VLSI System Design (Graduate Level)

Fall 2021

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: 李秉軒

Student ID: N26100595

* Design Explanation

本次作業設計的是一個符合RISC-V ISA 的5-stages pipelined CPU 電路，並支援Forwarding 和 Hazard Detection(Load Use)功能。本CPU設計採用RV32I指令集，內有32個32bits Register，以及32bit program counter，並支援37條RISC-V instruction。以下將分別介紹不同stage的功能和Hazard處理機制。

* IF(Instruction Fetch) stage

IF級負責向Instruction Memory要求存取Instruction，並透過Program Counter (PC)紀錄目前指令執行的位置。PC會在每個Cycle正緣更新輸出值，輸出值共有兩種：上個Cycle的輸出值+4, 以及跳轉指令(Jump/Branch)的目標地址。

由於系統使用Byte Address，因此下一條指令的位址會是目前的位址+4。至於跳轉指令的目標地址則是由EXE級的加法器運算得出。而PC的輸出值應在兩者中擇一，輸出值為何是由EXE級的jumpBranch訊號決定。

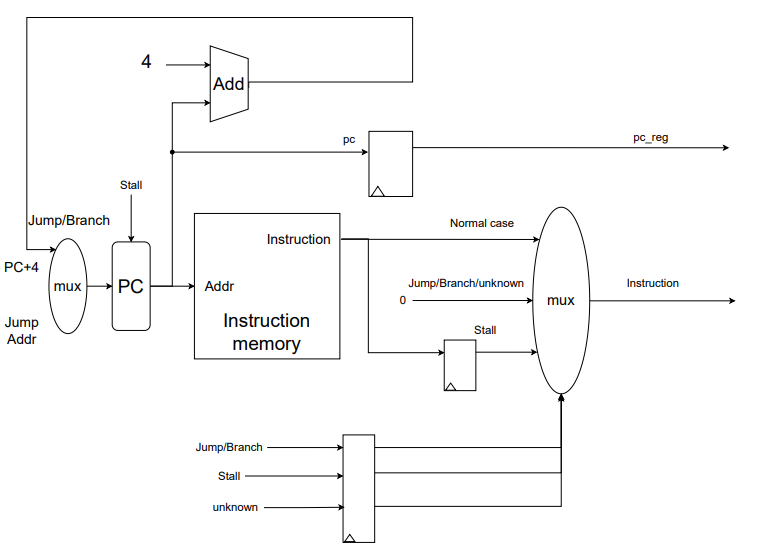
此外，pc有接收一條Stall訊號，是在ID stage偵測到Load Use hazard時發出，若此訊號為High，則pc在下個cycle必須輸出一樣的值。

可以注意到，圖中有一個名為unknown的register，這是為了處理在reset後第一個cycle，instruction memory會輸出unknown的值，所以必須使用一個只在reset會被設為1的register作為判斷依據，將輸出的unknown遮蔽。然而這樣的設計並不robust，在下次作業中應該要改為控制chip select的方法。

由於Instruction會在下一個cycle正緣才輸出值，因此輸出的訊號不用經過register，然而Instruction才傳至下一級前必須先處理，主要有三種情況：

1. 一般情況，直接輸出至下一級。
2. 跳轉指令(Jump/Branch)，或unknown，需輸出0。
3. 檢查到Hazard，需Stall一個Cycle，此時需輸出上一個cycle指令。

以上情況會分別在介紹跳轉指令機制及Hazard Detect機制時詳細介紹，需注意的是，由於Instruction memory晚一個cycle輸出的行為，相關的判斷訊號都必須透過register儲存，才能在正確的Cycle控制傳到ID stage的指令。



(圖) IF stage

* ID(Instruction Decode) stage

ID級負責進行指令的解碼，有Control Unit、Immediate Handler、Register File、Hazard Detection Unit等Component，以下簡單介紹各Component功能：

Control Unit:

負責整個CPU各模組的控制，共有以下控制指令：

|  |  |
| --- | --- |
| 指令 | 功能 |
| JAL/JALR | 判斷是否為此兩種指令，差異在於EXE級運算Target Address的Source不同。 |
| Branch | 是否為B-type指令，用於判斷是否要跳轉。 |
| MemRead/  MemWrite | 判斷存取Data memory，之所以需要兩個訊號是為了Load Use hazard的判斷。 |
| ALUOp | ALU運算的方式，EXE stage會詳細介紹 |
| ALUSrc1,2 | 傳入ALU的source，EXE stage會詳細介紹。 |
| RegWrite | 是否要將資料寫入Register File。 |
| DataWidth | Load/Store指令的存取寬度。 |
| ImmOp | Immediate值的運算方式。 |

Immediate Handler:

由於不同type指令對於Immediate值有不同解碼方式，固集中在一個component處理。

Register File:

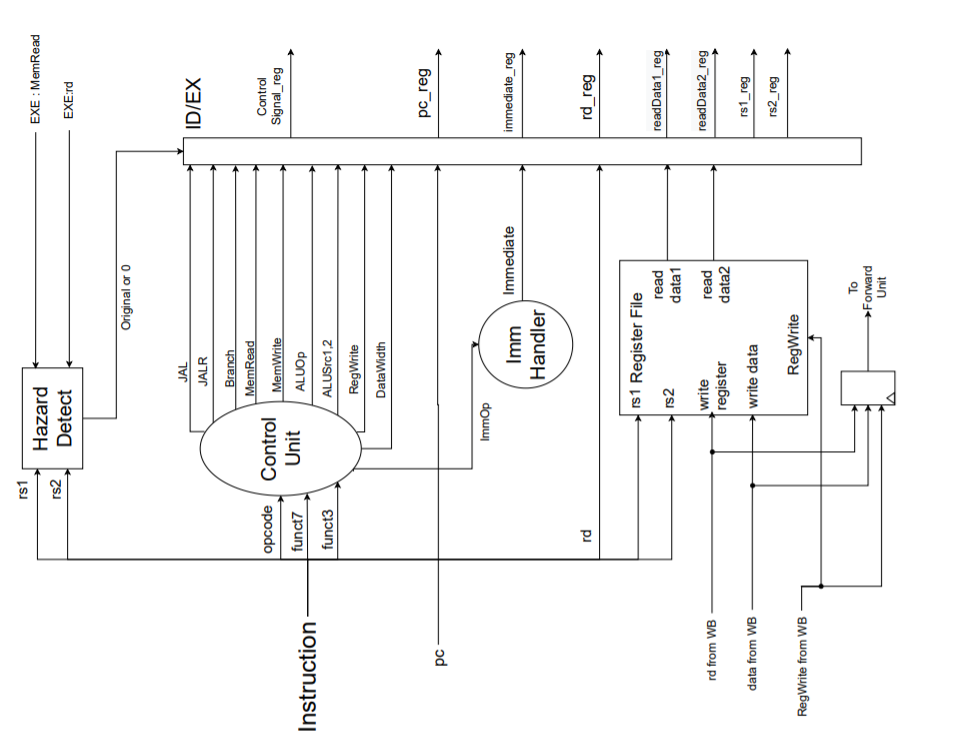
採用正緣觸發寫入，組合電路讀出的Register File，其中x0固定為0。放在ID級是為了讀出值供EXE stage運算。

