VLSI System Design (Graduate Level)

Fall 2022

HOMEWORK I

REPORT

Must do self-checking before submission:

Compress all files described in the problem into one tar

All SystemVerilog files can be compiled under SoC Lab environment

All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

No any waveform files in deliverables

Student name: 陸啟倫

Student ID: B073011051

**outline**

1.Summary **3**

2.Major problem encountered and resolution**3**

3.Block diagram**4**

4.Modules function introduction **5**

5.Hazard solve mechanism and related waveform**9**

(a)Data hazard**9**

(b)Stall conditions**1**7

(c)Control hazard**20**

6. Waveform of each type Inst. **21**

7.Simulation result **27**

8.Superlint result and explanation**29**

9.Performance **31**

10.Lesson learned **33**

1. **SUMMARY**

透過這次的作業，將我之前在計算機組織所學到的CPU架構、hazard處理等實做了出來。透過實作及結合之前在課堂上的理論，讓我對CPU有更深層的了解。但這次作業中，記憶體和CPU是包在同一個module當中，彼此要溝通或交換資料都不困難，是實際應用上比較不實際的狀況。

這次作業實作到gate-level，需在考慮timing問題的情況下做到function正確及優化performance的問題。

引入了superlint做程式碼的檢查，妥善的運用能有效的檢查設計時造成的疏漏，也能更好的幫助coding style的養成，學習更好的verilog寫法和習慣。

**2.Major problem encountered and resolutions**

**(a)回饋的判斷邏輯不夠完整**

Solve:需考慮得更全面，配合一些simulation case即可完成。

**(b)load及store指令所存取的資料不正確**

Solve:只要新增decoder將資料做修正即可，我新增了load data decode和store data decode來解決此問題，可以根據ALU結果[1:0]和指令function code來判別資料要如何修正。

**(c)管線的flush和stall同時發生導致要執行的指令被沖掉**。

Solve:在stall發生的時候不可以flush，修正後即可解決問題。

**(d)合成後遇到MEM error的問題**

Solve:此問題的原因是因為電路剛重啟時，記憶體吃到了未知的addr.或是讀出未知的指令而傳進管線中。只要在電路重啟時將addr.等相關資訊歸零就可解決。

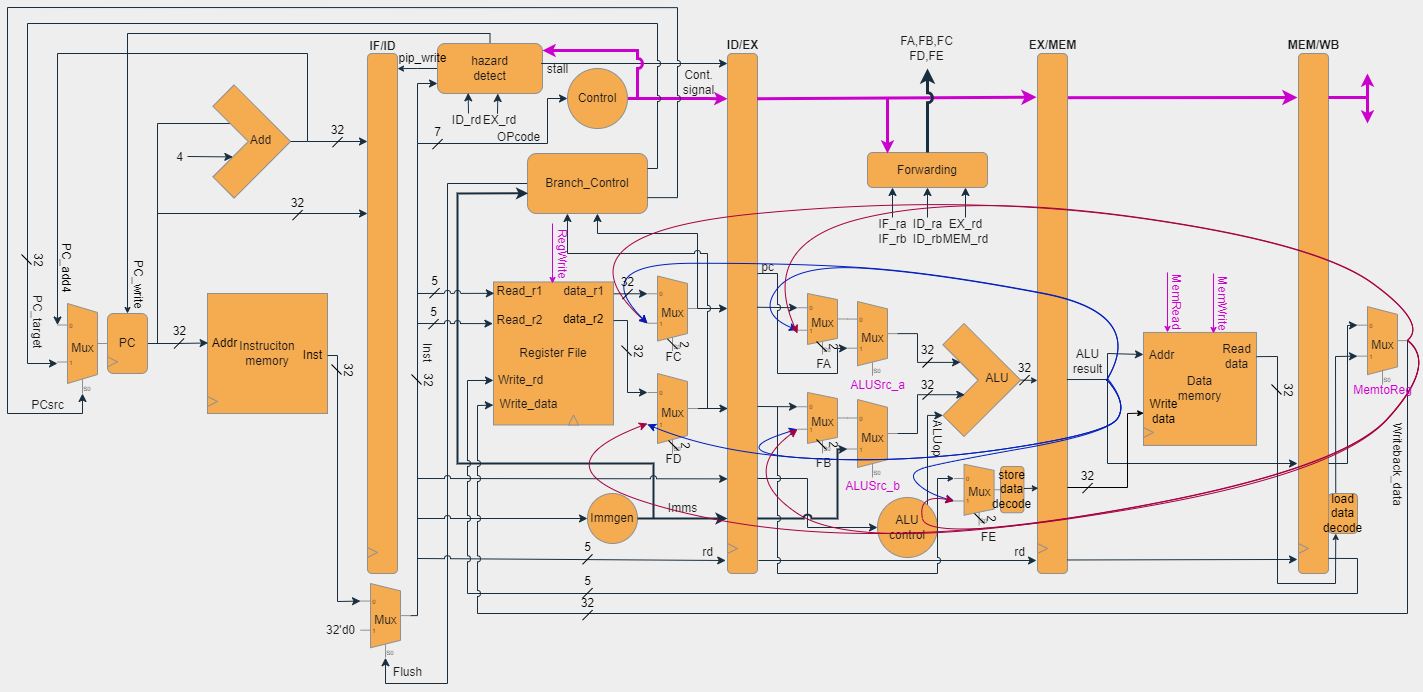
**(e)剛完成設計時，跑模擬遇到dead lock的問題**

Solve:通常遇到dead lock應該是stall訊號的問題，我將有吃stall訊號的相關元件都檢查過後，就有找出dead lock的根本問題，原因出自PC的數值未正確運作。

**(f)合成後遇到violation的問題**

Solve:通常是在critical path delay的問題，我將此路徑上的delay下修後，即可發現在當下的cycle period的violation消失(能往更低的cycle period跑)。

**3.Block Diagram**



在這此五級管線CPU當中，我採取的是branch/jal指令在ID決定的架構，動機出自希望能減少管線因控制危障所flush的指令。

考量到IM1、DM1在讀取資料時須等待一個clk，我選擇將兩個memory的輸出直接拉線到下一級(不再進pipeline)，以配合其他進管線的資料在時脈的狀況。

控制危障由hazard detect及branch control負責，前者輸出stall訊號將往下傳遞的指令變成NOP，以確保此指令不會更改管線內的資料，後者透過flush將誤入管線的指令清除。

資料危障由Forwarding負責，管線共有五條Forwarding path，當資料危障發生時，則會輸出對應的Forwarding signal傳送至對應的MUX做資料取代。

為了處理sw,sh,sb的問題，新增了store data decode。若指令為sb，則將儲存資料取[7:0]，並根據ALU運算結果的bit0、bit1決定byte的位置。舉例來說

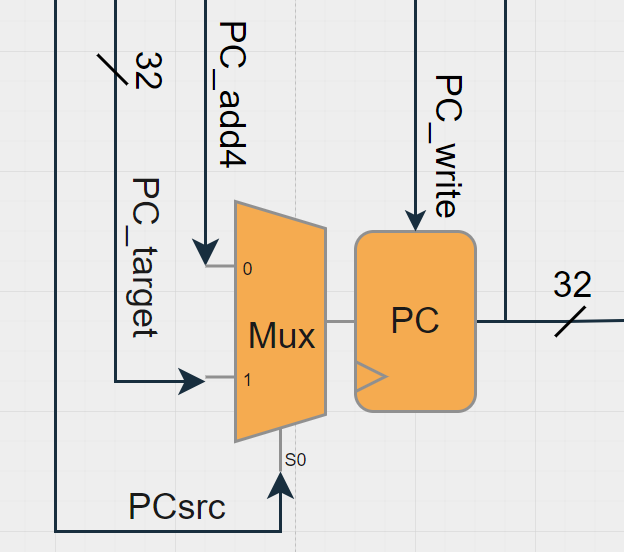
若儲存資料為32’hffff\_ffff，且指令為sb、ALU運算結果[1:0]為2’b11，則將儲存資料設為32’hff00\_0000。(sh同理，sw則不須做資料修正)

為了處理lw,lb,lh,lhu,lbu，新增load data decode。若指令為lbu，則將讀取資料取[7:0]，並做無號數的extension至32bit。舉例來說，若讀取資料為32’hffff\_ffff，且指令為lbu，則將讀取的資料設為32’h0000\_00ff。(其他指令同理，但在擴充上有有號擴充和無數擴充的差別)

