國立彰化師範大學電子工程學系

109學年第2學期期末報告

中央處理單元設計之新增MIPS指令

Central Processing Unit Design

Added MIPS instructions

指導教授:張孟洲

班級:電子四

作者:柯竣鑫

民國一百一十年六月

**一、摘要**

　　本次中央處理單元設計期末專題設計，是使用**陳光耀**學長的MIPSR2000 Pipeline版本進行修改及新增額外的指令，學生使用的編譯軟體為ModelSim v10.5.

　　在陳光耀學長的版本裡，我在每種形式都新增了幾個簡單的指令，分別為**R-Type**的XOR NOR SLLV SRAV指令、**I-Type**的XORI LH SH BLE BGT指令及**J-Type**的JAL指令，共計十種額外指令。

　　報告中，會依照學長檔案進行修改，並描述如何擴充這些指令，以及在何種單元下進行程式的撰寫。最後會使用陳光耀學長所使用的test\_bench(每種type都有自己的textbench)來驗證所撰寫的程式是否正確並附上模擬圖。

壓縮檔內有按照新增的順序分成**1st R-type**、**2nd I-type**、**3rd-Branch**以及最後嘗試做的**J-type**，各版本提供參考!

**二、R-type設計**

　　 首先，先說明R-Type的XOR與NOR指令。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **opcode (6)** | **rs (5)** | **rt (5)** | **rd (5)** | **shamt (5)** | **function (6)** |

在R-type指令中，opcode皆為000000，這部分在Control程式中光耀學長已撰寫一個判別是否為R-Type的指令，所以只需要幫ALU新增新的運算即可使用(圖1)。

因此，我們找到MIPS訂定的Function code，分別是XOR 100110和NOR 100111加入ALU\_Control的parameter，並新增控制線sel (XOR 1001 NOR 1010)，使其與ALU做連結ALU才能使用(圖2)。

ALU部分我們必須先新增sel的parameter，並增加兩個運算，分別是使data\_1和data\_2 做xor及nor (圖3)。

Test\_bench這部分我研究了一下光耀學長memory儲存方式後，了解到放指令的順序，利用先前計算機組織學到的寫法(圖4)，我們就能輕鬆地寫出我們要的指令了。

由於一開始暫存器內都是空的，我們必須先加入數值進去(圖5)，才能使用XOR及NOR，所以我就先依序丟入0、1、2，在兩兩做XOR，得到為1、3，再拿0、1做NOR，得到4294967294(-1) (圖6)。

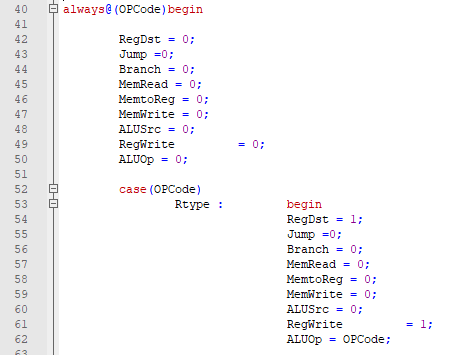
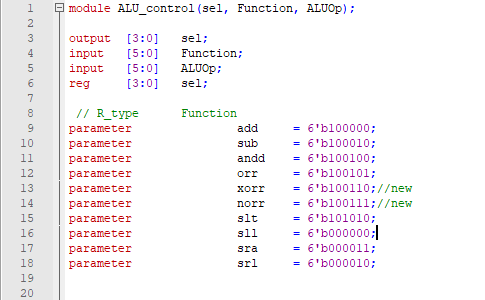
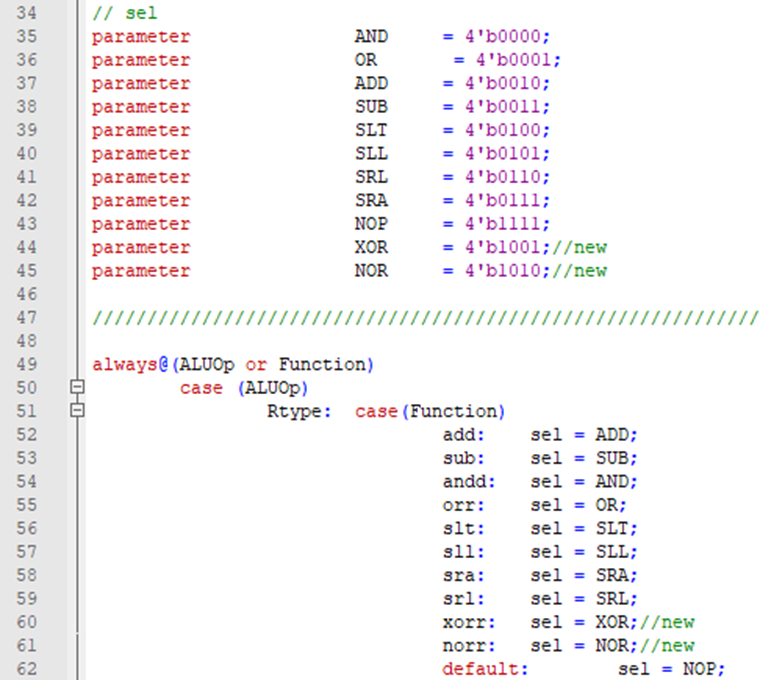
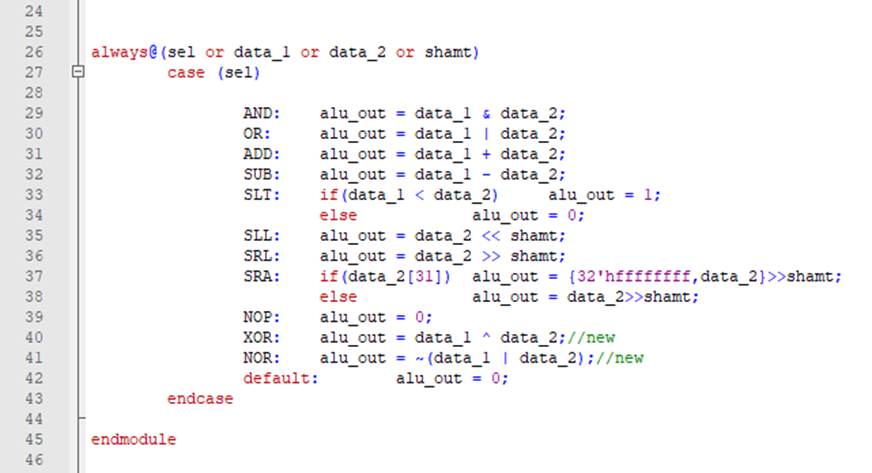
接著說明R-Type的SSLV SRLV SRAV。

在R-type指令中，opcode皆為000000，而MIPS訂定的Function code分別是SSLV 000100、SRLV 000110、SRAV 000111。

ALU\_\_control的部分必須新增sel，使用SSLV 1100、SRLV 1101、SRAV 1110，並新增Function的parameter (圖7)。

ALU部分我們必須新增sel的parameter，並增加三個運算，分別是使data\_2左移data\_1、data\_2右移data\_1及data\_2右移data\_1(並將空出來的位置使用0填充) (圖8)。