Hazard Detection:

由於本設計使用Forwarding機制，Hazard Detect僅處理Load Use hazard。判斷依據為：

Stall = (EXE.MemRead) || (EXE.rd == rs1 || EXE.rd == rs2)

若偵測到Load Use hazard，需stall一個cycle，實際的行為是讓進入到ID級的指令重複一次，並往EXE級輸入bubble(全0指令)。為了要讓ID級重複收到一樣的指令，必須使用一個register存取上個cycle的指令。



(圖) ID stage

* EXE(Execute) stage

EXE級主要負責進行各種運算，包括ALU的運算以及跳轉指令的Target Address的運算，以及是否跳轉的判斷。此外，為了解決Hazard，也在此級實作了Forwarding機制。

ALU

ALU主要負責各種算數運算，以及Branch指令的比較結果。將算術運算的指令和Branch比較指令合併在同一條訊號(ALUOp)的結果，可以減少接線的數量，但也導致Decode的時間成本增加，是可以考慮改善的部分。輸出部分則將算術運算的結果和Branch比較結果分開。

其中ALU一條比較特別的指令，是JAL指令時讓pc+4，這樣的作法可以讓前面的stage只傳pc值，而不用pc和pc+4都傳。

Jump/ Branch

跳轉指令的目標地址和是否跳轉，都是在EXE stage進行。由於Branch指令必須使用ALU得出比較結果，因此必須有額外的加法器運算目標地址。由於JALR和JAL/Branch的運算的Source不同，需要使用一個Mux區分加法器的input source。

當確定指令要跳轉，必須把Target Address傳至IF stage給pc，並傳送一個jumpBranch訊號到IF和ID，IF和ID收到後，必須在下個cycle輸出Nop訊號到下一級。也就是說，當發生Jump/Branch時，指令會stall一個cycle，這是因為pc要下一個cycle才會給Instruction memory要跳轉的位址，再下一個cycle， IM才會給出我們要的指令。

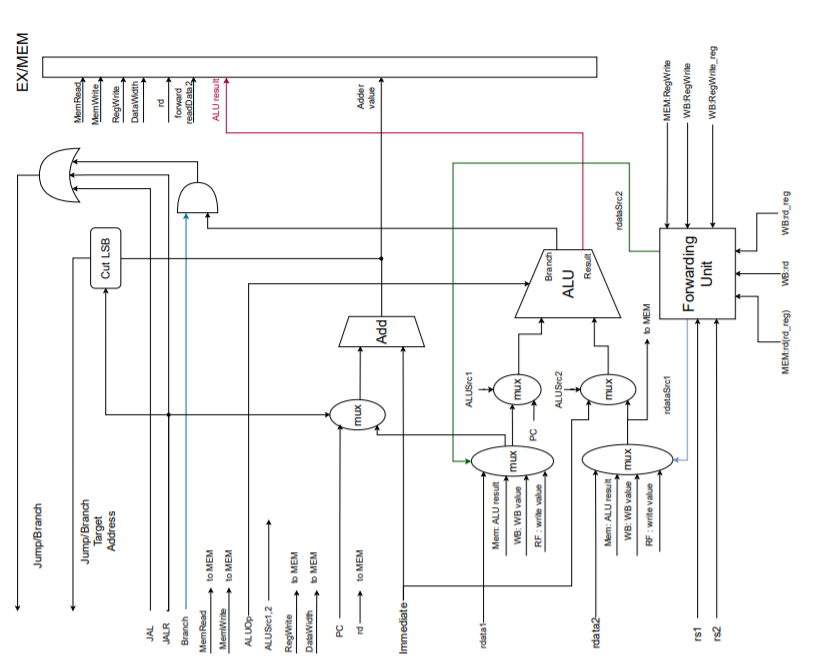
因此，下一個cycle時IM給出的指令必須要被遮蔽掉(給0)，也就是IF級所做的處理。

Forward

為了解決Read After Write(RAW) hazard問題，可以使用forwarding把前幾個cycle算出來的結果回傳給ALU。

當目前指令要運算的Operand，是上一條指令要寫回的Register(rd)，就需要把MEM stage的ALU 運算結果(aluResult)傳回給ALU input。而如果是上兩條指令的情況，則需要從WB stage傳回。在本CPU的設計中，由於Register File是正緣寫入，這樣會導致儘管運算結果已經寫入Register File，但ALU仍然會收到錯誤的值。因此，必須再多往前檢查一條指令，此外，必須再多增加Register儲存寫入Register File的值。

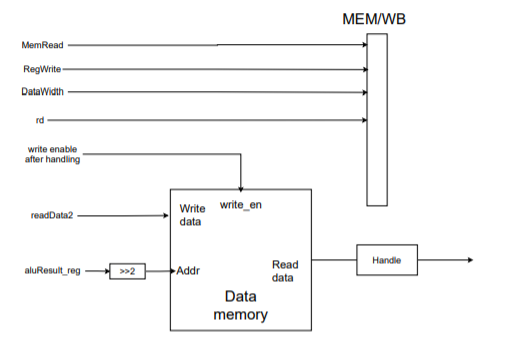
在實作上，是透過檢查前面指令的RegWrite訊號以及rd訊號做判斷，而當遇到前一條和前兩條指令檢查都符合的情況，則以最接近的結果為主。最後一點，由於某些指令會透過將值寫入x0來避免寫入Register File，因此在判斷時，若rd為x0，則不用forward，否則會forward到錯誤的值。