**4. Modules Function Introduction**

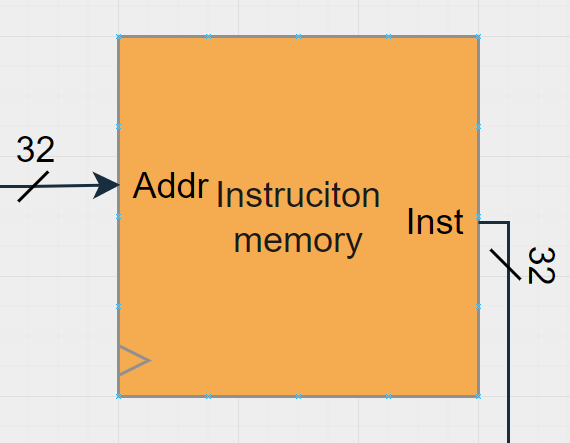
(a)Program counter

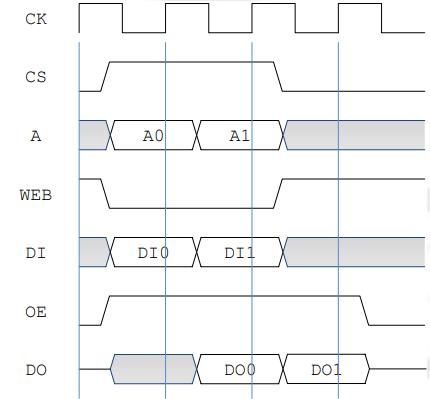
介紹:32bit的特殊目的暫存器，存放下一個要被執行指令的記憶體位置。



(b)Instruction memory

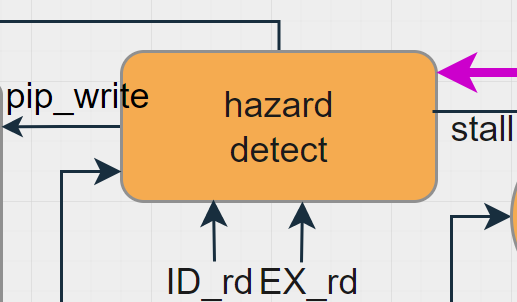
介紹:儲存程式的元件，有32bit的位址輸入及32bit的指令輸出，在給定位址且正緣來到時，需等待一個clk，數值才可被讀出。在控制訊號的設定為恆可被讀取、不可寫入，且位址需以四的倍數給入(4bytes為一個Inst.)。





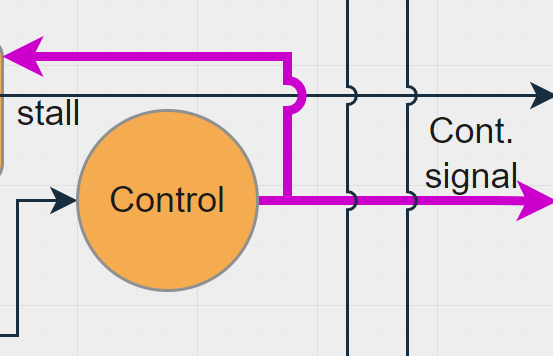
(c)hazard detect

介紹:處理管線中控制危障的元件。當控制危障發生時，會發出stall訊號，將下個傳遞的指令變成NOP指令、控制IF/ID pipeline不可存取數值、且維持PC的數值。



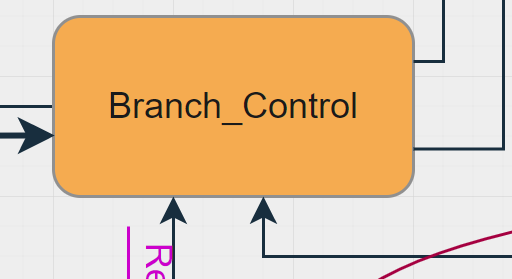
(d)Control

介紹:產生CPU中控制訊號的元件，會根據不同指令的opcode來輸出不同type指令所對應的控制信號。



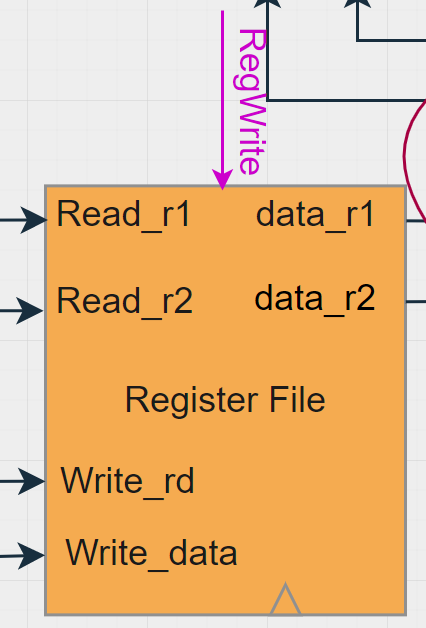
(e)Branch\_control

介紹:處理分支指令及跳躍目的指令的元件，會判別當下指令的跳躍條件是否成立。若成立則會傳遞控制訊號及新目的位置至PC，並產生flush訊號將IF\_stage的指令清除。



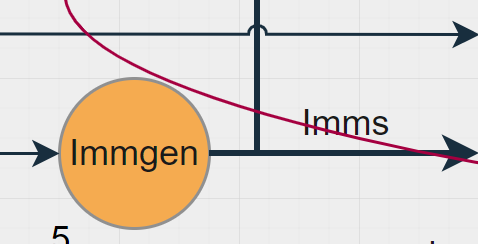
(f)Register file

介紹:一群暫存器的集合，透過暫存器編號，可以讀取或寫入任一暫存器。可同時讀取兩個暫存器及寫入一個暫存器內容。需要控制訊號(RegWrite)來控制暫存器的寫入。



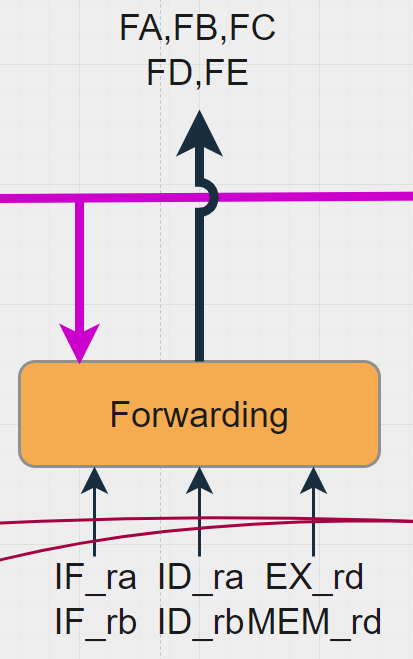
(g)Immgen

介紹:考量到ISA中指令的常數部分並不是非常整齊，我們需要此元件來找到各個type指令中的常數部分，並做sign extension擴充到32bit，供後方元件使用。



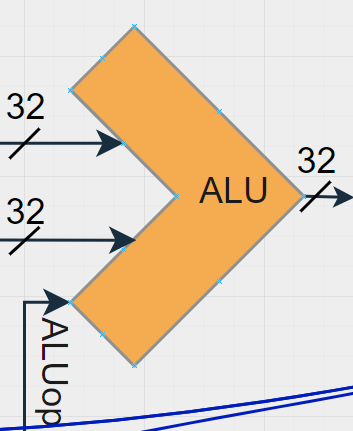
(h) Forwarding

介紹:此元件專門處理資料危障。當資料危障發生時，則會輸出對應的回饋控制訊號(FA,FB,FC,FD,FE)來將資料回饋。



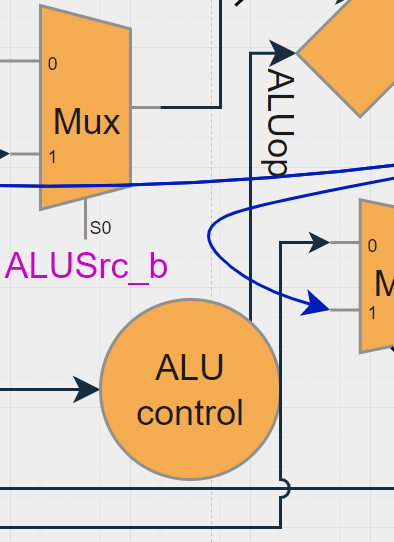
(i)ALU

介紹:執行算數邏輯相關運算的元件，一次只接受兩個32bit的運算元參與運算，並輸出32bit的ALUresult。由ALUop控制當下的運算。



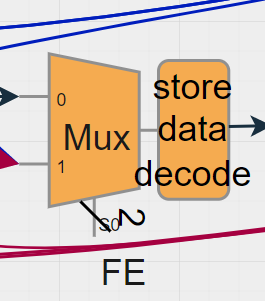
(j)ALU control

介紹: 根據當下指令的opcode及function code來解碼並輸出ALUop，以提供ALU進行對應的運算。



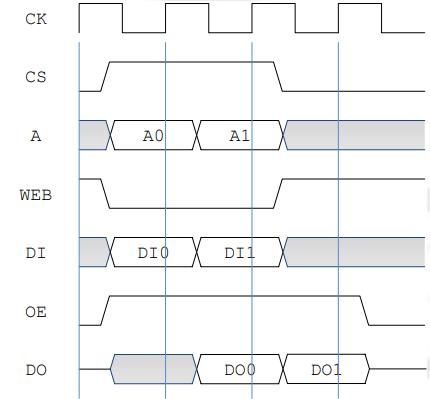
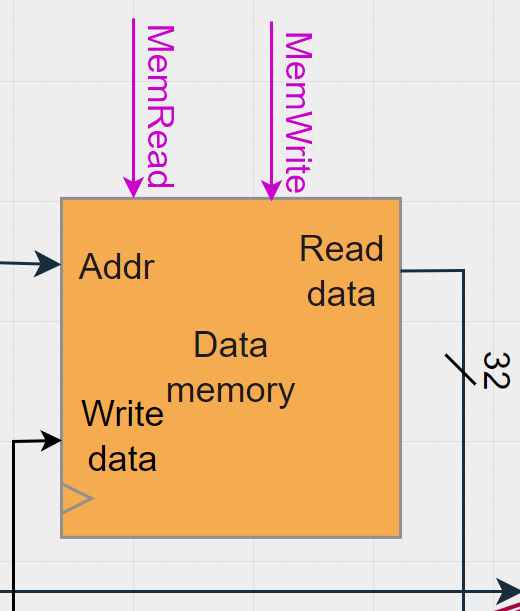
(k)store data docode