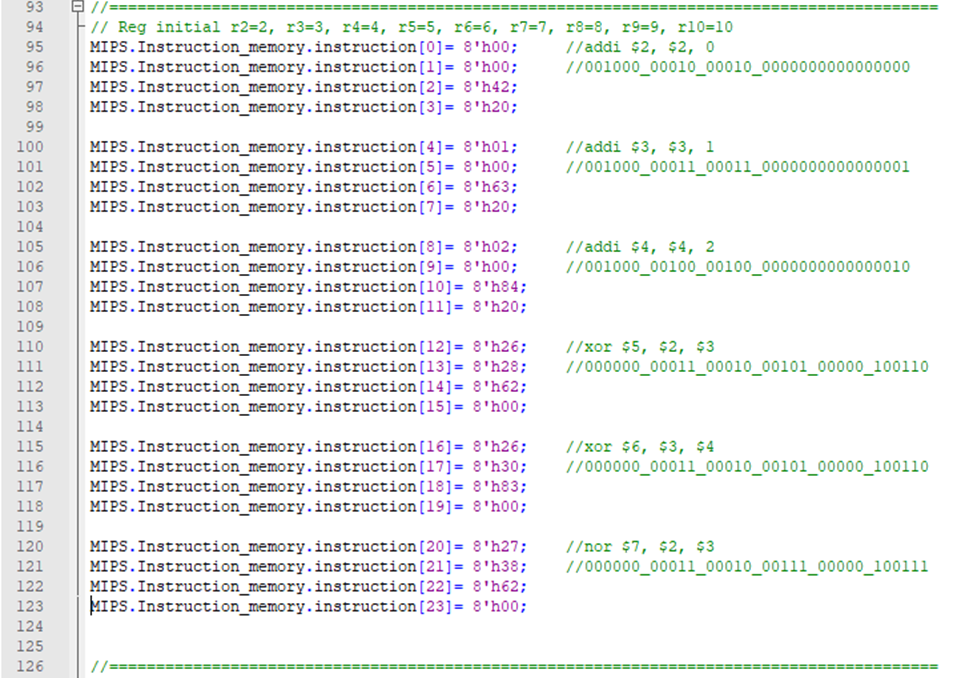
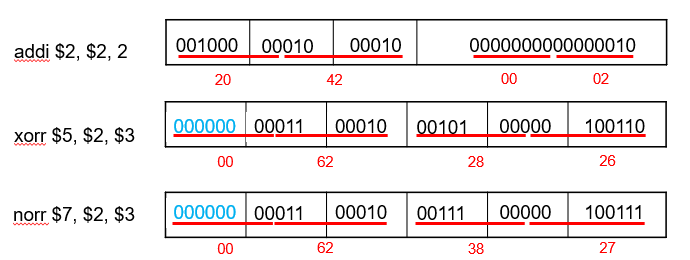
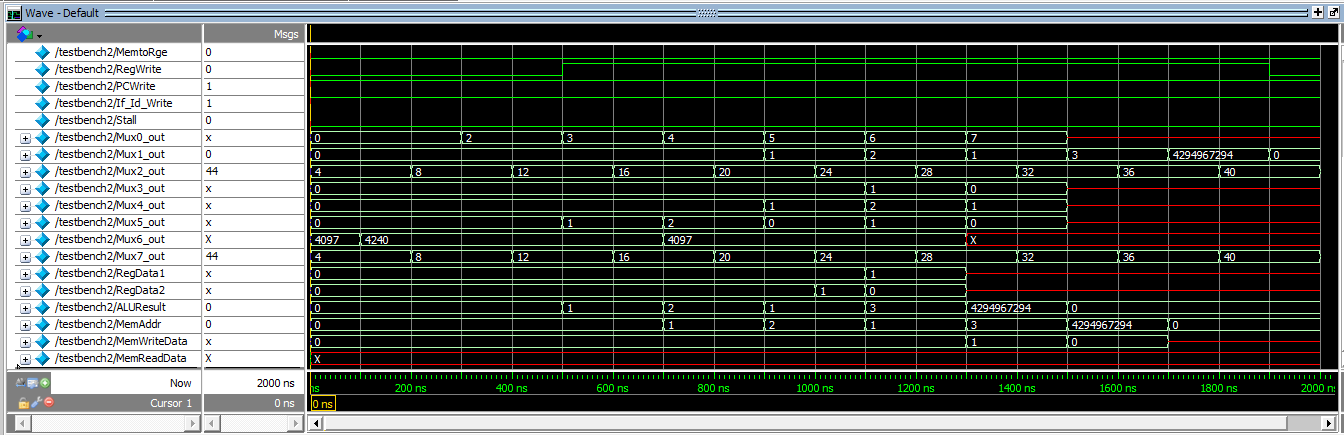
Test\_bench的部分我使用2跟4作為我的data\_1和data\_2， 00100左移兩位為10000 (16) (圖9)，00100右移兩位為00001 (1) (圖10)，測試後符合指令功能。