(圖) EXE stage

* MEM(Memory) stage

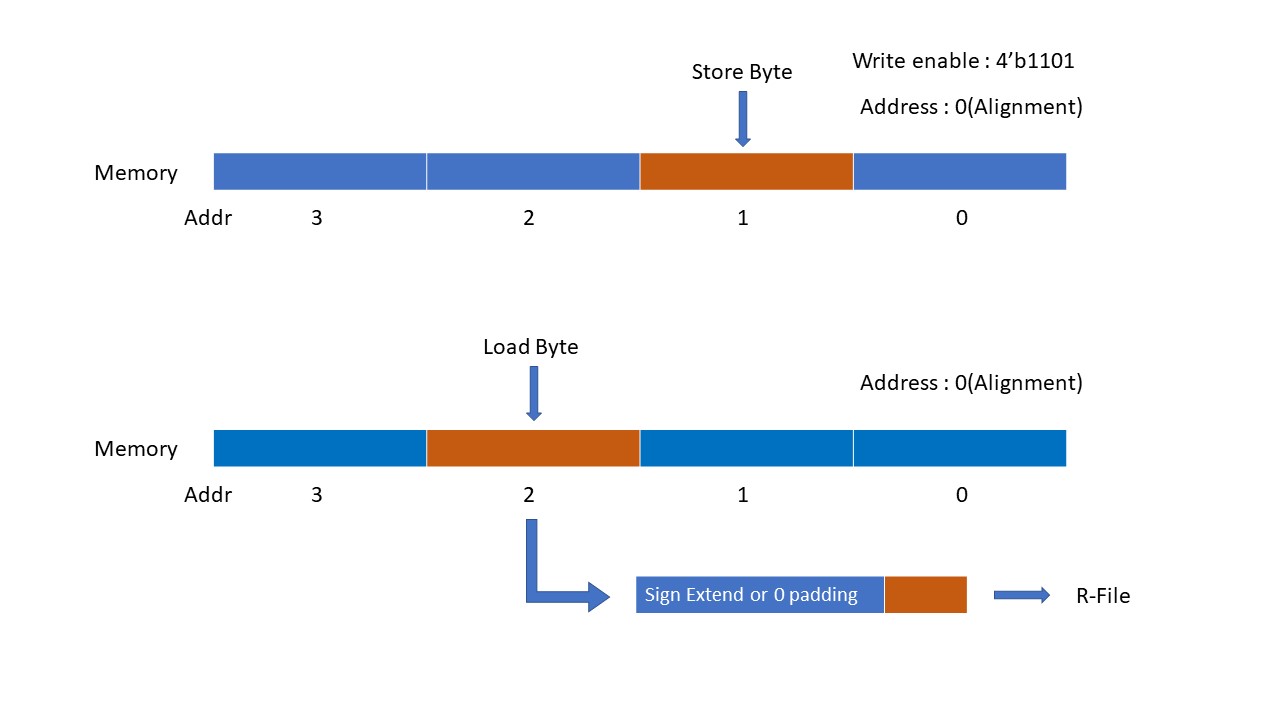
MEM級負責將ALU運算的結果進行Alignment的運算後，傳給Data Memory作為欲存取的位址。此外，也必須透過DataWidth訊號及Memory存取地址的最右2bit判斷欲存取的資料寬度，將write enable訊號傳給Data Memory。



(圖) MEM stage

* WB(Write Back) stage

WB級主要將資料寫回Register File，在寫回Register File前，需先透過MemRead訊號判斷是要寫回ALU的運算結果或是Memory的存取資料。而同時，Memory的存取資料會透過DataWidth訊號及Memory存取地址的最右2bit做處理，並最終寫回Register File，正式完成一個指令。