介紹:考量到ISA中有sb,sh,sw三種指令，此元件根據對應的指令和ALU運算結果的bit0、bit1，來決定存取的位元數和位置(同時完成sign extension的擴充至32bit)



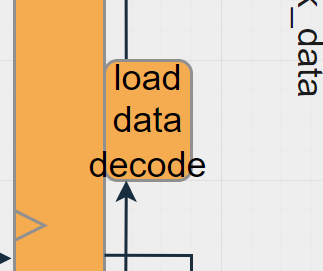
(l)data memory

介紹:提供資料存儲的元件，資料可被寫入或是讀取。在位址被指定後要等待一個clk資料才會被讀出。在控制信號的設定上，只有遇到讀取指令時才可被讀取，與到寫入指令時才可被寫入。

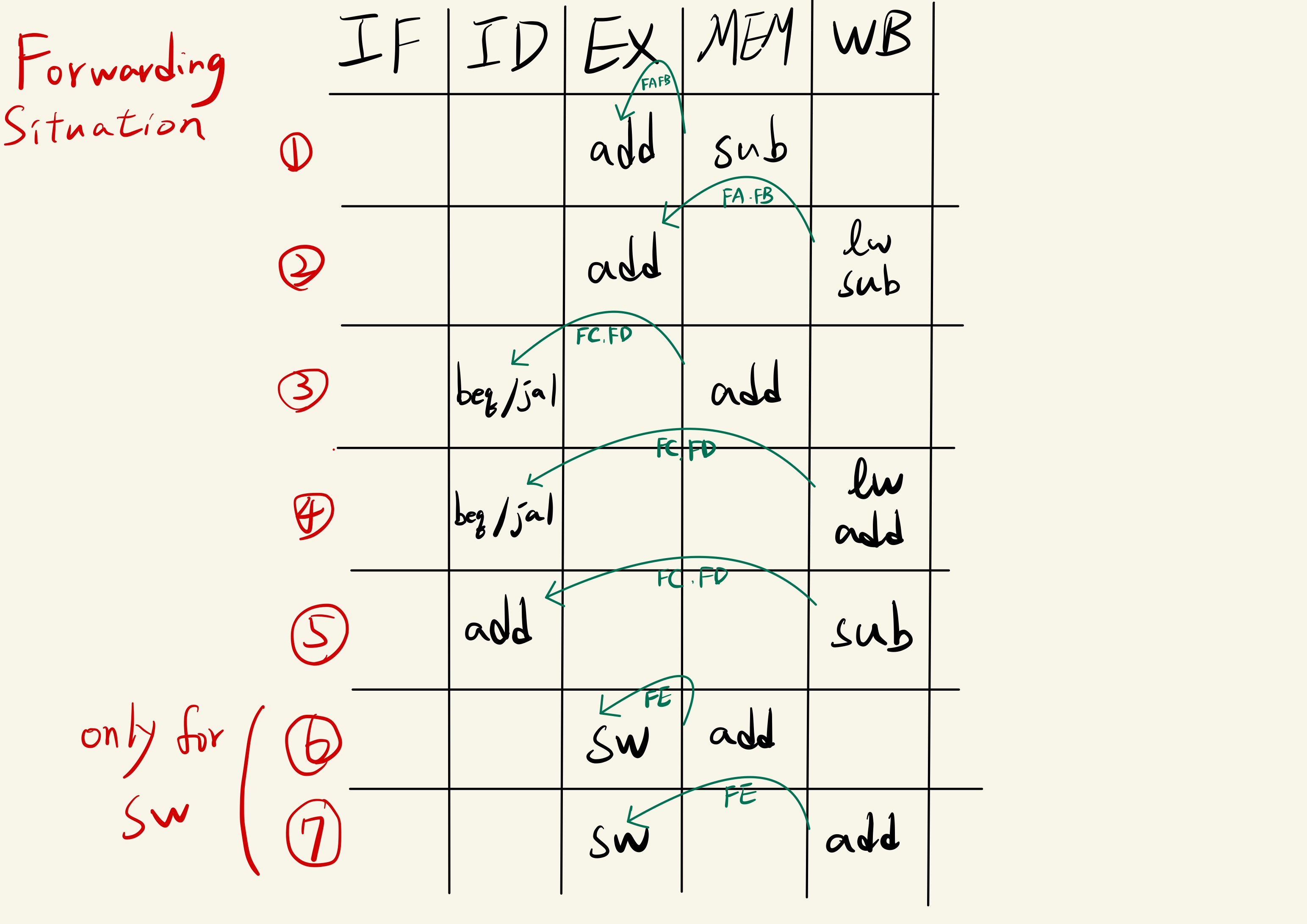
(m)load data store

介紹:ISA中有lw,lb,lh,lhu,lbu五種指令，此元件根據對應的指令解碼要寫入暫存器的數值，並根據指令做有號數或無號數的擴充至32bit。



**5.Hazard solve mechanism and related waveform**

**(a)Data hazard**



**管線回饋的七種情況(五條回饋路線)**

註:上圖1~5為舉例情況，實際包含相同type或類似行為之指令(E.g.add指令包括所有類似行為的Rtype及Itype指令)。(下方依順序解說驗證波型圖。)

FA:當EX\_stage中的來源ra與MEM或WB中的目的相依，則將rd回饋至EX。

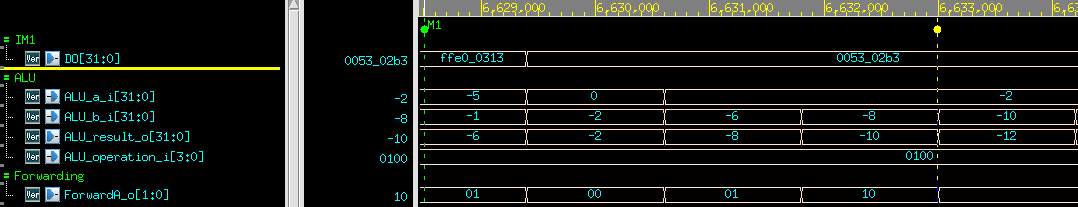
FB:當EX\_stage中的來源rb與MEM或WB中的目的相依，則將rd回饋至EX。

FC:當ID\_stage中的來源ra與MEM或WB中的目的相依，則將rd回饋至ID。

FD:當ID\_stage中的來源rb與MEM或WB中的目的相依，則將rd回饋至ID。

FE:當EX\_stage中sw指令的rb與MEM或WB中的目的相依，則將rd回饋至EX。

**驗證波型圖**

(1) 說明:



考慮執行lui和add指令,且lui的目的暫存器(t1)和add的來源暫存器相依(t1)。在clk1時，lui指令在ID\_stage。在clk2時，lui指令被ALU執行，將-2存入t1，同時add指令在ID\_stage存取需要的來源暫存器數值。在clk3時，add指令被ALU執行，同時ForwardingA將上個指令存於t1暫存器的值回饋至ALU運算。



(2) 一張含有 文字, 監視器, 室內, 螢幕 的圖片

自動產生的描述

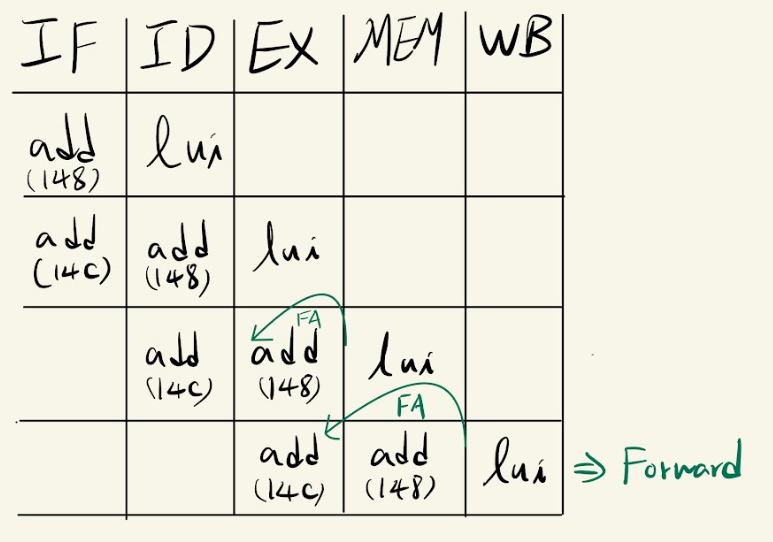


說明:

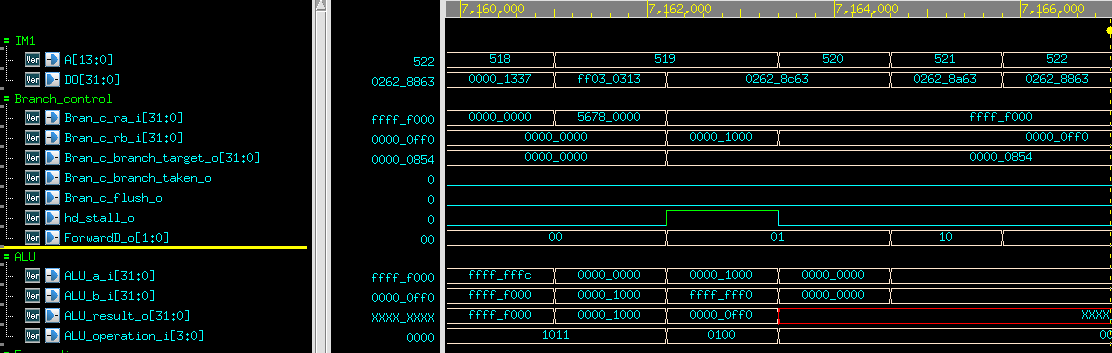
考慮執行lui指令和兩個add指令，lui指令的目的暫存器和add(148)、add(14c)的來源暫存器相依，且add(148)的目的暫存器也和add(14c)的來源相依。在clk1時，lui指令在ID\_stage。在clk2時，lui指令被ALU執行，且add(148)在ID\_stage。在clk3時，add(14c)在ID\_stage，add(148)被ALU執行此時回饋發生，將t1的值拉回EX\_stage運算。在clk4時，add(14c)被ALU  
執行，此時回饋發生，將存取至t1暫存器的數值回饋至ALU運算。

一張含有 文字, 螢幕, 螢幕擷取畫面 的圖片

自動產生的描述



(3)





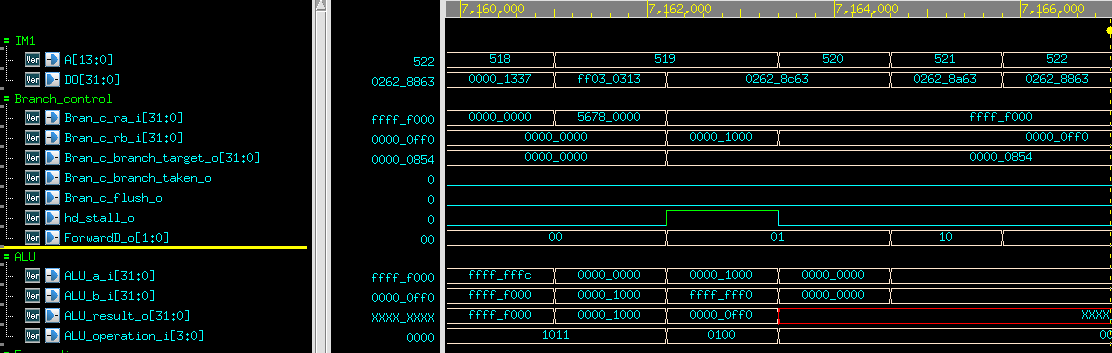
說明:考慮執行下列四個指令，其中addi的目的暫存器t1和beq(81c)的來源暫存器相依。在clk1時，lui指令在ID\_stage。在clk2時，addi指令在ID\_stage且lui指令被ALU執行。在clk3時，beq(81c)在ID\_stage，此時stall發生。在clk4時，beq(81c)依舊在ID\_stage，NOP指令在EX\_stage，而addi指令在MEM\_stage，且回饋發生將t1暫存器的數值拉回ID\_stage做運算，以判別beq指令是否要採取跳躍。

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

(4)

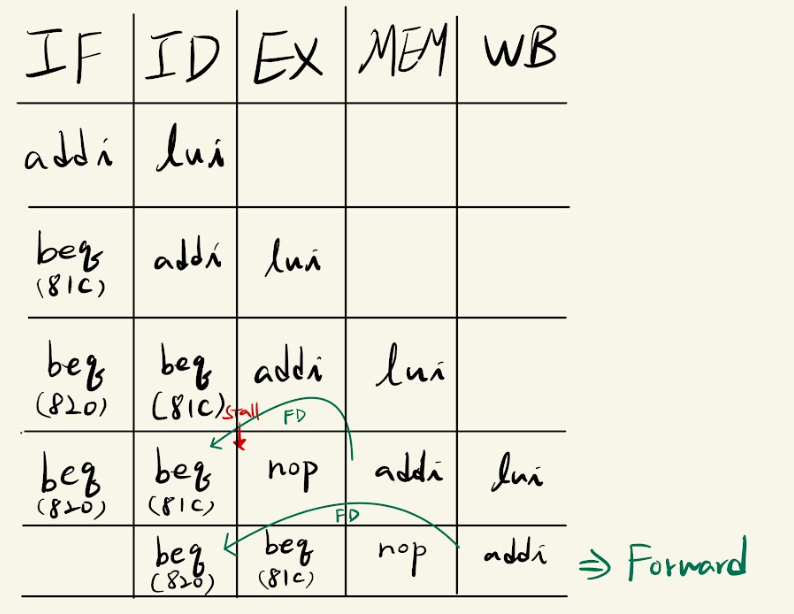




說明:

考慮執行下列四個指令，其中addi的目的暫存器t1和beq(81c)及beq(820)的來源暫存器相依。在clk1時，lui指令在ID\_stage。在clk2時，lui指令被ALU執行，addi指令在ID\_stage。在clk3時，beq(81c)在ID\_stage，addi在EX\_stage，此時stall訊號發生。在clk4時，beq(81c)維持在ID\_stage，NOP指令在EX\_stage，addi指令在MEM\_stage且回饋發生將t1拉回ID\_stage運算。在clk5時，beq(820)在ID\_stage，且回饋發生，將WB\_stage的t1數值拉回ID\_stage運算。

一張含有 文字 的圖片

自動產生的描述

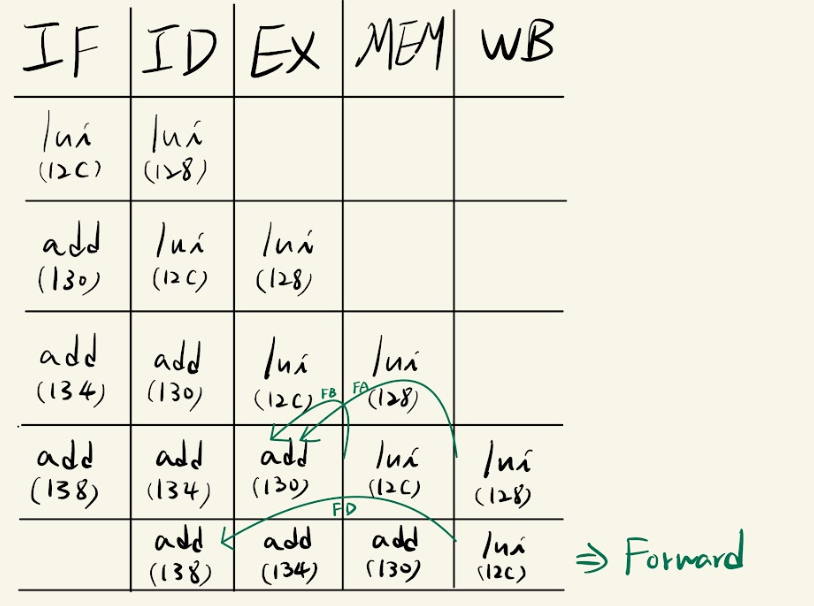
(5) 一張含有 文字, 螢幕, 公寓 的圖片

自動產生的描述

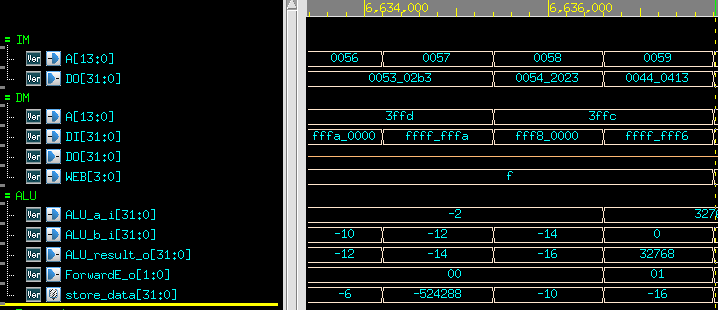


說明:考慮執行下列五個指令，其中lui(128)的目的暫存器t0和add(130)來源暫存器有相依，lui(12c)的目的暫存器t1和add(130)、add(134)、add(138)的來源相依，add(130)的目的暫存器和add(134)的來源暫存器有相依，add(134)的目的暫存器和add(138)的來源暫存器有相依。在clk1時，lui(128)在ID\_stage。在clk2時，lui(12c)在ID\_stage。在clk3時，add(130)在ID\_stage。在clk4時，add(134)在ID\_stage，且回饋發生至add(130)的兩個來源存器。在clk5時，add(138)在ID\_stage，且回饋發生將WB\_stage的t1數值拉至add(138)的來源。

一張含有 文字, 電子用品 的圖片

自動產生的描述

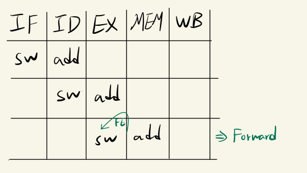
(6)