(圖3): ALU

(圖2): ALU\_Control

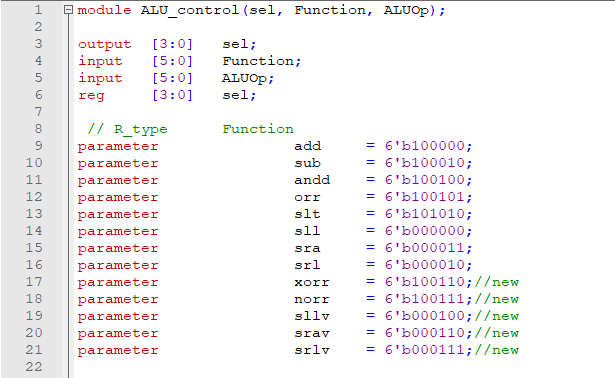
(圖1): Control

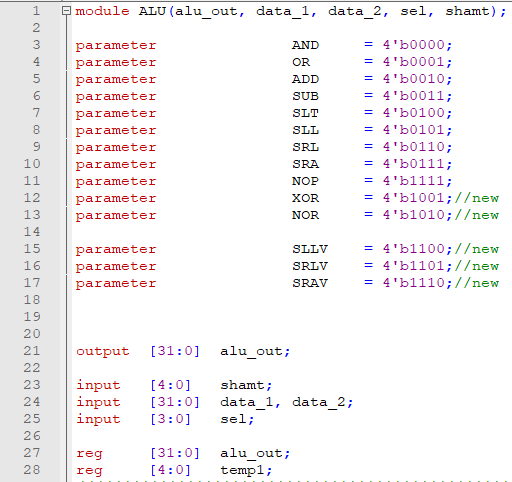


(圖6): wave

(圖5): test\_bench

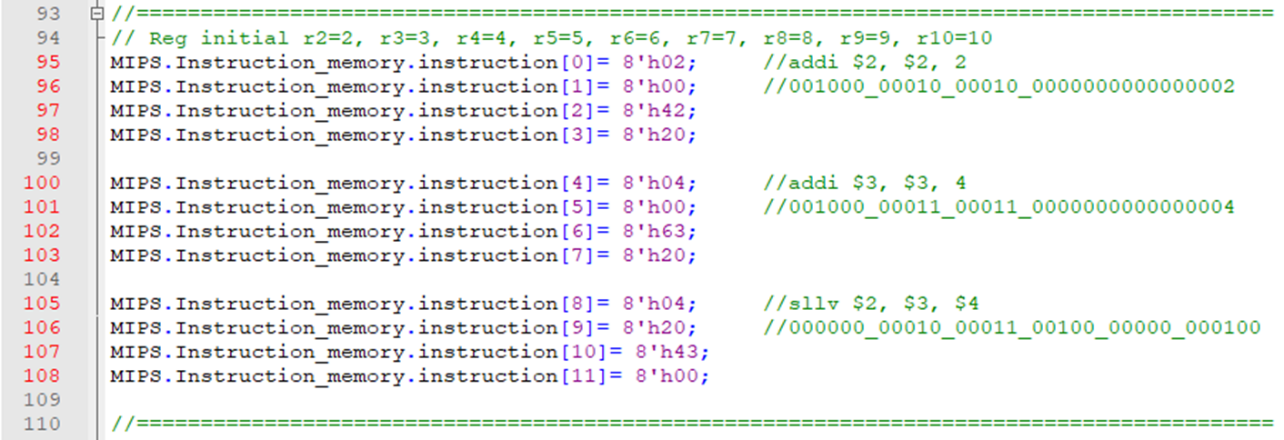
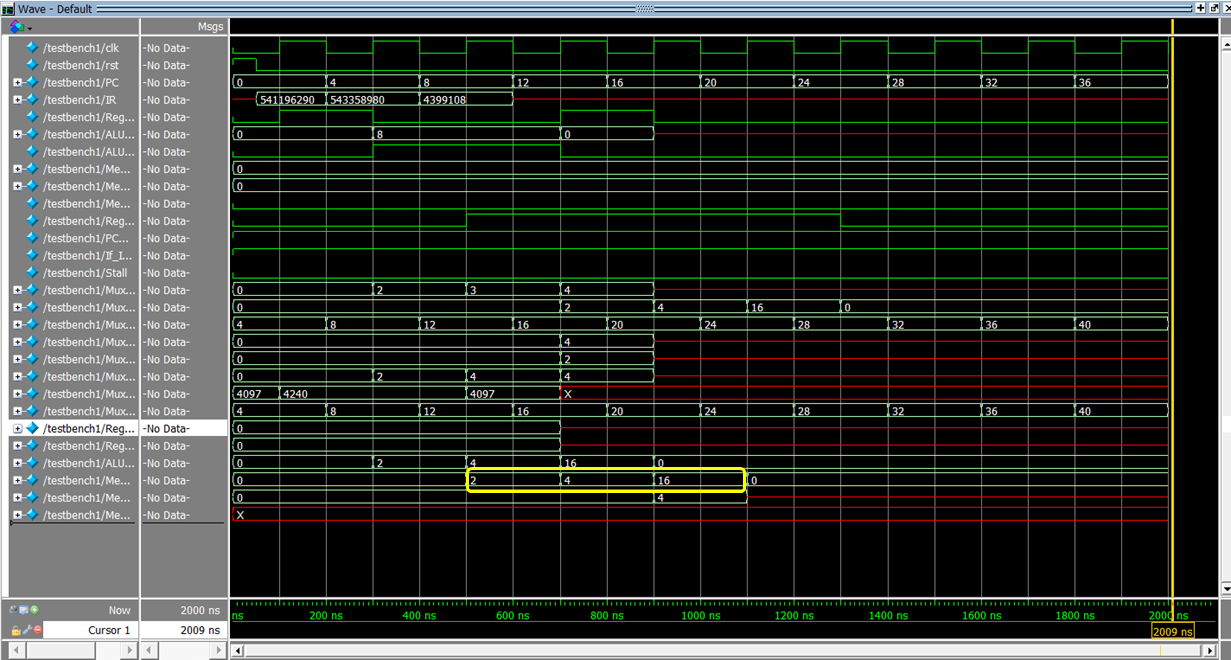
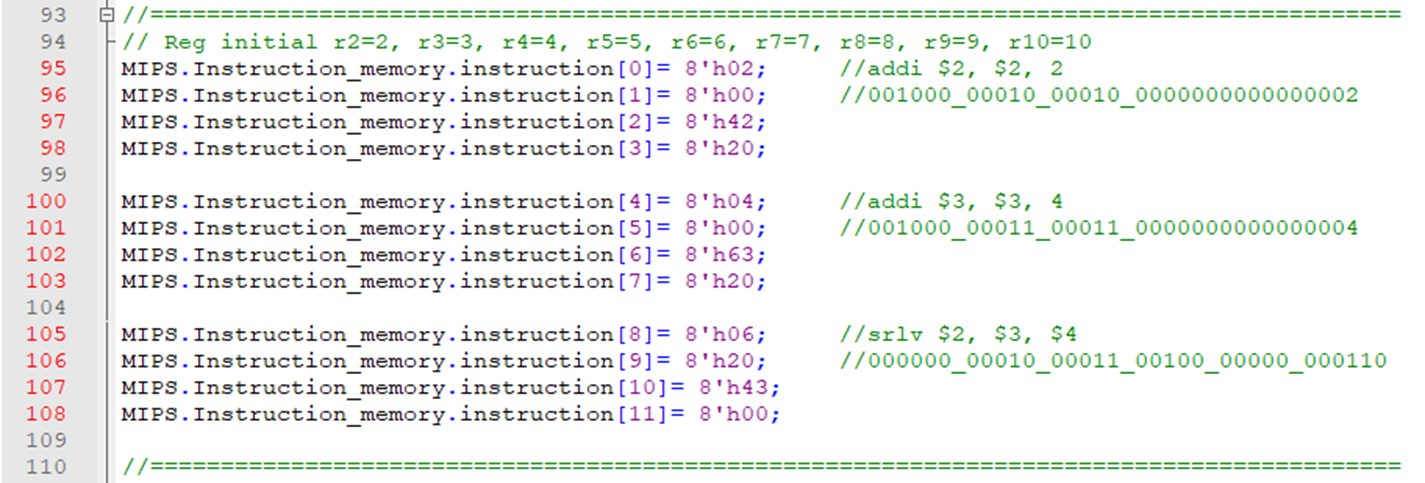
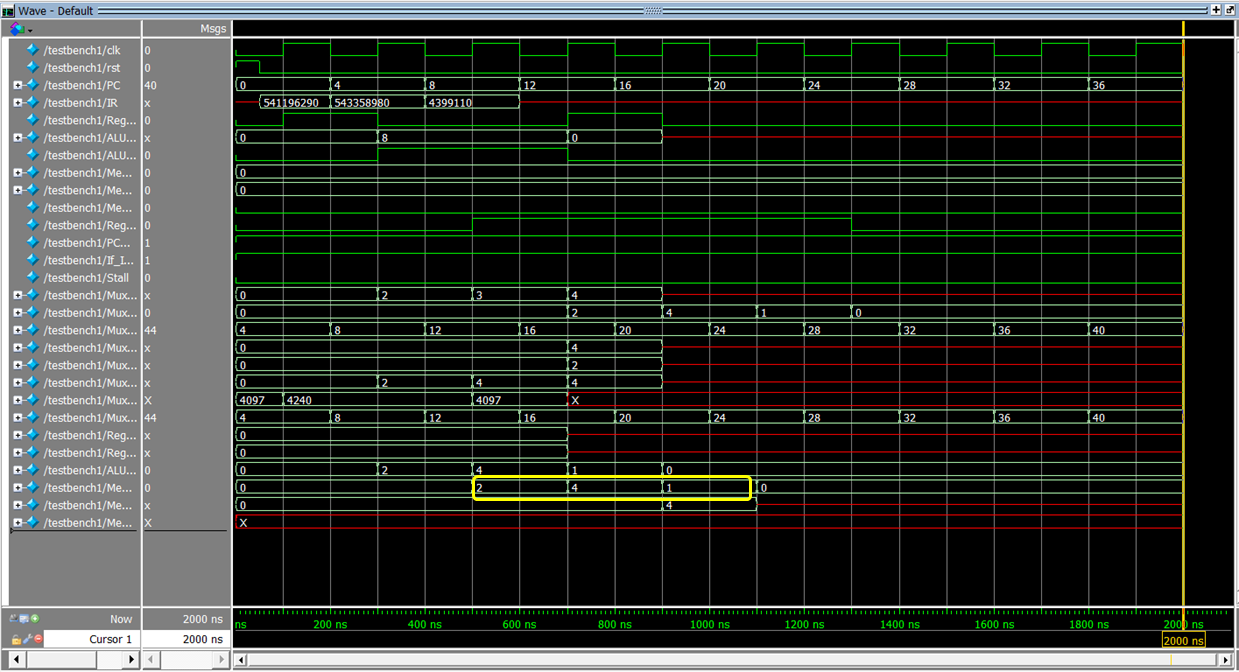
(圖4): instruction





(圖8): ALU

(圖7): ALU\_Control



(圖10): srlv testbench and wave

(圖9): sllv testbench and wave

**三、I-type設計**

　　首先，先說明I-Type的XORI指令。

|  |  |  |  |
| --- | --- | --- | --- |
| **opcode (6)** | **rs (5)** | **rt (5)** | **Immediate(16)** |

在I-type指令中，opcode需要自行新增，如:XORI 001110由於在R-type已經有說明相關功能了，這邊就不多加贅述(圖11)，而且我們已經有新增XOR的運算式，只需要修改成使用immediate的方式就是XORI了(ALUSrc用來sign-extend immediate)。(圖12)

Test\_bench這部分我使用xori $2, $5, 1($2內為0)做XOR為1，xori $3, $6, 2($2內為1)做XOR為3，符合設計功能(圖13、圖14)。