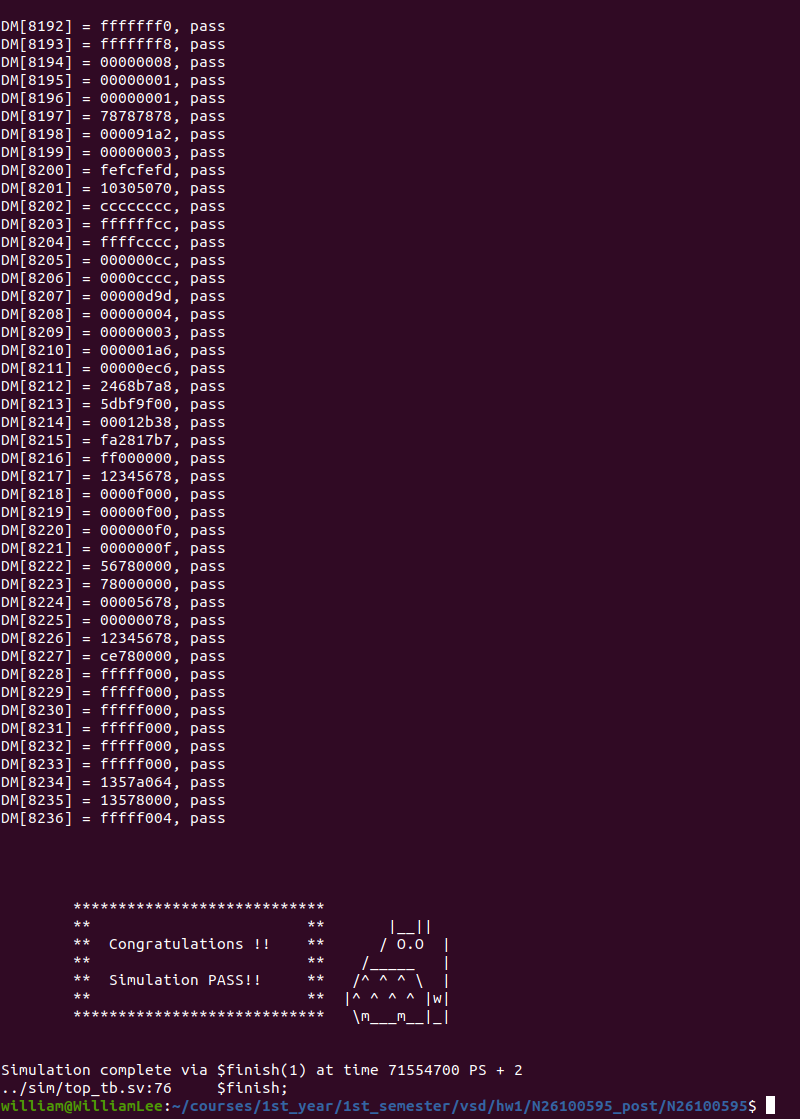


(圖) Load/Store示意圖

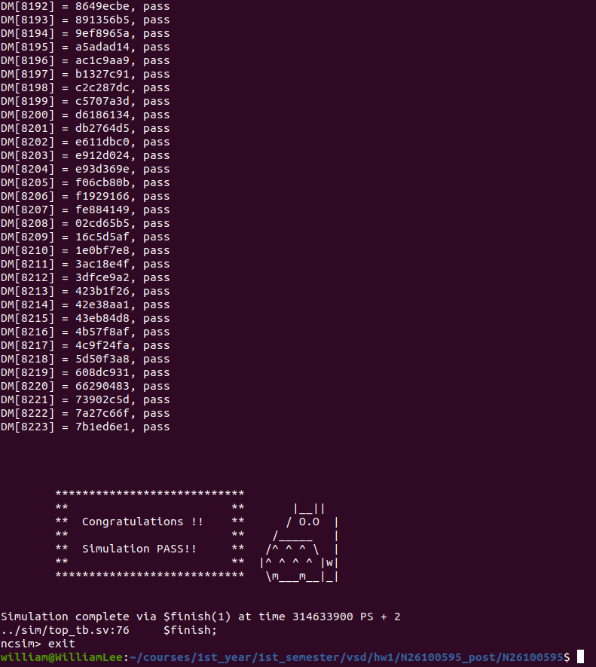
* Simulation Result

- Terminal

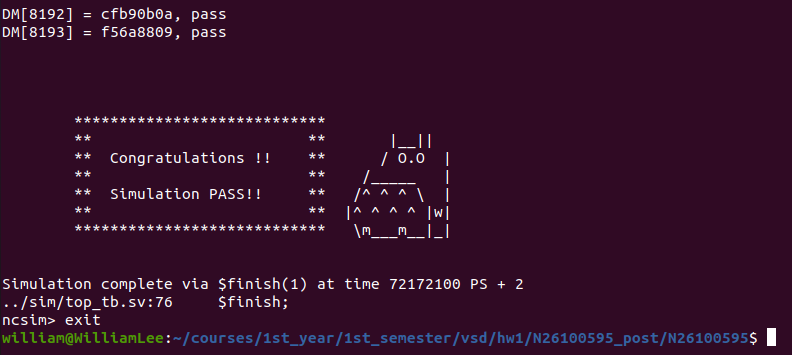
Clock Period: **9.8ns**



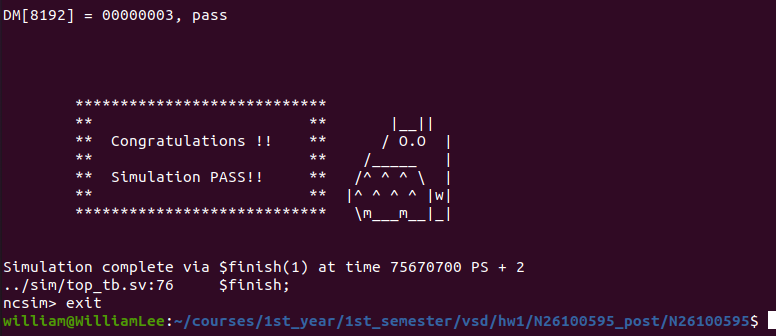
(圖) Program 1 : 71554700 ps



(圖) Program 2 : 314633900 ps



(圖) Program 3 : 72172100 ps