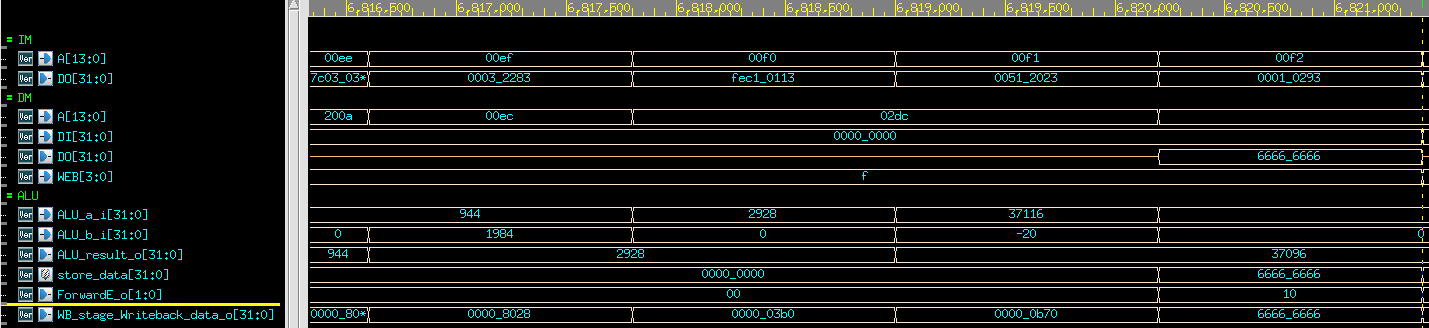




說明:考慮執行下列兩個指令，其中add的目的和sw的來源t0相依。在clk1時，add指令在ID\_stage。在clk2時，add指令被ALU執行且sw指令在ID\_stage。在clk時，sw在EX\_stage且回饋發生將MEM\_stage的t0數值拉回至EX的t0。



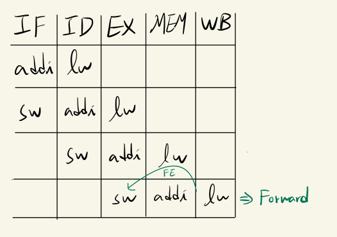


(7) 



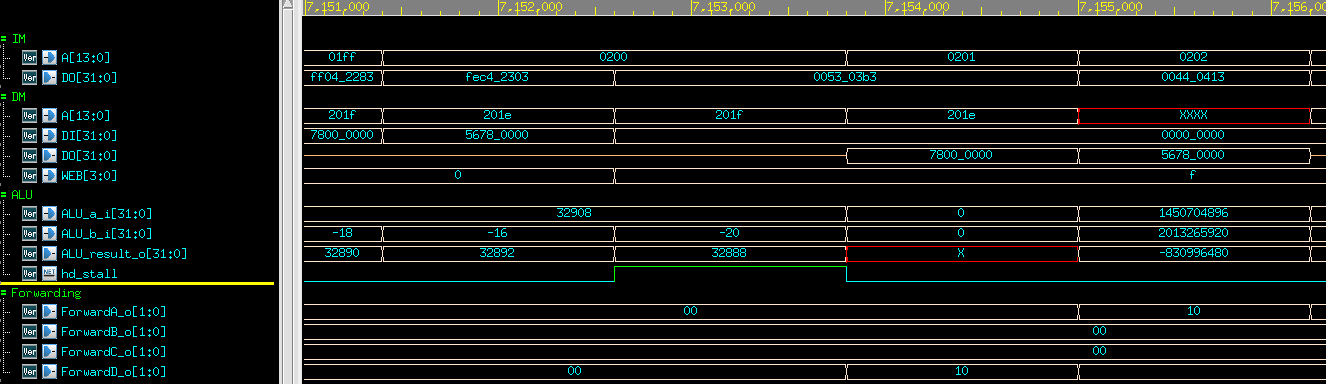
說明:考慮執行下列三條指令，且lw的目的暫存器t0和sw的來源t0相依。在clk1時，lw指令在ID\_stage。在clk2時，addi在ID\_stage、lw在EX\_stage。在clk3時，sw指令在ID\_stage、lw指令輸入addr至DM。在clk4時，lw在EX\_stage，同時回饋發生將WB\_stage的t0數值拉回EX\_stage。





**(b)Stall conditions**

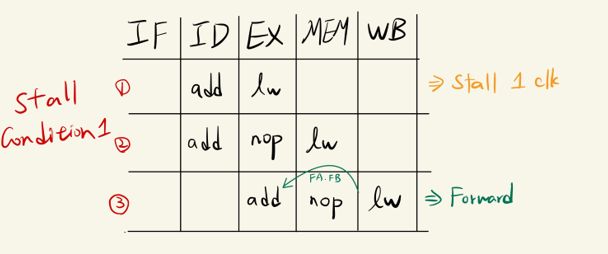
(1)





說明:此狀況也稱load-use data hazard。考慮執行下列兩個指令，且lw的目的暫存器t1和add的來源有相依。在clk1時，lw指令在ID\_stage。在clk2時，add指令在ID\_stage且lw指令在EX\_stage，且stall拉起。在clk3時，add指令維持ID\_stage，NOP指令在EX\_stage，且lw指令在MEM\_stage。在clk4時，add指令在EX\_stage且回饋拉起，將t1的數值拉回EX\_stage。



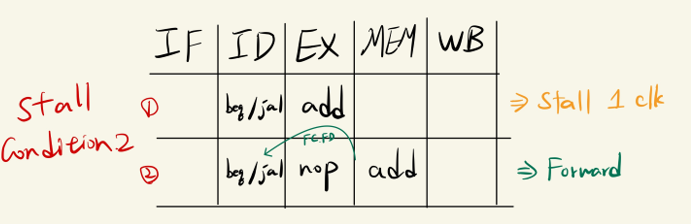


(2) ****

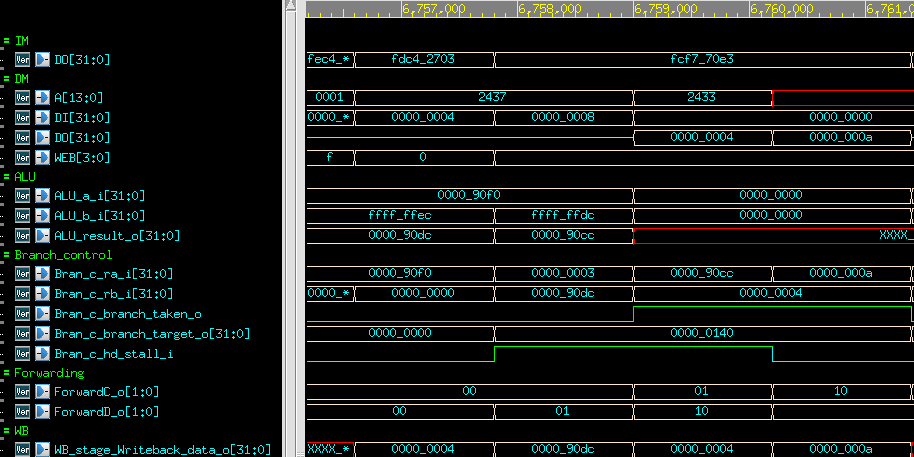


說明:考慮執行下列兩個指令，且addi的目的暫存器t1和jalr的來源t1有相依。在clk1時，addi在ID\_stage。在clk2時，jalr在ID\_stage且addi在EX\_stage，同時stall訊號拉起。在clk3時，jalr維持在ID\_stage、NOP維持在EX\_stage、且addi在MEM\_stage。





(3)