接著新增lh sh指令，其opcode為100001 101001，這邊有發現學長的lw sw opcode有誤，修正為100010 101010 (圖15)，sel的部分選擇為ADD，功能上lh 為將記憶體資料讀入暫存器(圖16)，MemtoReg 和 RegWrite設為1，MemRead為halfword形式；sh 為將暫存器資料寫入記憶體，因此MemWrite為halfword形式。

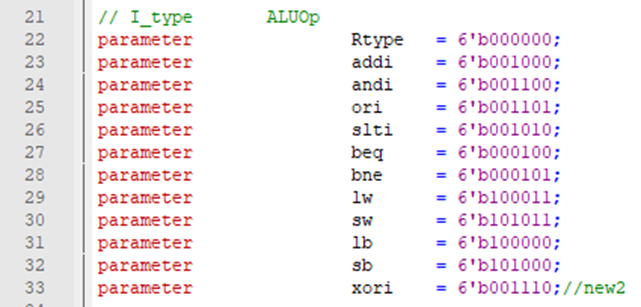
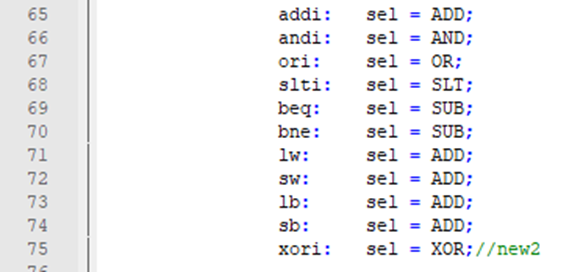
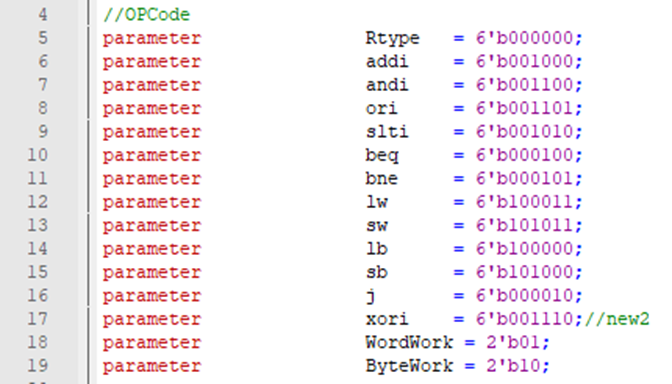
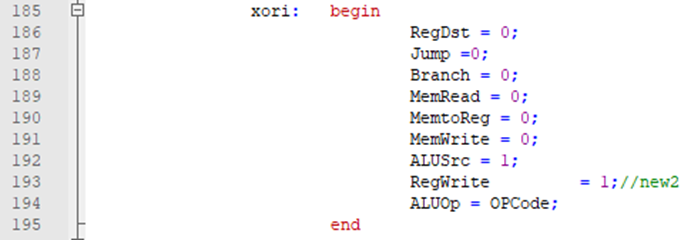
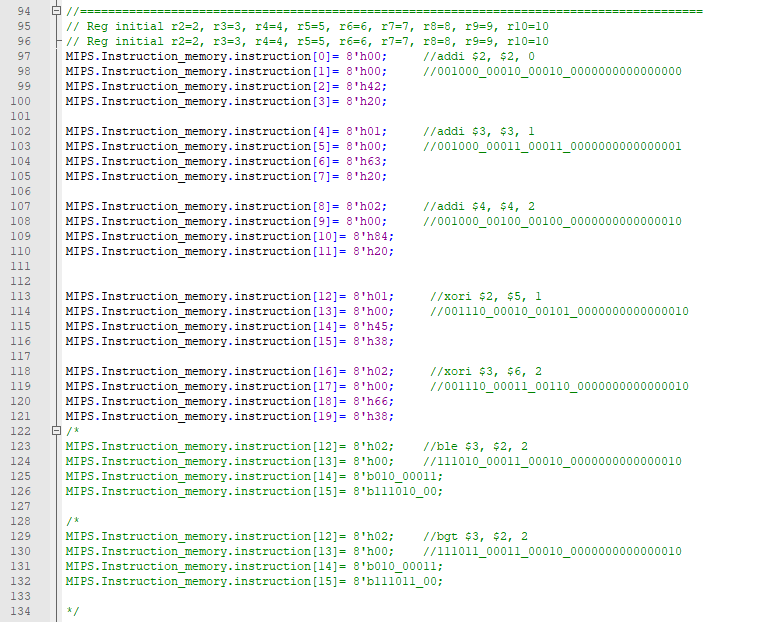
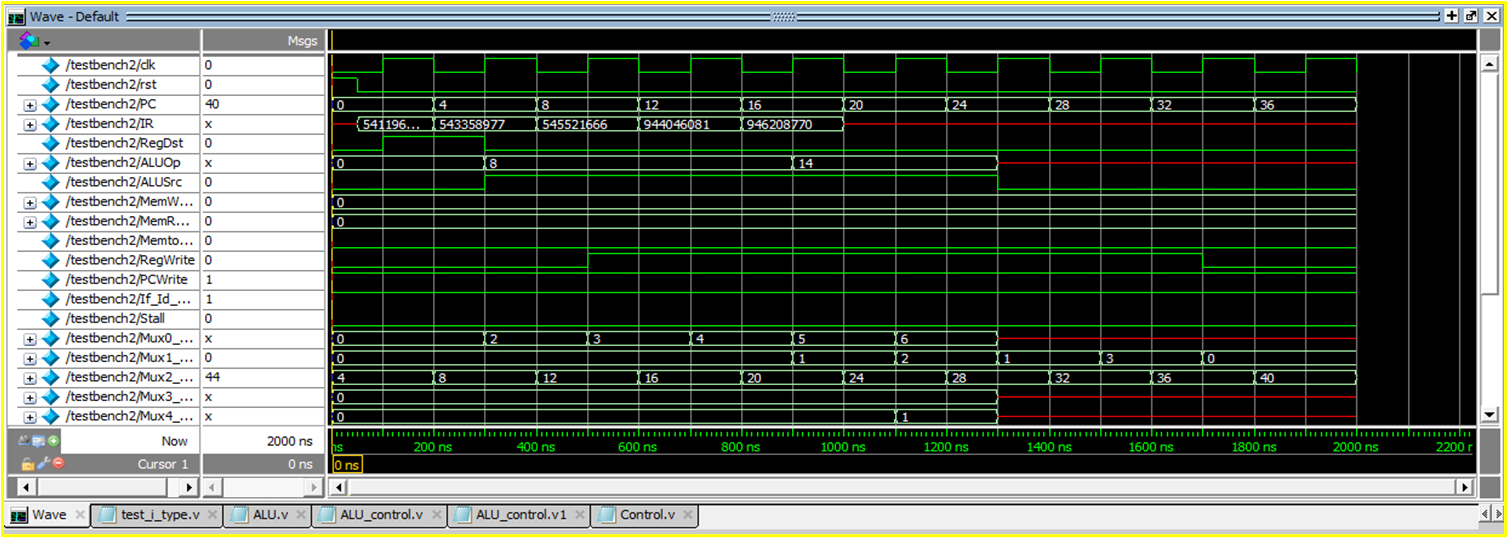
因為新增了halfword的格式(圖17)，在Data\_memory部分需增加case，其意義為存入半個word，因此只存入data[7:0]和[15:8]。

Test\_bench的部分我就簡單地將記憶體的1234做載入暫存器後，轉存於記憶體中(圖18、圖19)。

Branch的部分，先加opcode BLE 111010 和BGT 111011(圖20)於ALU\_Control，再於Control將我們要的功能設為1(圖21)，這邊光耀學長已經有寫過BNE BEQ了，稍微改一下他Branch內的值(00、01、10、11)，就能多新增兩個指令上去。