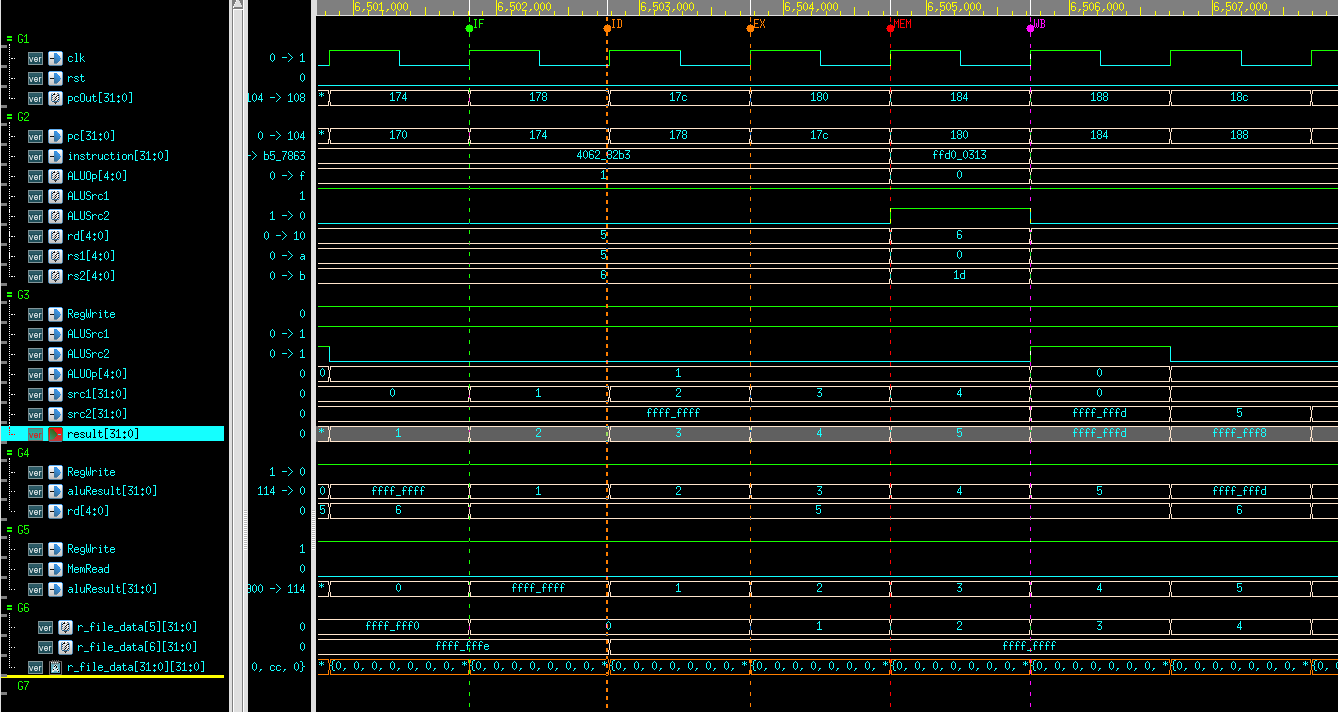


(圖) Program 4 : 75670700 ps

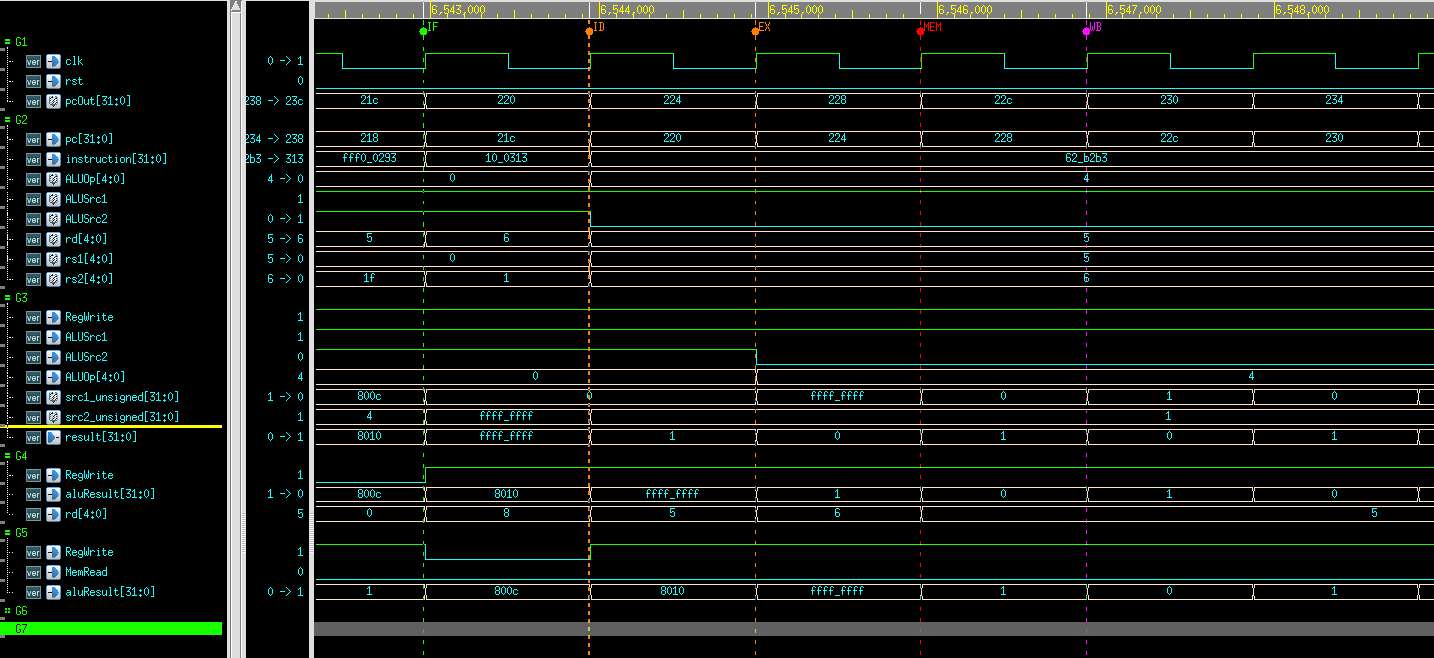
* Waveform

以下波型截圖自Program0：

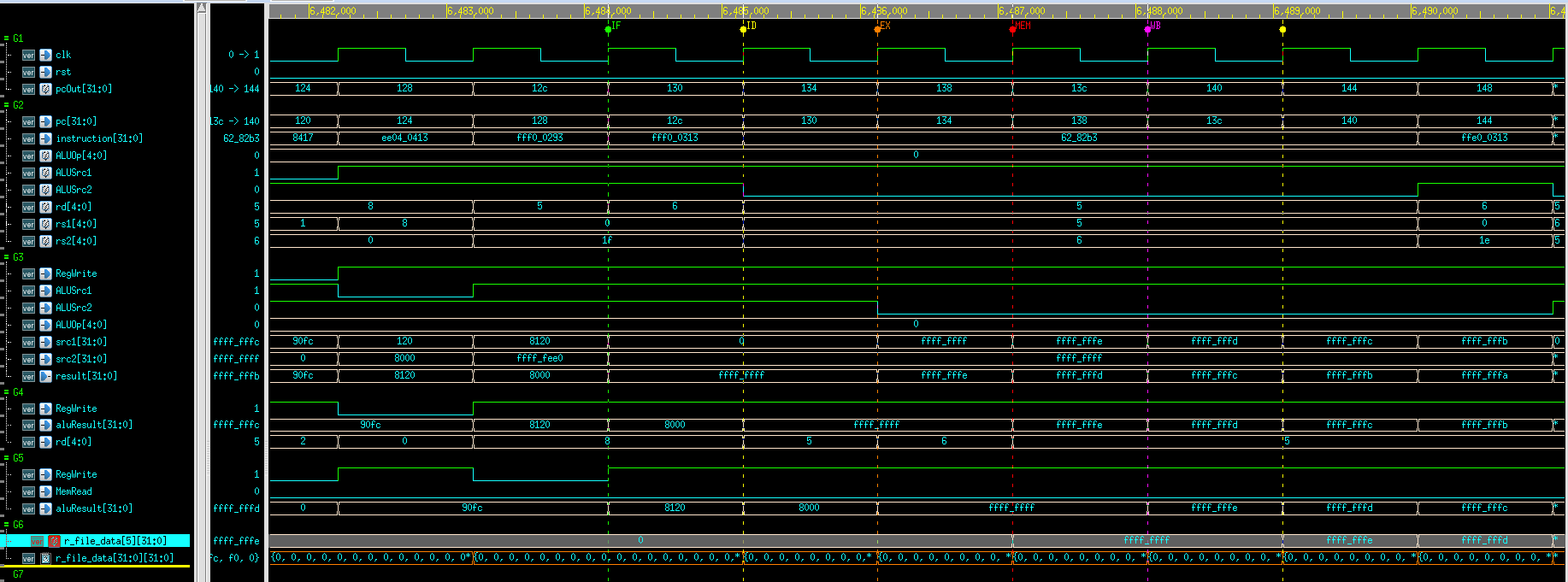
R-Type



(圖) SUB指令 : pc = 0x178



(圖) SLTU指令：pc = 0x220



(圖) ADD指令：pc = 0x130

在Decode為R-Type指令後，Control Unit會將ALUSrc1訊號設為1, ALUSrc2訊號設為0，代表ALU input為Register File的資料。

