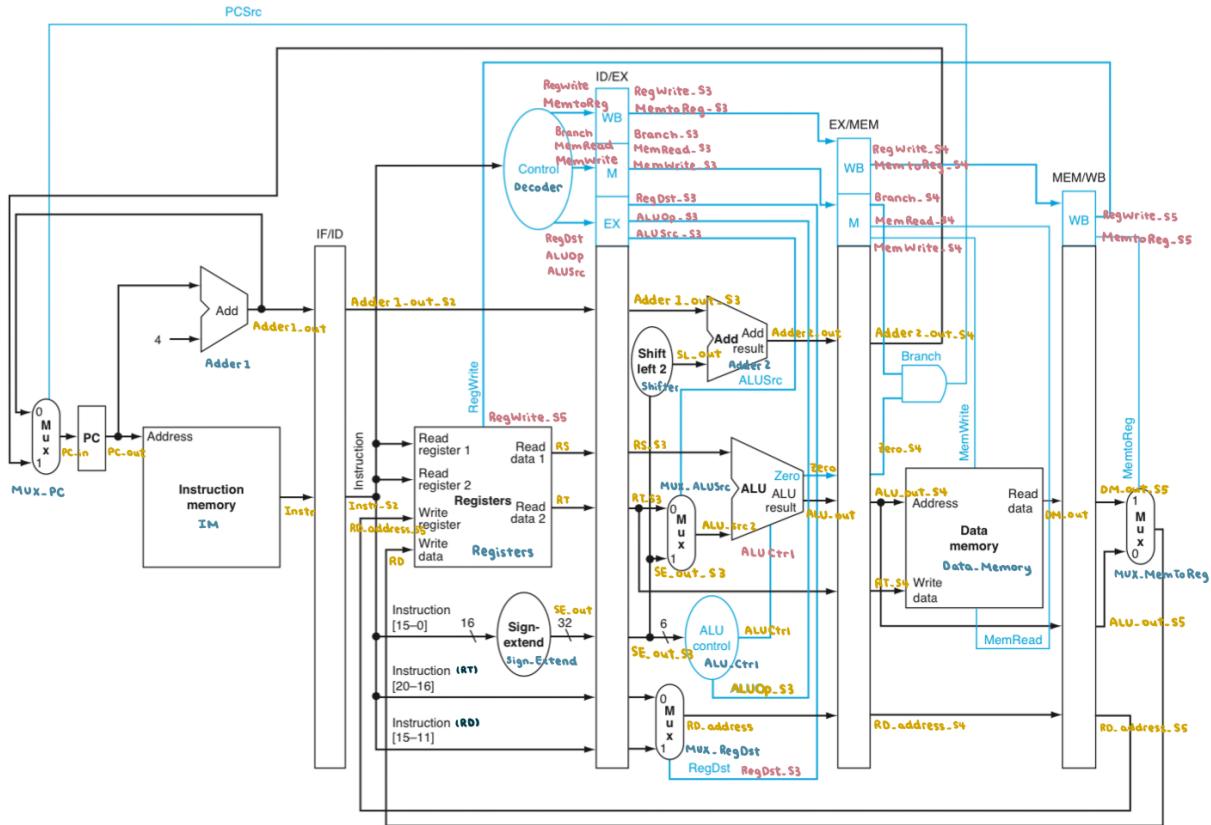


Computer Organization Lab4

Name: 莊婕妤 ID: 109550182

Architecture Diagram



Hardware Module Analysis

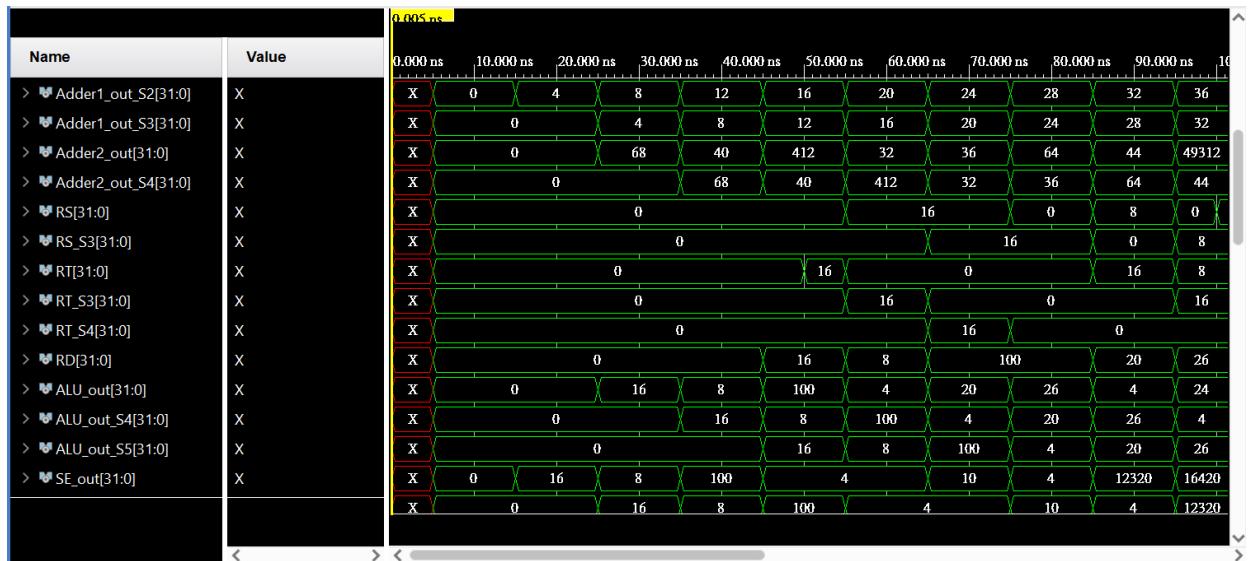
這次我以 Lab 3 的程式碼為基礎，因為不需處理 jal、jr 等指令，因此改變了 Decoder、Mux_RegDst、Mux_MemToReg 的設計，並減少 Branch_Type、Jump 等 Control Signal。所有 Module 的設計以及電路的連接如上圖所述。