Branch\_Jump\_Detection的部分稍微改一下判斷式，當他讀到不同的Branch值時，做出不同的判斷(圖22)。

Test\_bench的部分我只舉例BLE(Branch on Less than or Equal)，BGT (Branch on Greater Than)也是同樣原理所以我就不寫了。將暫存器分別存入3、4做交換比較，3 <= 4 條件成立為 Ture，輸出1；4 <= 3 條件不成立為 Flase，輸出-1 (圖23)。

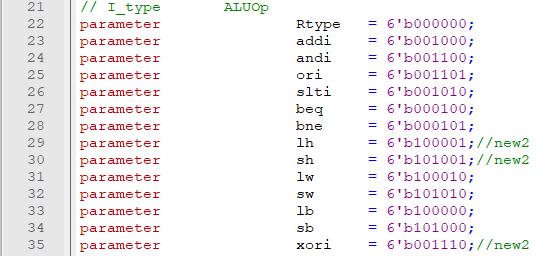


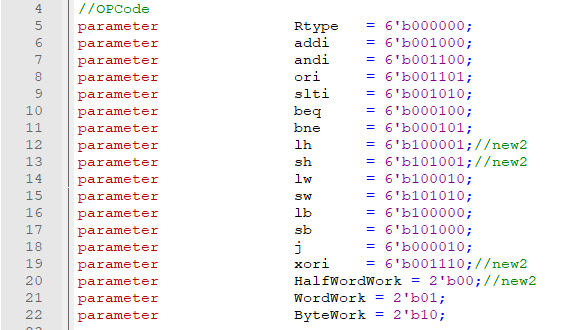
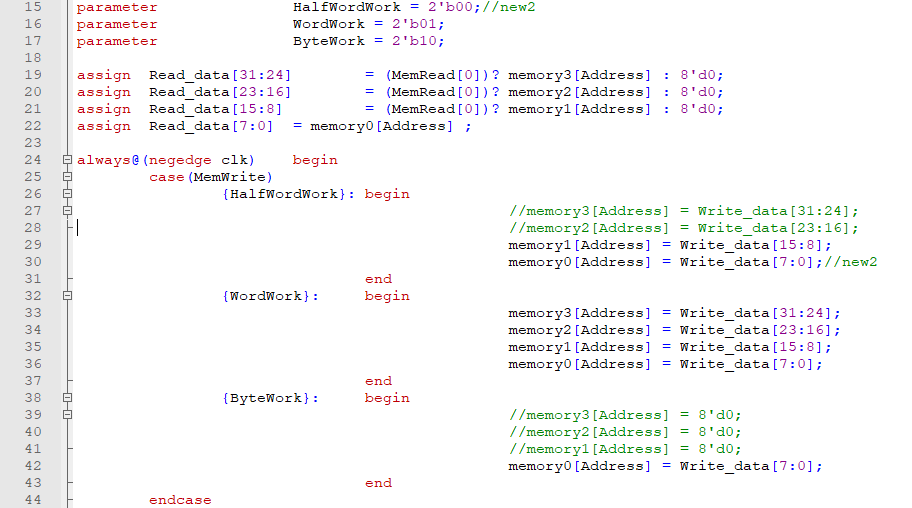
(圖12): Control