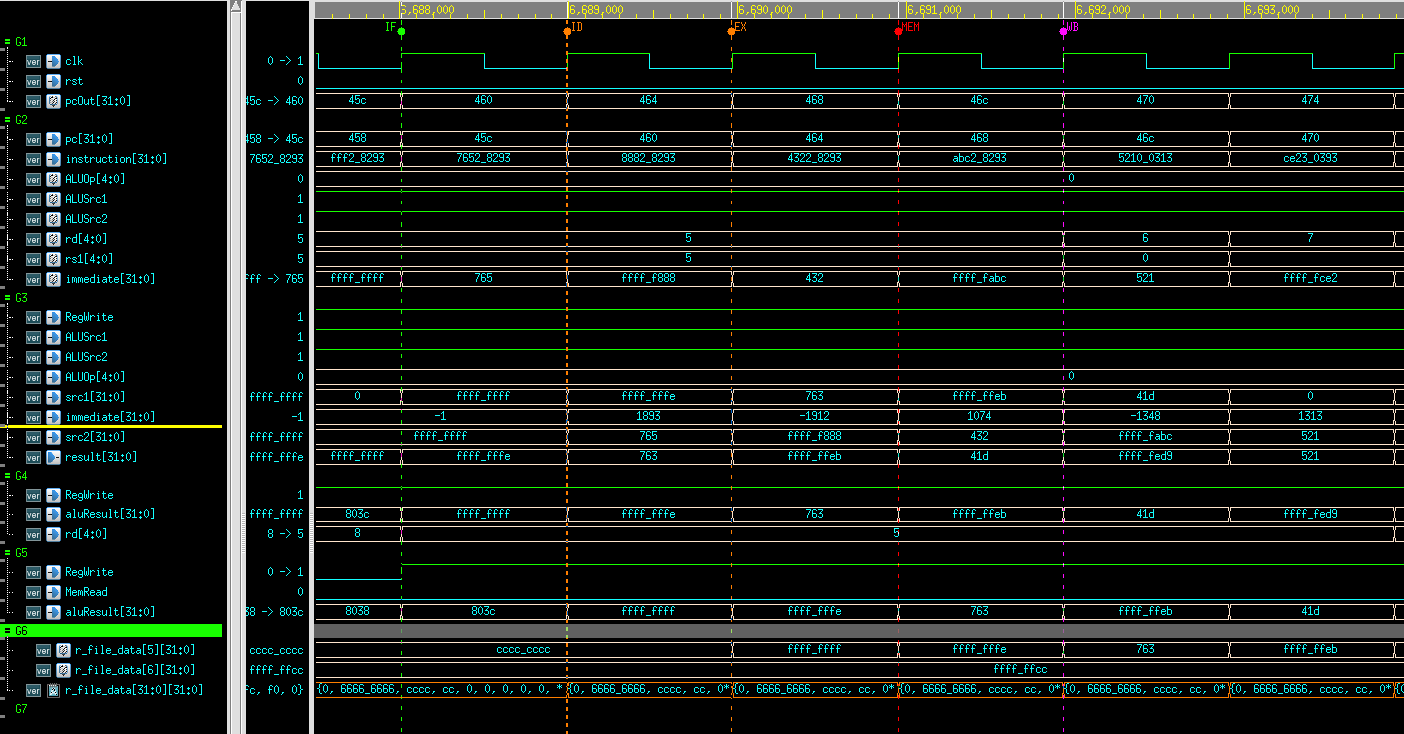
首先看SUB指令，觀察EXE stage，ALU結果為 3-(-1) = 4，並在Write back後下一個正緣寫入t1。