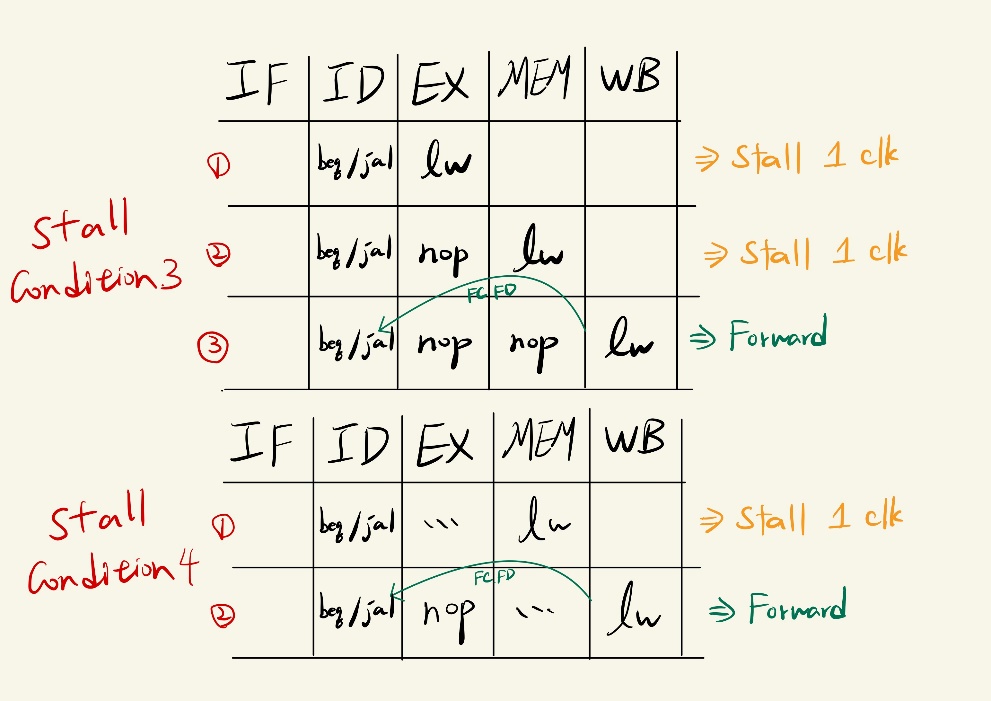




說明: 考慮執行下列兩個指令，且lw的目的暫存器a4和bleu的來源相依。

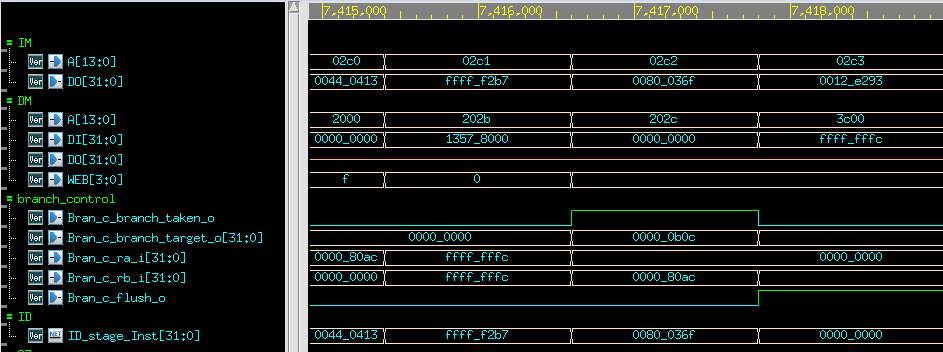
在clk1時，lw指令在ID\_stage。在clk2時，bleu在ID\_stage且lw在EX\_stage，且stall訊號拉起。在clk3時，bleu維持在ID\_stage、NOP在EX\_stage、lw在MEM\_stage、且stall訊號拉起。在clk4時，bleu維持在ID\_stage，EX\_stage和MEM\_stage都是NOP，lw在WB\_stage，此時回饋發生，將WB\_stage中的a4數值拉回ID\_stage運算。





**註:狀況4為狀況3的一部分，在此不贅述**

**(c)Control hazard**

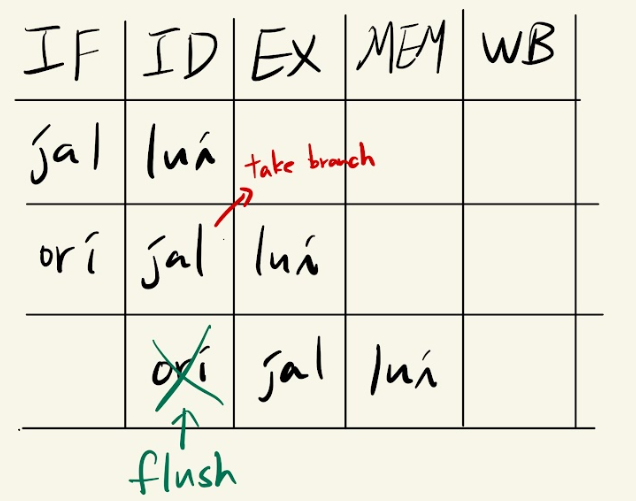
****



說明:考慮執行下列三個指令。在clk1時，lui指令在ID\_stage。在clk2時，jal指令在ID\_stage，branch\_taken訊號會為1(表示要採取跳躍)。在clk3時，flush訊號會拉起，將要輸入至ID\_stage的指令清除為全零，以確保不會影響到後續管線的行為。

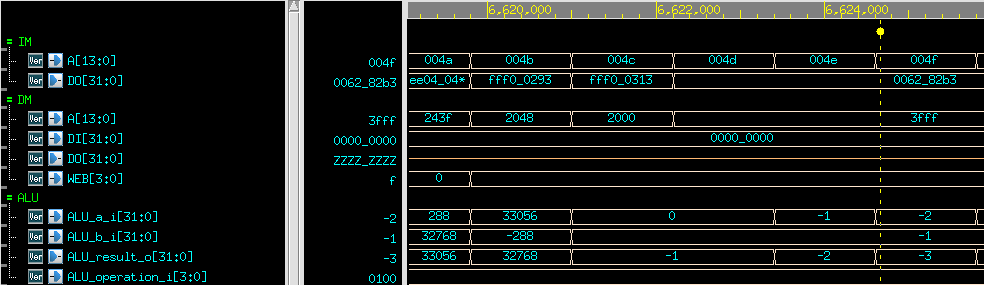
一張含有 文字 的圖片

自動產生的描述



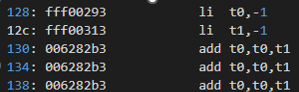
**6.** **Waveform of each type Inst.**

(1)R-type

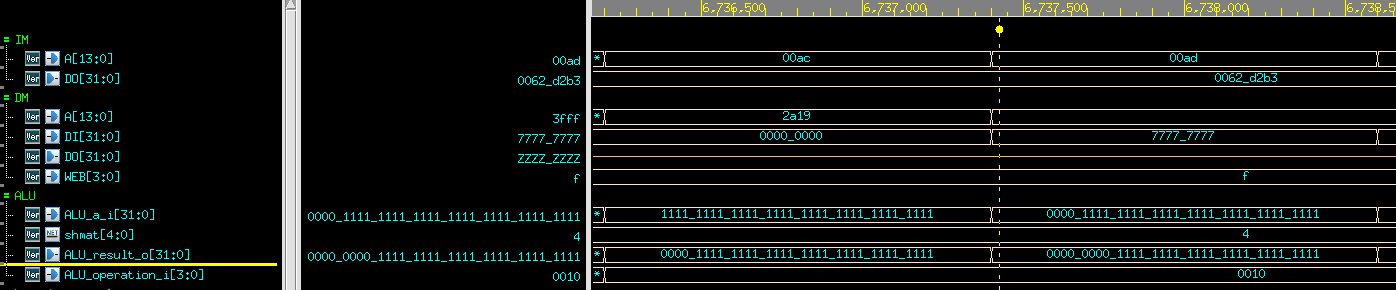
<add> 

ALU\_operation=0100(表ALU執行相加)

ALU\_a\_i=-2且ALU\_a\_i=-1，則ALU\_result=-3



<srl>



一張含有 文字 的圖片

自動產生的描述

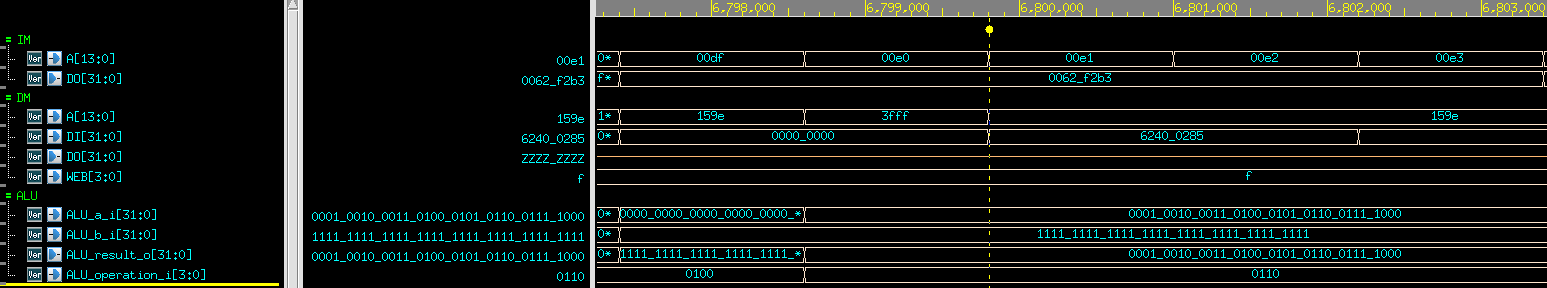
ALU\_operation=0010(表ALU執行srl)

ALU\_a\_i=0000\_1111\_1111\_1111\_1111\_1111\_1111\_1111

shmat=4

ALU\_result=0000\_0000\_1111\_1111\_1111\_1111\_1111\_1111

<and>



ALU\_operation=0110(表ALU執行and)

ALU\_a\_i=0001\_0010\_0011\_0100\_0101\_0110\_0111\_1000

ALU\_b\_i=1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111

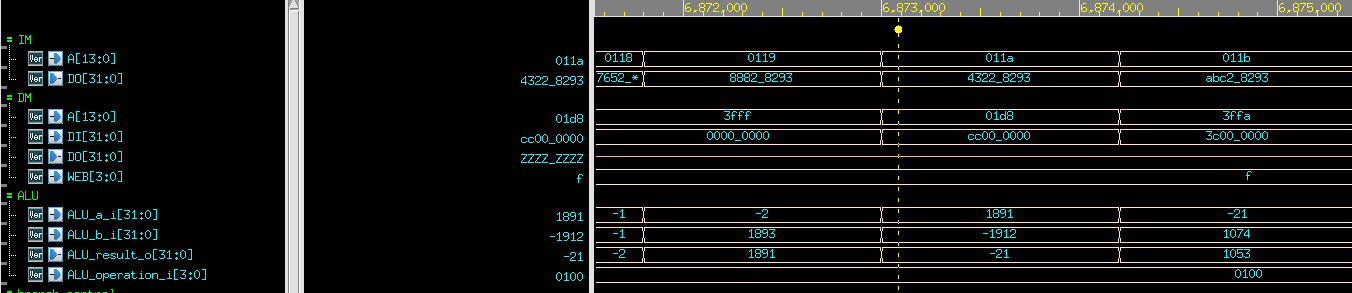
ALU\_result=0001\_0010\_0011\_0100\_0101\_0110\_0111\_1000