(圖14): wave

(圖13): test\_bench

(圖11): ALU\_Control

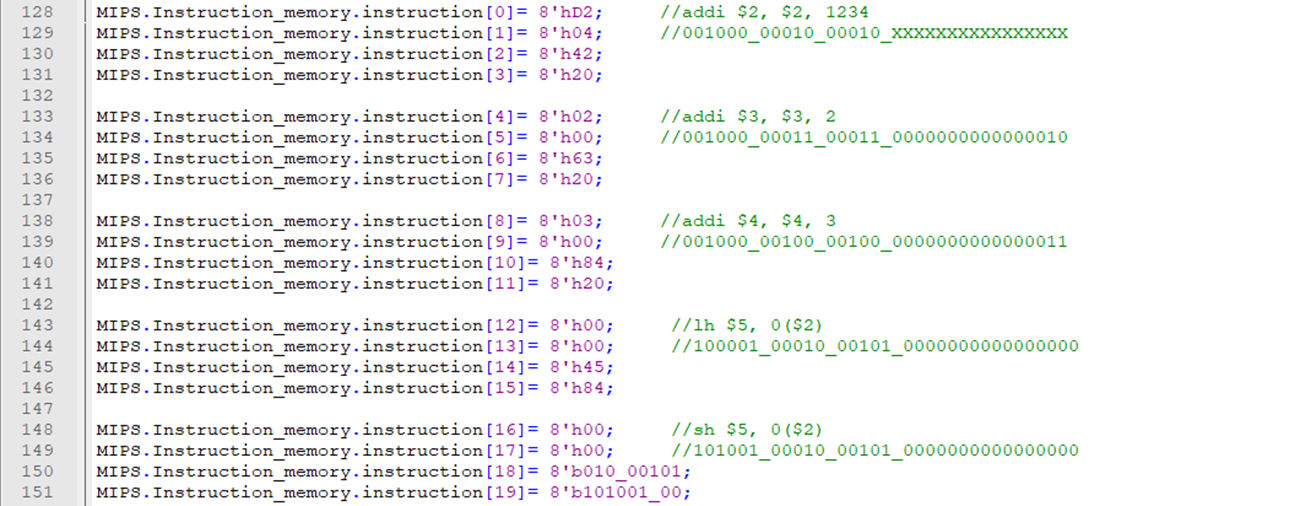


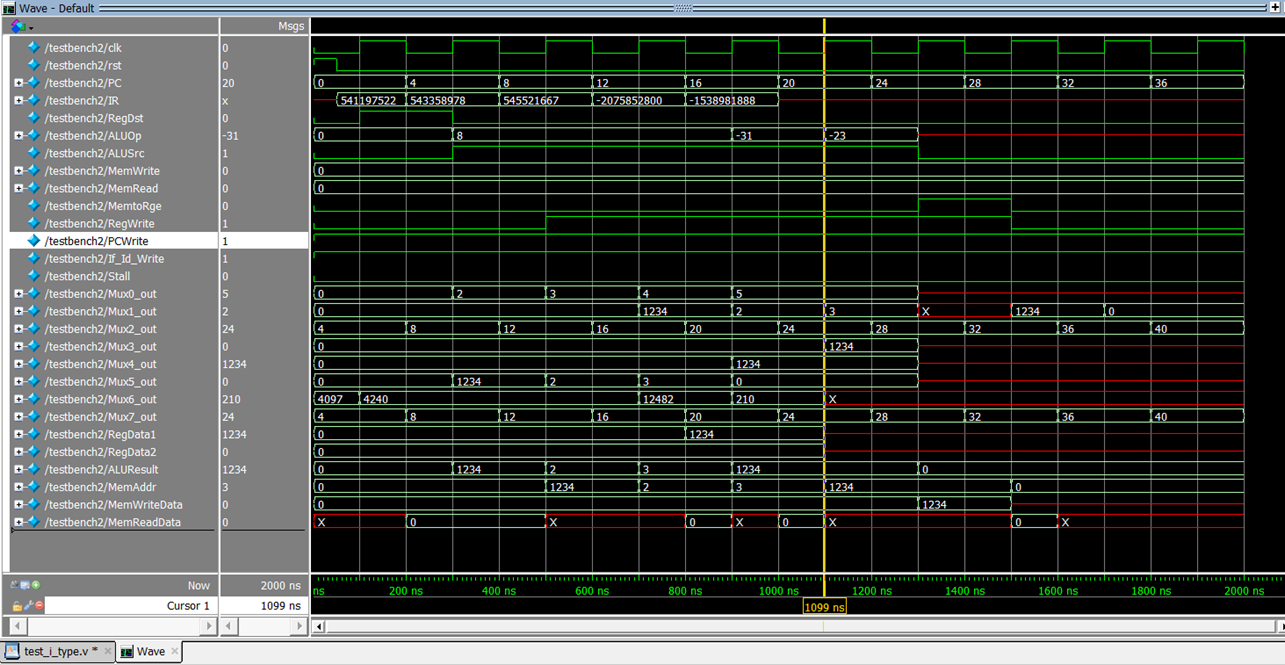


(圖17): Data\_memory

(圖16): Control

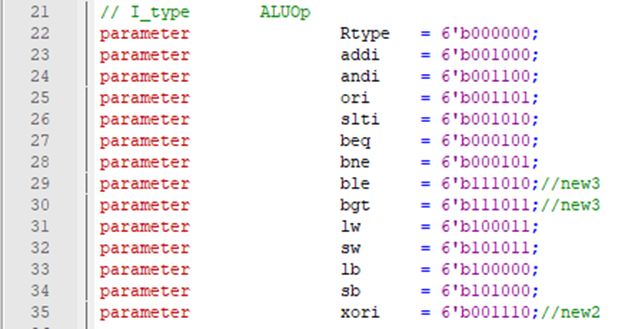
(圖15): ALU\_Control

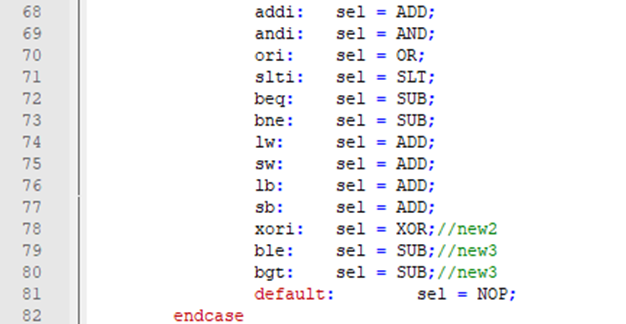
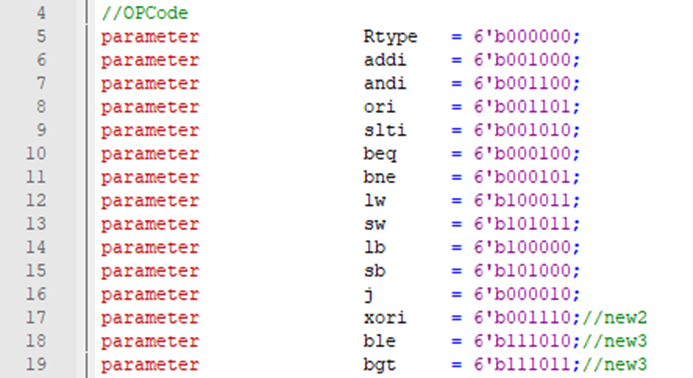
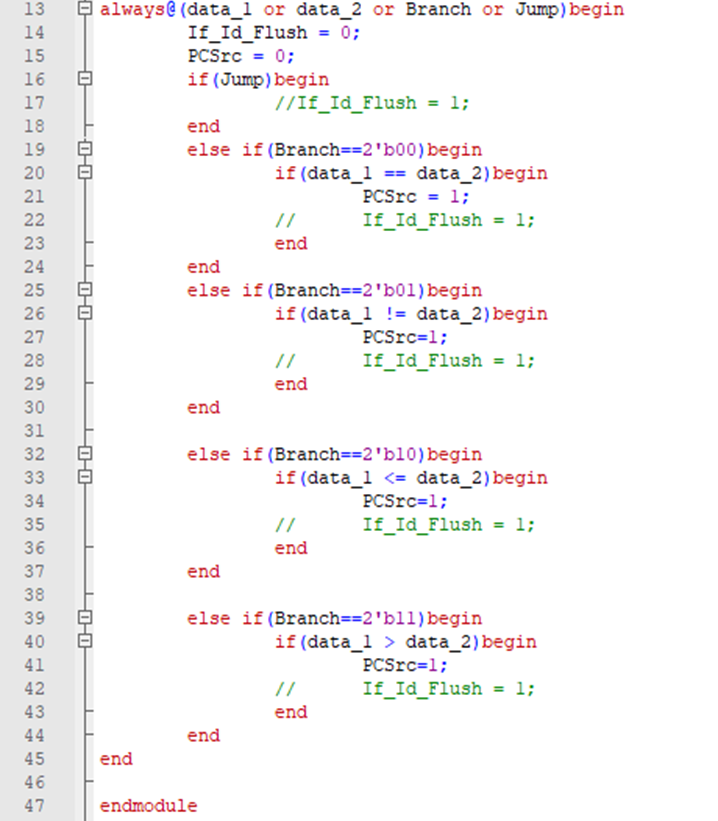
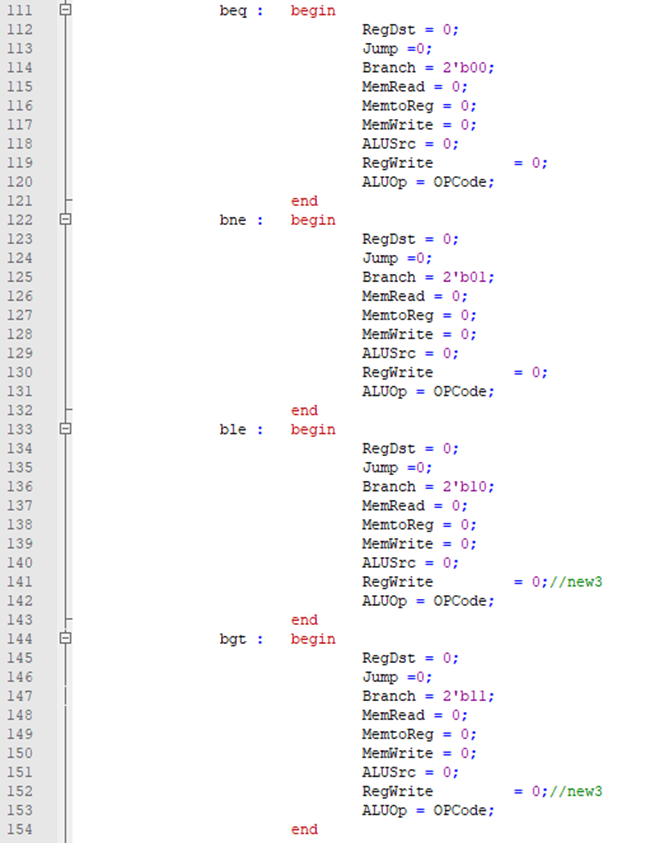