再來看SLTU指令，為了在ALU執行unsigned運算，必須有另外宣告為unsigned的logic，結果為(ffff\_ffff > 1)。

最後是ADD指令，ALU結果為(-1)+(-1) = -2，並寫回t0。

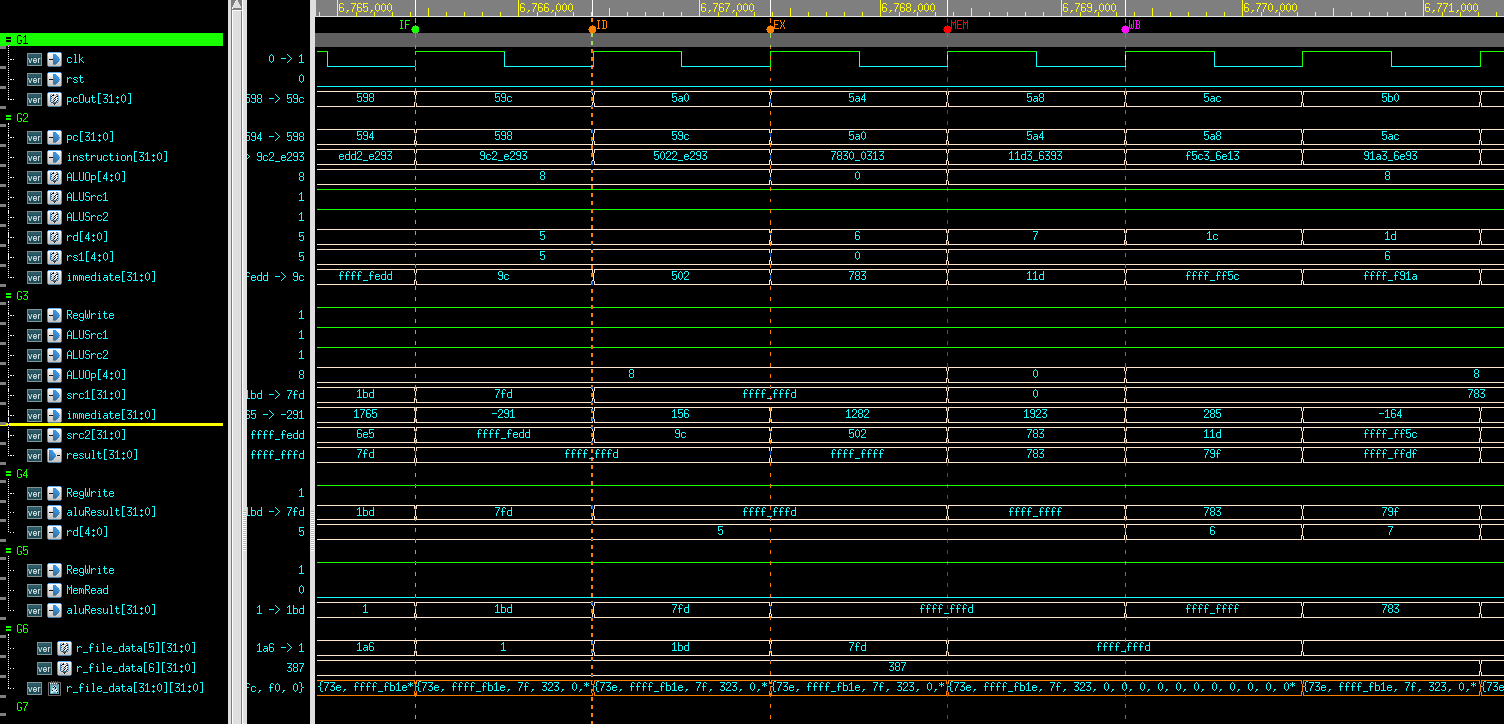
I-Type

I-Type指令和R-Type指令相似，但要將ALUSrc2設為1。首先觀察ADDI指令，結果為0x763+0xffff\_f888 = 0xffff\_ffeb，並寫回t0。



(圖) ADDI指令：pc = 0x460

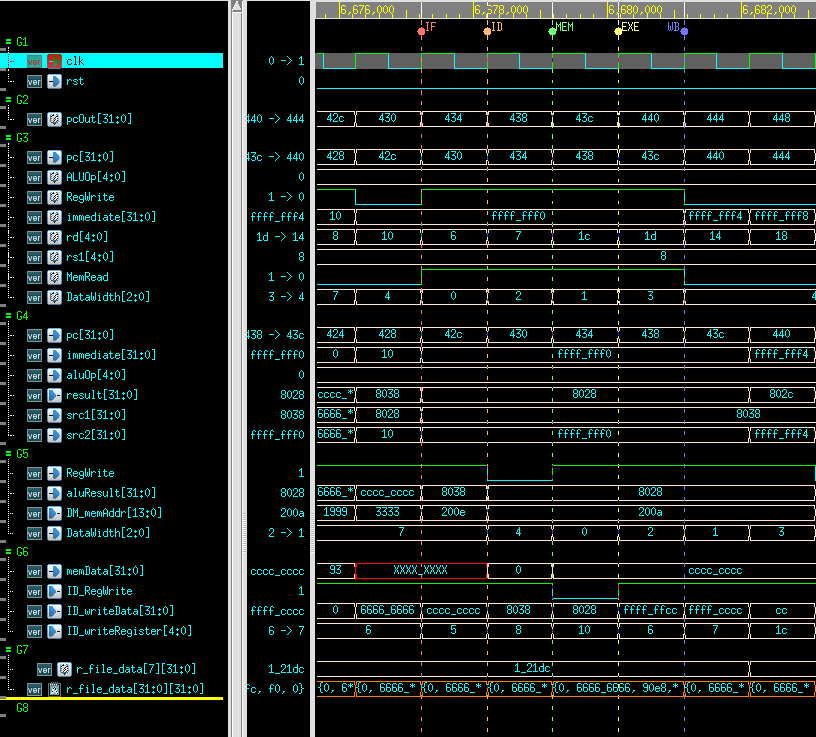
再來觀察ORI指令，結果為0xffff\_fff7 OR 0x502 = 0xffff\_ffff，並寫回t0。



(圖)ORI指令：pc = 0x59c

最後觀察LH指令，Control Unit必須傳送資料寬度訊號(DataWidth)，有Byte, Byte Unsigned, Half, Half Unsigned和Word，而只有Load才有是否unsigned的區別。在EXE級，使用ALU加法計算Memory的存取地址。在MEM級，將ALU運算結果右移2bit(Alignment)傳給Data Memory。在WB級，傳回的Data必須透過DataWidth和ALU result的最右2bit判斷資料如何回傳給RegisterFile。