一張含有 文字 的圖片

自動產生的描述

(b)I-type

<addi>

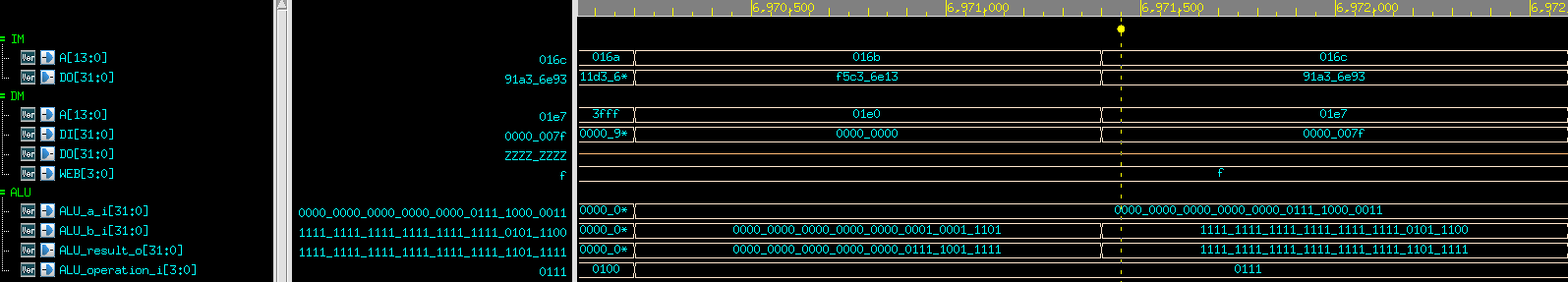




ALU\_operation=0100表ALU執行addi

ALU\_a\_i=1891，ALU\_b\_i=-1912，ALU\_result=-21

<ori>





ALU\_operation=0111(表ALU執行or)

ALU\_a\_i=0000\_0000\_0000\_0000\_0000\_0111\_1000\_0011

ALU\_b\_i=1111\_1111\_1111\_1111\_1111\_1111\_0101\_1100

ALU\_result=1111\_1111\_1111\_1111\_1111\_1111\_1101\_1111

<lw>

一張含有 文字, 監視器, 黑色, 陳列 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

經addi計算後，t1為2928。再經一clk傳遞到DM後，可看到addr為732。因記憶體位置是接線ALU\_result[15:2]，所以剛好差四倍。

(c)S-type

<sw><sh><sb>

一張含有 文字, 監視器, 黑色 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

先由lw(7dc)指令將t5數值讀出，經三個clk後可發現DM的DO為t5的數值。而sw、sb、sh則分別將t5數值存入對應的記憶體位置。其中1234\_5678、0000\_0078、0000\_5678分別對應word、half word、byte。

(d)B-type

<beq>

一張含有 文字, 監視器, 牆, 電視 的圖片

自動產生的描述



Bran\_c\_ra\_i=ffff\_f000、Bran\_c\_rb\_i=ffff\_f000

因ra==rb所以beq指令要跳，可看到Bran\_c\_branch\_taken\_o=1表示條件成立

<bne>

一張含有 文字, 監視器, 牆, 公寓 的圖片

自動產生的描述

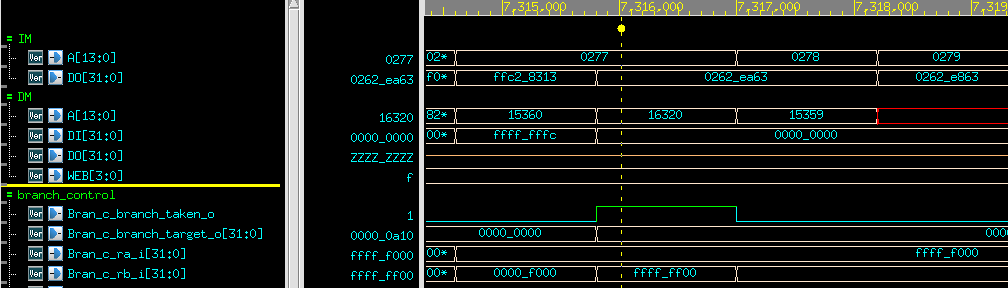
一張含有 文字 的圖片

自動產生的描述

Bran\_c\_ra\_i=ffff\_f000、Bran\_c\_rb\_i=0000\_0ff0

因ra!=rb所以bne指令要跳，可看到Bran\_c\_branch\_taken\_o=1表示條件成立

<bltu>



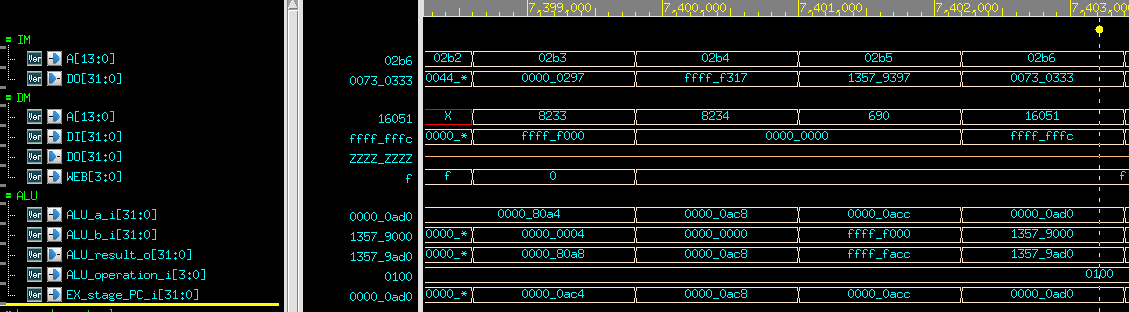


Bran\_c\_ra\_i=ffff\_f000、Bran\_c\_rb\_i=ffff\_ff00

因ra<rb所以bltu指令要跳，可看到Bran\_c\_branch\_taken\_o=1表示條件成立

(e)U-type

<auipc>



一張含有 文字 的圖片

自動產生的描述

Auipc的行為是rd=PC+imm<<12。可以看到上圖的ALU\_a\_i為EX\_stage中的PC數值，ALU\_b\_i為imm<<12，兩者透過ALU相加，得到新的rd數值。

<lui>

一張含有 文字, 監視器, 牆, 電視 的圖片

自動產生的描述

Lui的行為是rd=imm<<12。可以看到上圖的ALU\_b\_i為imm<<12，而新的rd數值即為ALU\_result\_o(即ALU\_b\_i)。

一張含有 文字 的圖片

自動產生的描述

(f)J-type

<jal>

一張含有 文字, 監視器, 螢幕, 陳列 的圖片

自動產生的描述



由上圖可看到當jal在ID\_stage時，branch\_taken\_o=1，且branch\_target為0000\_0b0c。

(g)CSR

<RDINSRTET>

一張含有 文字, 監視器, 螢幕, 螢幕擷取畫面 的圖片

自動產生的描述

由上圖可看到RDINSRTET將instret\_i[31:0]存為ALU\_result，以將數值寫入t1。



<RDCYCLE >

一張含有 文字, 監視器, 螢幕, 螢幕擷取畫面 的圖片

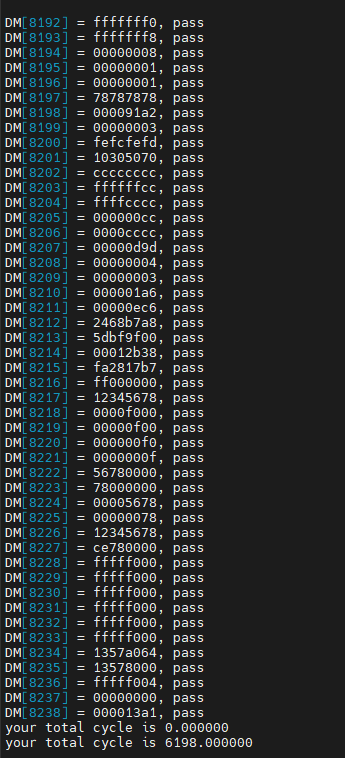
自動產生的描述



由上圖可看到RDCLCYE將cycle\_i[31:0]存為ALU\_result，以將數值寫入t1。

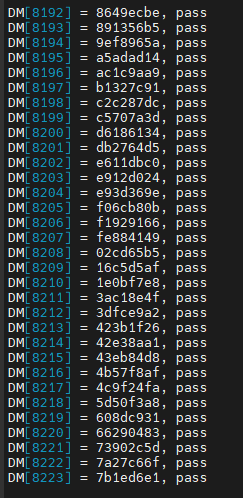
**7.Simulation result**

(a)Prog0

**** **一張含有 文字, 電子用品 的圖片

自動產生的描述**

(b)Prog1

一張含有 文字 的圖片

自動產生的描述

(c)Prog2

一張含有 文字 的圖片

自動產生的描述

(d)Prog3

一張含有 文字 的圖片

自動產生的描述

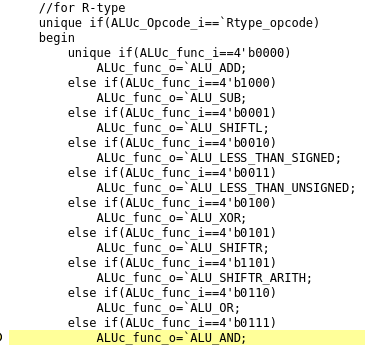
(e)Prog4

**一張含有 文字 的圖片

自動產生的描述**

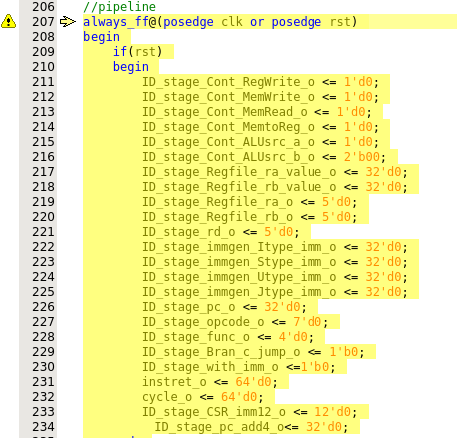
**8.Superlint result and explanation**

(a)most freq. warning:



