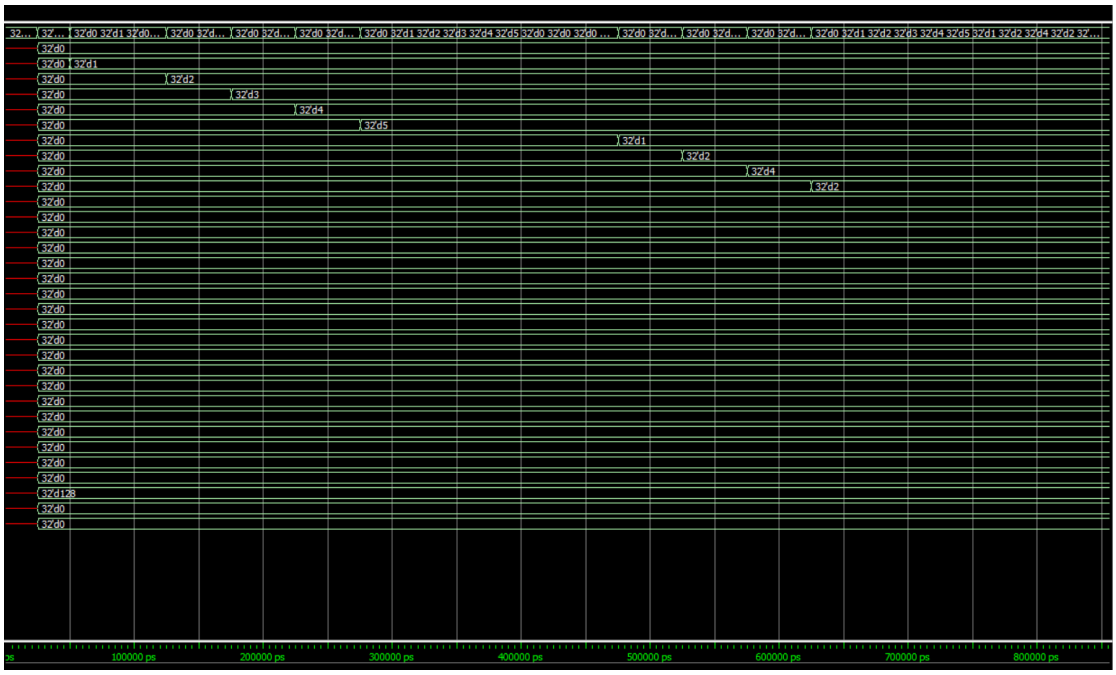
Mux_beq

根據 branch 值跟 ALU 的 zero 決定進入 mux_pcsrc 的 0 號地址為 beq or pc +4。

RESULT:

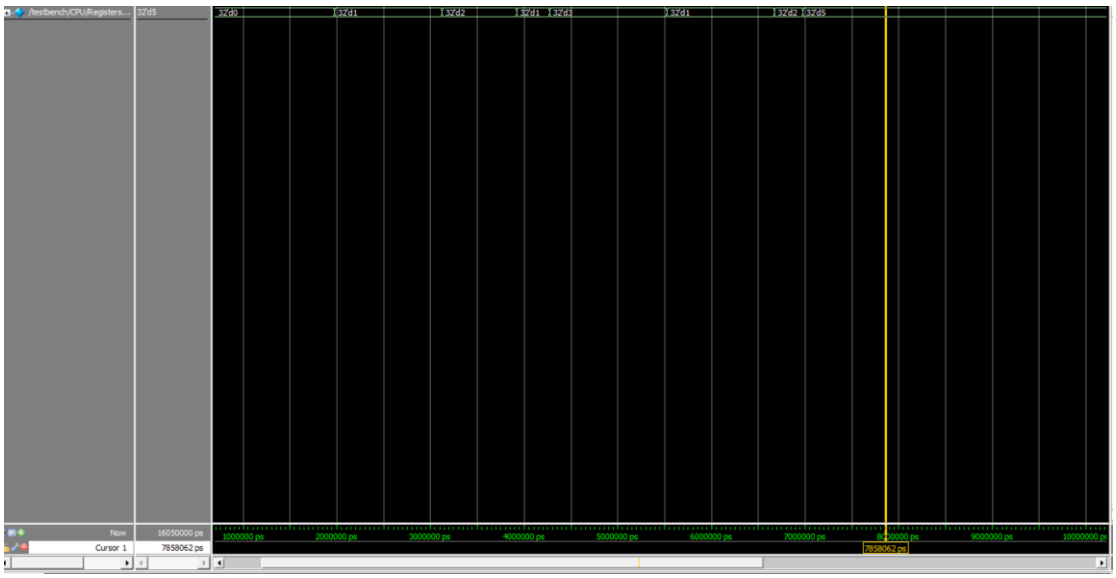
Test1

```
# PC =       1284
# Data Memory =          1,          2,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Registers
# R0 =          0, R1 =          1, R2 =          2, R3 =          3, R4 =          4, R5 =          5, R6 =          1, R7 =          2
# R8 =          4, R9 =          2, 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
# ** Note: $stop    : C:/Users/Andy/Documents/CO/CO_Lab3/code/testbench.v(36)
#    Time: 16050 ns  Iteration: 0  Instance: /testbench
```



Test2

```
# PC =        816
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,         68,          2,          1,         68
# Data Memory =          2,          1,         68,          4,          3,         16,          0,          0
# Registers
# R0 =          0, R1 =          0, R2 =          5, R3 =          0, R4 =          0, R5 =          0, R6 =          0, R7 =          0
# R8 =          0, R9 =          1, 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 =         16
# ** Note: $stop    : C:/Users/Andy/Documents/CO/CO_Lab3/code/testbench.v(36)
#    Time: 16050 ns  Iteration: 0  Instance: /testbench
```



Problems you met and solutions:

Problem1:檔案讀不到
Solution:改成絕對地址
Promblem2: beq 指令壞掉
Solution: 發現是 ALU 中的 zero_o 打成 zero_0
Promblem3: jr 指令壞掉
Solution: 發現是 wire 線接錯，把{pc+4[31:28],jump_shifter}的值放到了 mux_PCsrc 的 date1_i、date2_i 中，所以無法順利跳轉

Summary:

逐漸習慣硬體語言。
只犯了一點小錯誤，例如:
　　接錯一次 wire
Debug 速度大幅提升，清楚知道出了甚麼問題要看哪個 wave。