(圖19): wave

(圖18): Textbench

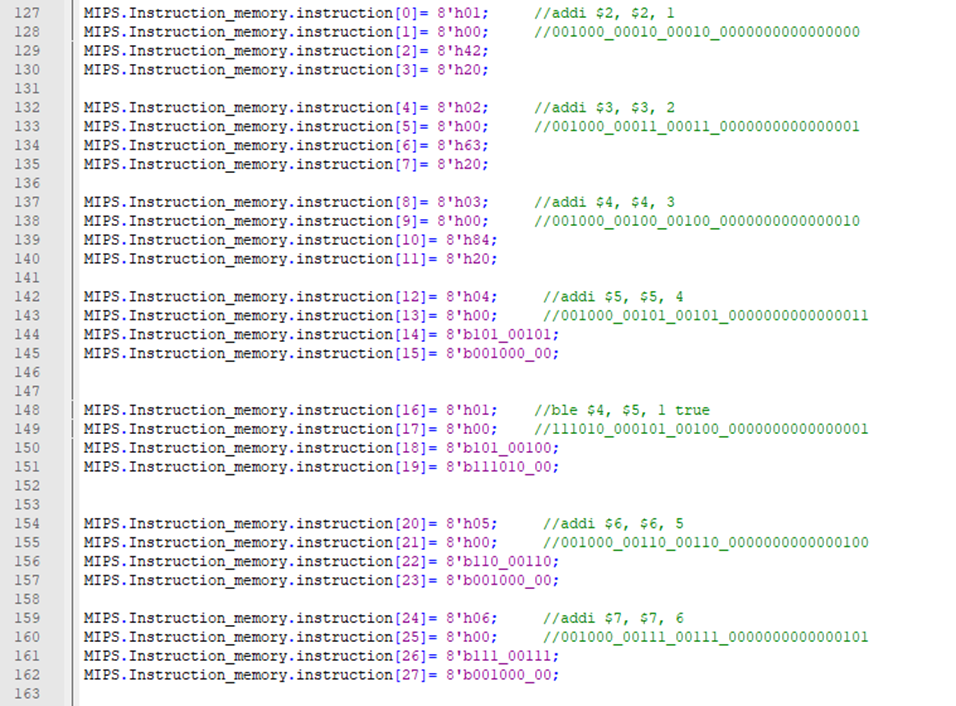
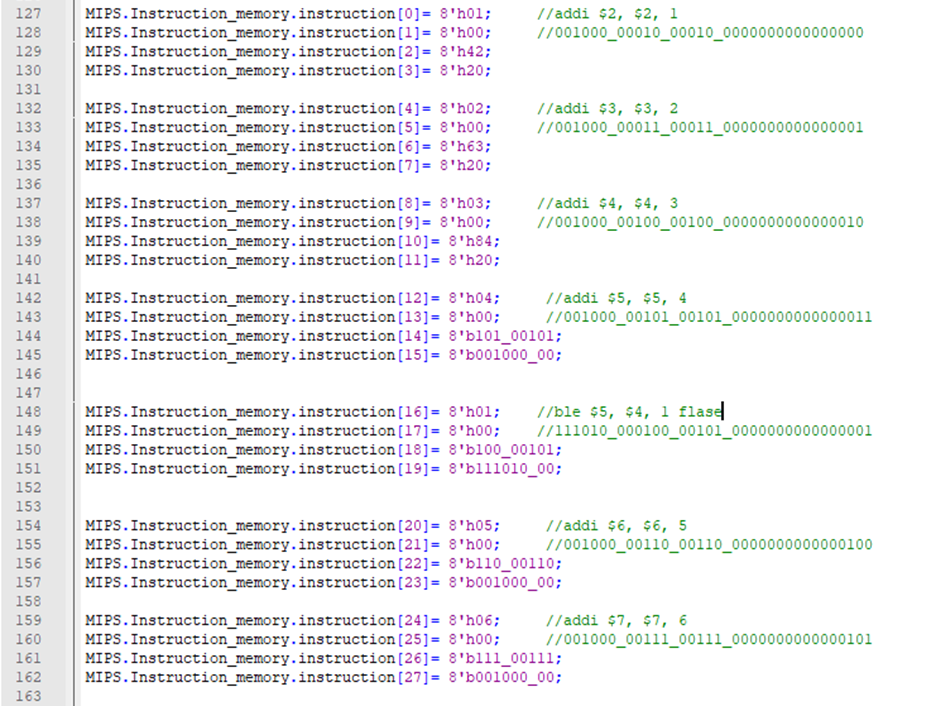


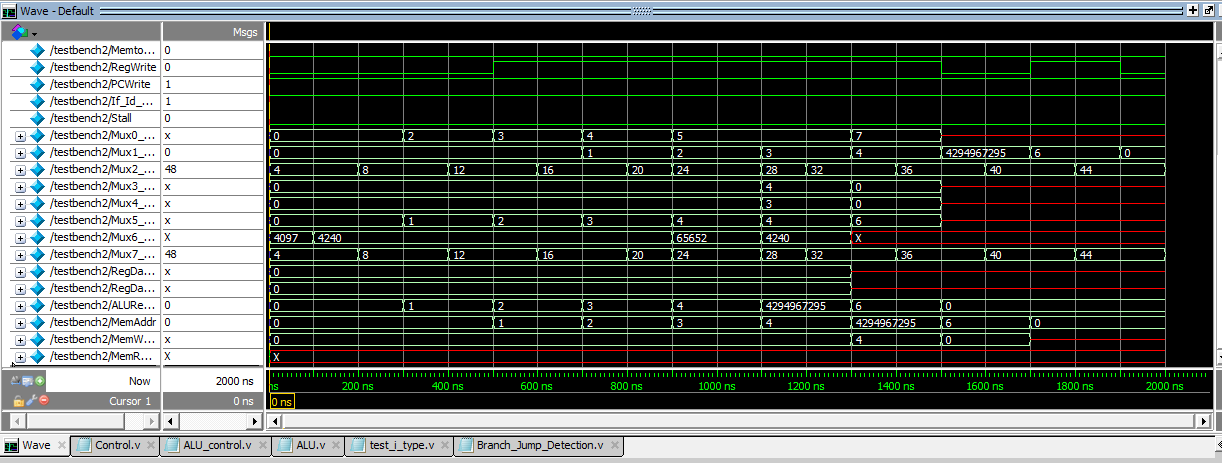
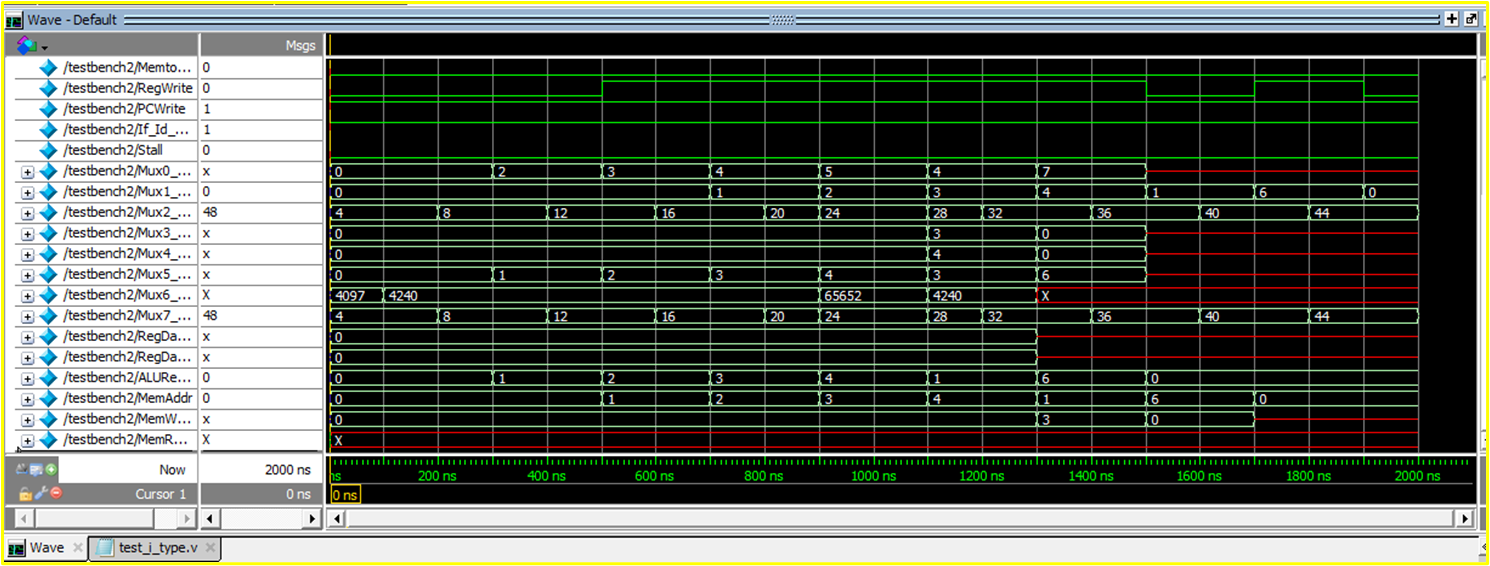