Pipeline 的設計與上次最大的不同點在於我們需要講整的 CPU 的運作切分成五個不同的 stage，且須加入 Pipeline Register 將很多指令、變數、Control Signal 打包帶入下一個階段使用，這也使我的 Pipe CPU 更加複雜。

另外，這次的 CPU 有要求需要可以處理 MULT 指令，我改變了 ALU Control 以及 ALU，以完成該指令需求。

Finished Part

Test Data 1



Register=====

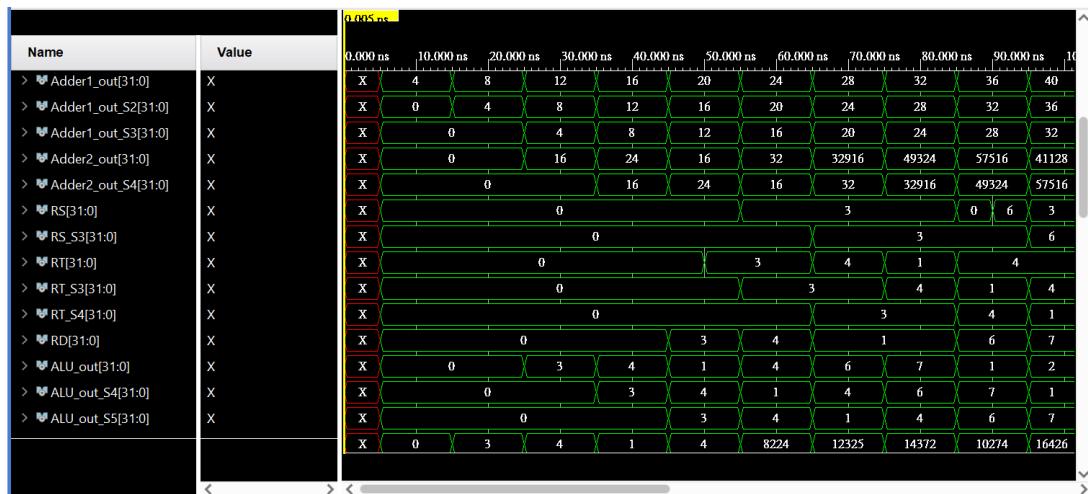
r0=	0, r1=	16, r2=	20, r3=	8, r4=	16, r5=	8, r6=	24, r7=	26
r8=	8, r9=	100, r10=	0, r11=	0, r12=	0, r13=	0, r14=	0, r15=	0
r16=	0, r17=	0, r18=	0, r19=	0, r20=	0, r21=	0, r22=	0, r23=	0
r24=	0, r25=	0, r26=	0, r27=	0, r28=	0, r29=	0, r30=	0, r31=	0

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
r24=	0, m25=	0, m26=	0, m27=	0, m28=	0, m29=	0, m30=	0, m31=	0

上面 Test Data 1 波形圖以及跑出的結果都跟我自己 trace 過整個 assembly code 的結果一模一樣，因此我認為我的電路圖實作是正確的。（我有將所有中間的 wire 當作 output 顯示在波形圖上，並將波形圖的值改成 unsigned decimal 方便我做檢查及除錯）

Test Data 2



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

與 Bonus Part 一同做講解：

為了避免該組測資出現 data hazard 的問題，我將助教提供的 machine code 重新排序成以下，成功讓所有指令在一樣數量的 clock cycle 內執行完成（不需插入空指令），取得正確的輸出結果。同樣的，上面 Test Data 2 波形圖以及跑出的結果跟我自己 trace 過整個 reorder 後的assembly code 的結果一模一樣，因此我認為我的實作是正確的。

```
I1: addi $1,$0,16
I2: addi $3,$0,8
I3: addi $9,$0,100
I4: sw    $1,4($0)
I5: addi $2,$1,4
I6: addi $7,$1,10
I7: lw    $4,4($0)
I8: add   $6,$3,$1
I9: and   $8,$7,$3
I10: sub   $5,$4,$3
```

Problems You Met and Solutions

就即使我理解了老師在課堂中教授的 Pipeline CPU 的運作原理，在剛開始實作此次 Lab 時，我依然沒有頭緒，不知道如何改變之前的設計讓之前的 CPU 轉換成 Pipeline 的形式。在詢問同學以及上網查過資料後，才發現原來是要多設定很多不同 stage 的 control signal 並增加 register 來打包他們給下個階段做使用，讓他們得以做平行處理。實作過程中，我也發現先將電路圖的每一條線上的 signal 都 trace 過，並標記好名稱，對我後來實作 Pipe_CPU 以及 debug 時有很大的幫助。

Summary

就如同上面所述，這次的 Lab 讓我更了解 Pipelined CPU 的實作原理，運用不斷的打包各階段的控制訊號，我們得以在同個 clock cycle 中處理不同指令的不同 stage。這次的 Pipe_CPU_1 相較前幾次的 Simple_CPU 更為複雜，多了更多不同階段的訊號，因此我花了很多時間在除錯上，檢查每一條現是否有接對，也提醒我下次實作時需要更加細心。