Solve:將if else 寫滿，即可消除此warning。

(b)most freq. warning: 



Solve:將部分的register拉去另一個always block即可解決。

(c)most freq. warning:



一張含有 文字 的圖片

自動產生的描述

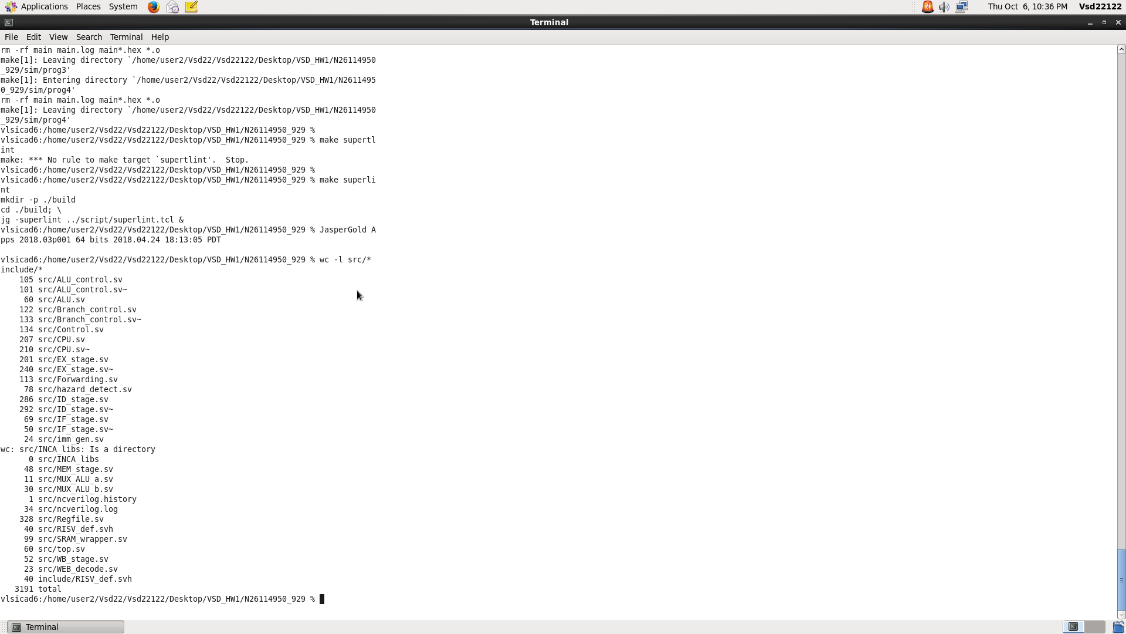
Solve:因之前debug而忘記拿掉的接線，拿掉即可解決此warning。

Final result:

一張含有 文字 的圖片

自動產生的描述

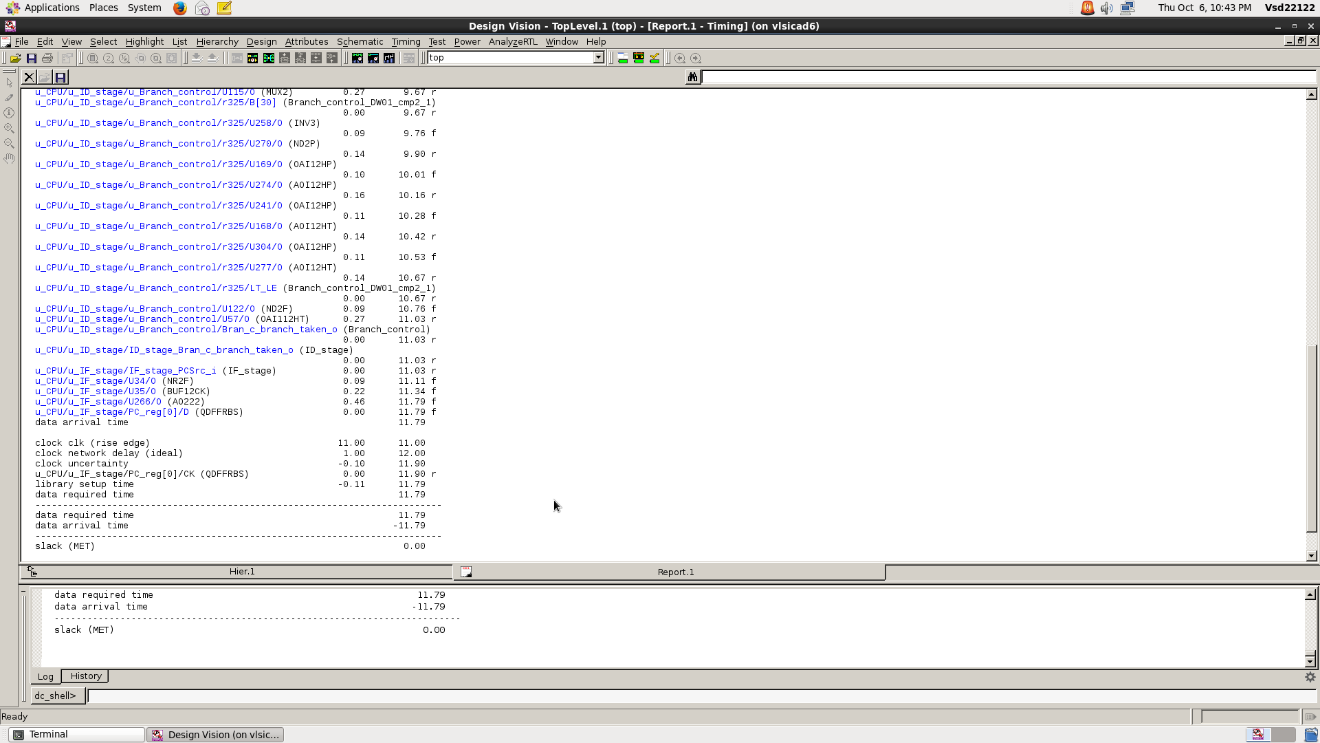
Total line of my RTL codes:



**9.Performance**

(a)report timing

一張含有 文字 的圖片

自動產生的描述

(b)report area一張含有 文字 的圖片

自動產生的描述

(c)report power一張含有 文字 的圖片

自動產生的描述

**10.Lesson learned**

**(a)計算機組織中的CPU架構、危障處理，與實務上的差別**

在實作過程中，可以發現課本上的架構圖只有大方向，實際執行上有更多的元件需要考慮還有時脈及延遲的問題，課本是比較理想的情況。在危障處理上，課本是給了幾個經典的例子，但在實務面上判斷回饋的邏輯考慮得更周到，才有辦法完成正確的回饋。舉例來說，課本中的回饋邏輯並未考慮到addi指令的常數，若未在回饋邏輯考慮足夠清楚，則有可能將要相加的常數回饋掉，導致結果錯誤。

**(b)RISC-V與MIPS的差異之處**

之前在計算機組織是以MIPS來介紹CPU，這次實作的是RISC-V。可以發現在ISA的設定上，常數的bit數、擺放位置、以及指令的分類，MIPS和RISV-V還是有一些不同。但這些不同並不影響實作，大方向及基本知識都是相通的，甚至還能看到不同架構在設定ISA時的優缺點。舉例來說，MIPS在計算跳躍目的位置時，需將暫存器內容再乘上4才是目的，但RSIC-V中，站存器中的數值就直接是指令的數目，無須再乘4，可省下一些運算。

**(c)coding style的培養**

這次功課算是我目前做過要接最多的線的設計，所以在I/O的取名上就非常的重要，我個人習慣在取名後加上\_i或\_o，這樣可以很好得幫助我了解這根線是屬於輸入或是輸出。在連接module上也能比較清楚。除此之外寫註解也是很重要的習慣，我喜歡在每個always block上寫出其行為，能很好的提升我回顧程式碼的速度，尤其在邏輯較複雜的元件上更有效。

(d)**supertlint的使用**

這是我第一次使用到這個工具，他能有效的檢查程式碼因設計疏失而造成的疏漏，並回報給設計者。這些警告可能不會對電路的功能性有影響，但若能修正可以學習到更多好的寫法及習慣，對於日後設計電路也很有幫助。