(圖21): Control

(圖22): Branch\_Jump\_Detection

(圖20): ALU\_Control





(圖24): wave

(圖23): test\_bench

**四、J-type設計**

　　 JAL與J的差異是JAL要將下一個指令的位子記錄在R31。

|  |  |
| --- | --- |
| **opcode (6)** | **Target(26)** |

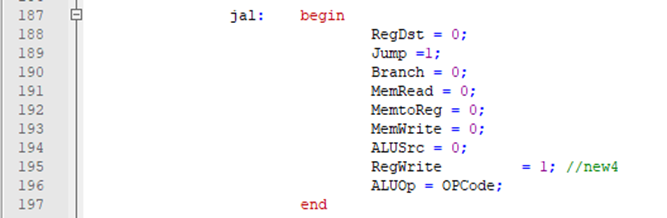
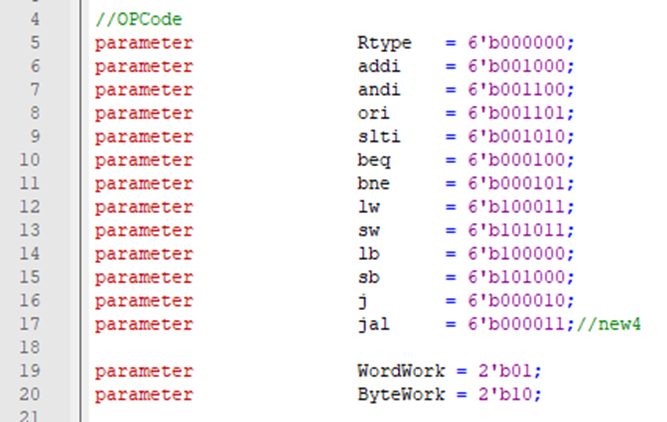
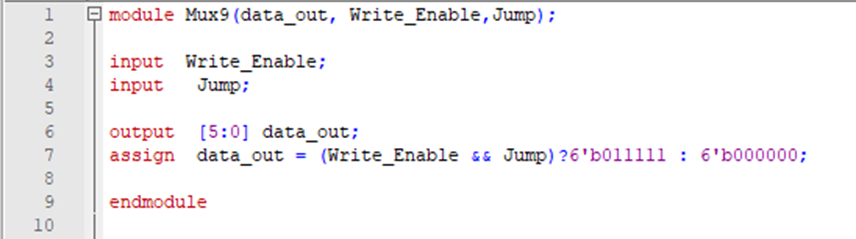
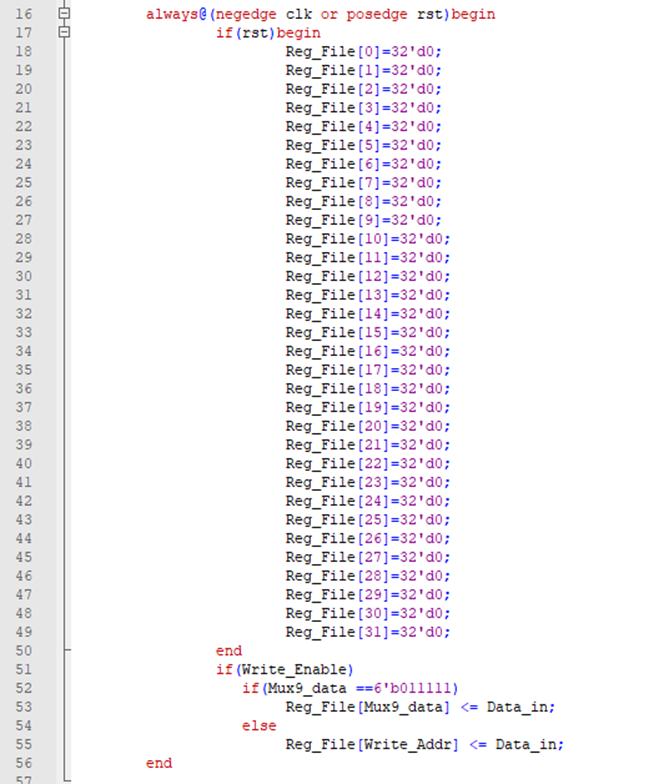
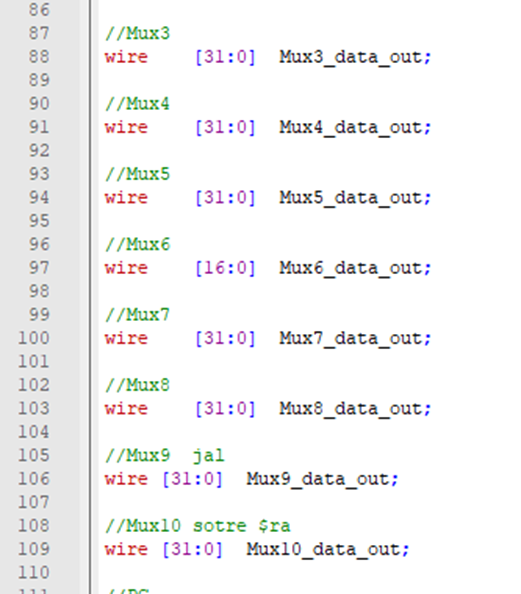
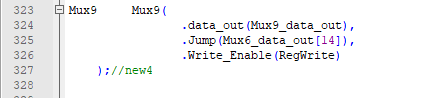
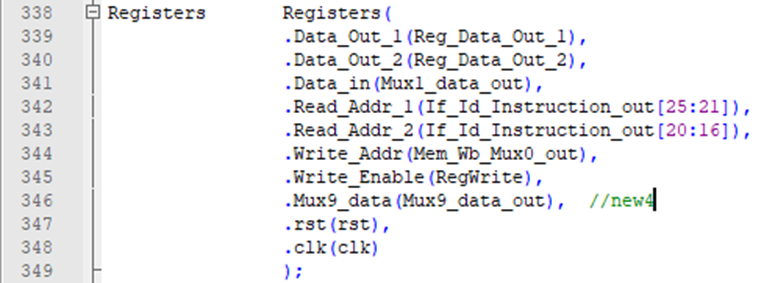
在Control部分，加入JAL opcode 000011，並建立一個屬於JAL的Control (圖26)。

學長的程式中，已經有設計Jump的程式(圖25)，接著加入一個多工器，其功用是當程式觸發JAL，就會回傳31這個值，其目的是為了能讓JAL可以將下個指令的位子放到R31暫存器內(圖27)。

 將31的值傳出後，要修改Register的程式，需多一個判別式，加入一個腳位(上述描述的多工器output)input到Register，當此值為31時，將寫入的位子變為31(即存在R31內)。在將PC+4的值寫入即可完成JAL (圖28、圖29)。

(圖25): 架構圖

學長架構圖有，但實際程式未加mux9

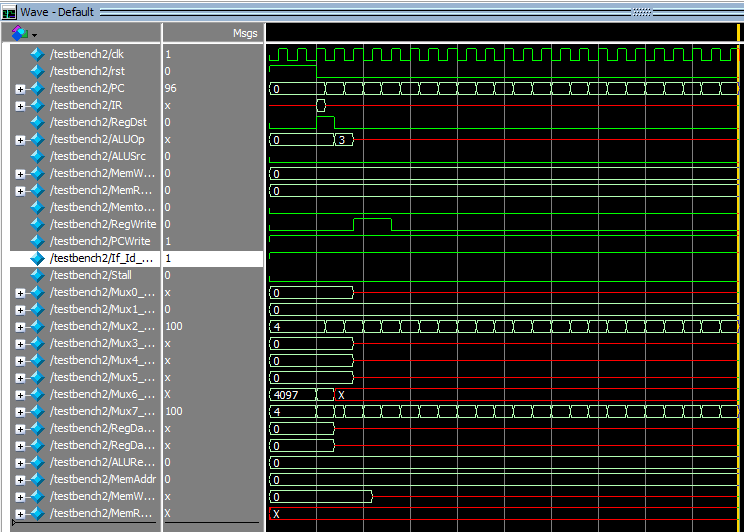


(圖28): Register

(圖29): MISPR2000

(圖27): Mux9

(圖26): Control



(圖30): test\_bench

**五、心得**

　　 經歷一個月的分析，我對MIPSR2000更加熟悉，也越來越看的懂光耀學長的寫法了，在修改指令的時候需要考慮十分周到，有時候跑出來的error真的很多，必須很有耐心的一條一條檢查。如果有些程式忘了加，雖然程式會過編譯，但就是不會跑正確的東西出來，這過程發生了應該有二三十次吧，明明只是簡單的指令，但真正要弄懂它卻是要花上足夠的心力才行，反覆地做模擬、修改，一直不斷的嘗試，才能成功新增指令!