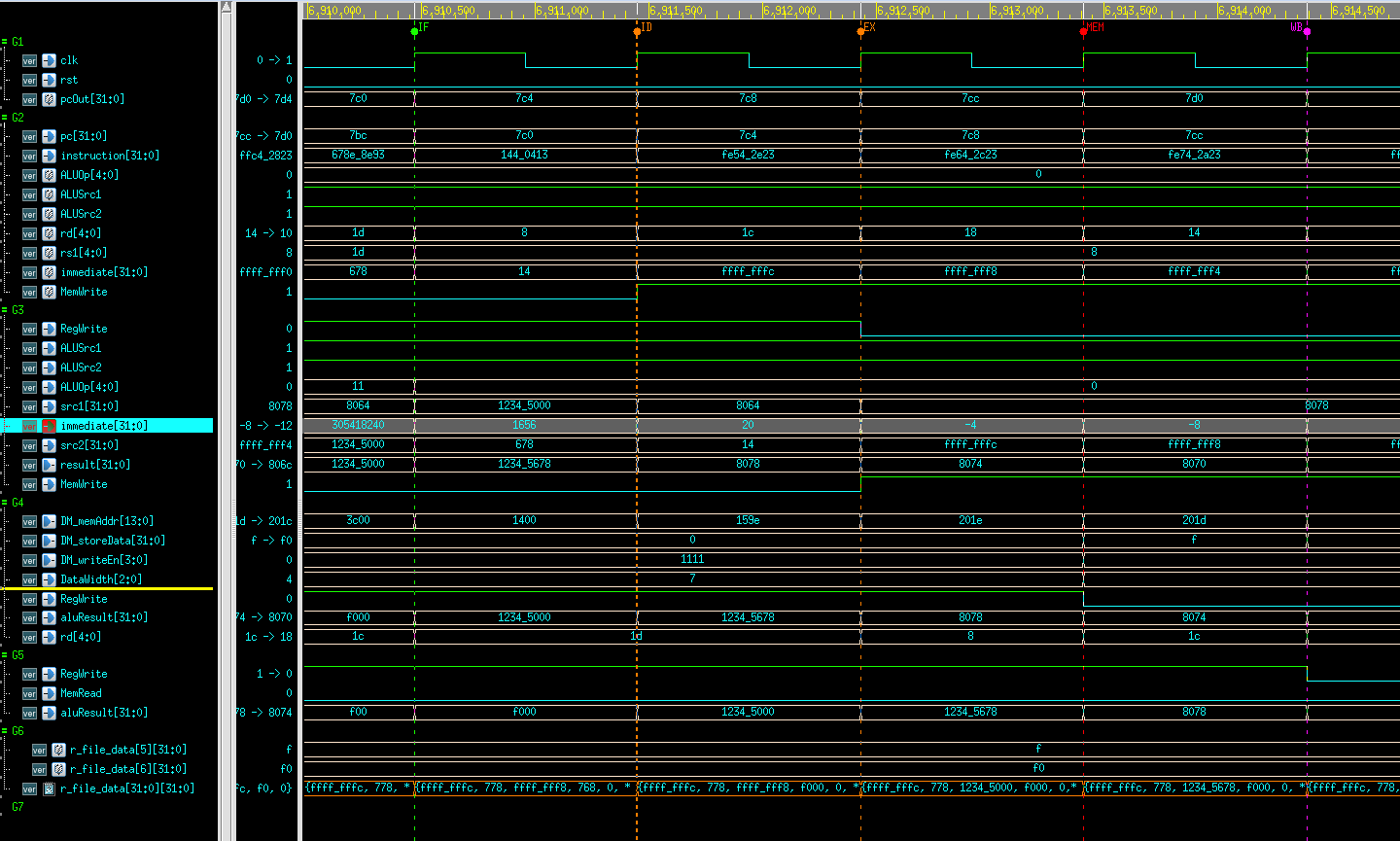
在該條指令中，ALU計算結果(0x8028)的最右2bit是00，因此所要的資料是在Memory回傳資料(0xcccc\_cccc)中的右邊16bit，並做sign-extension後，變成0xffffcccc回傳給Register File。



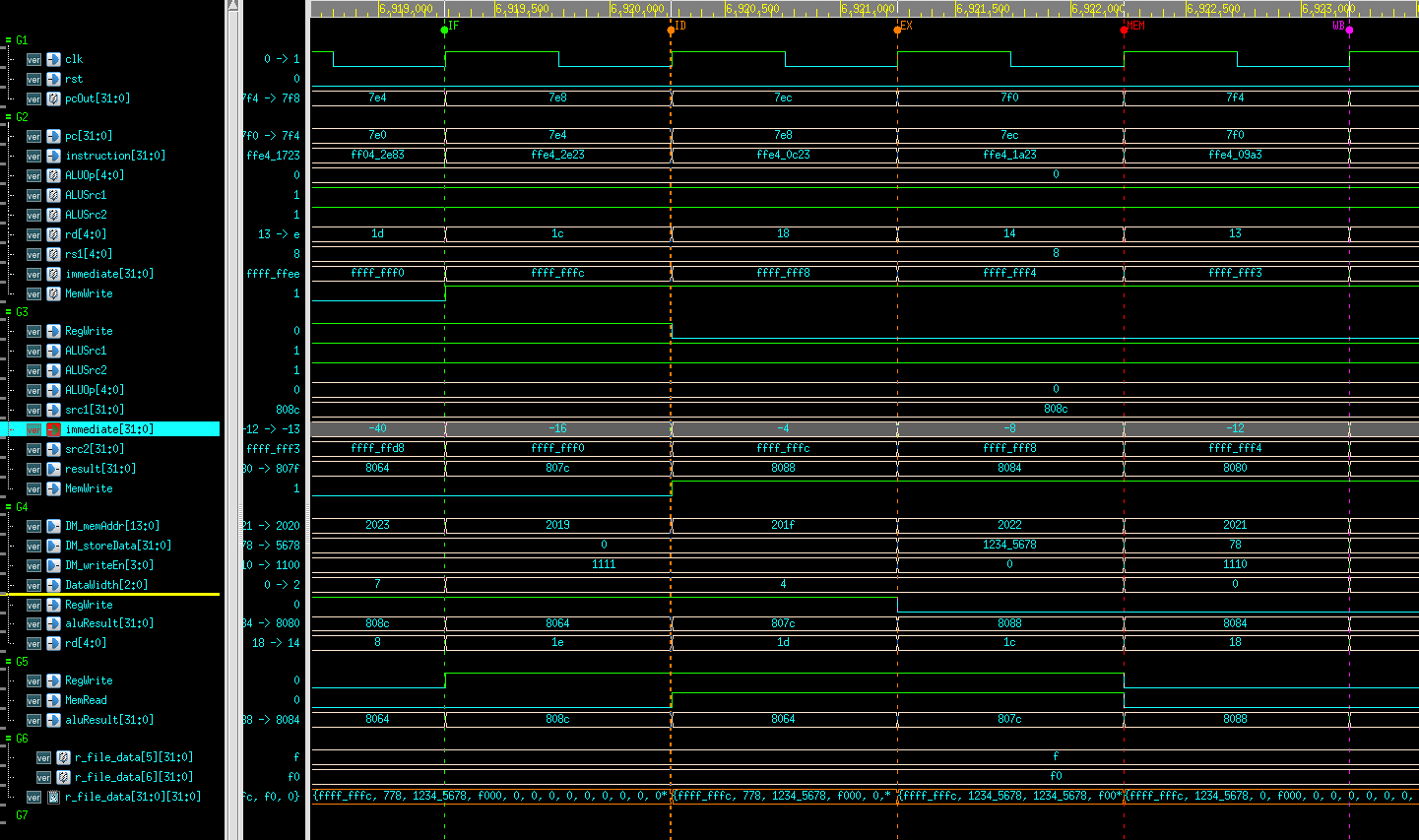
(圖)LH指令：pc = 0x434

S-Type

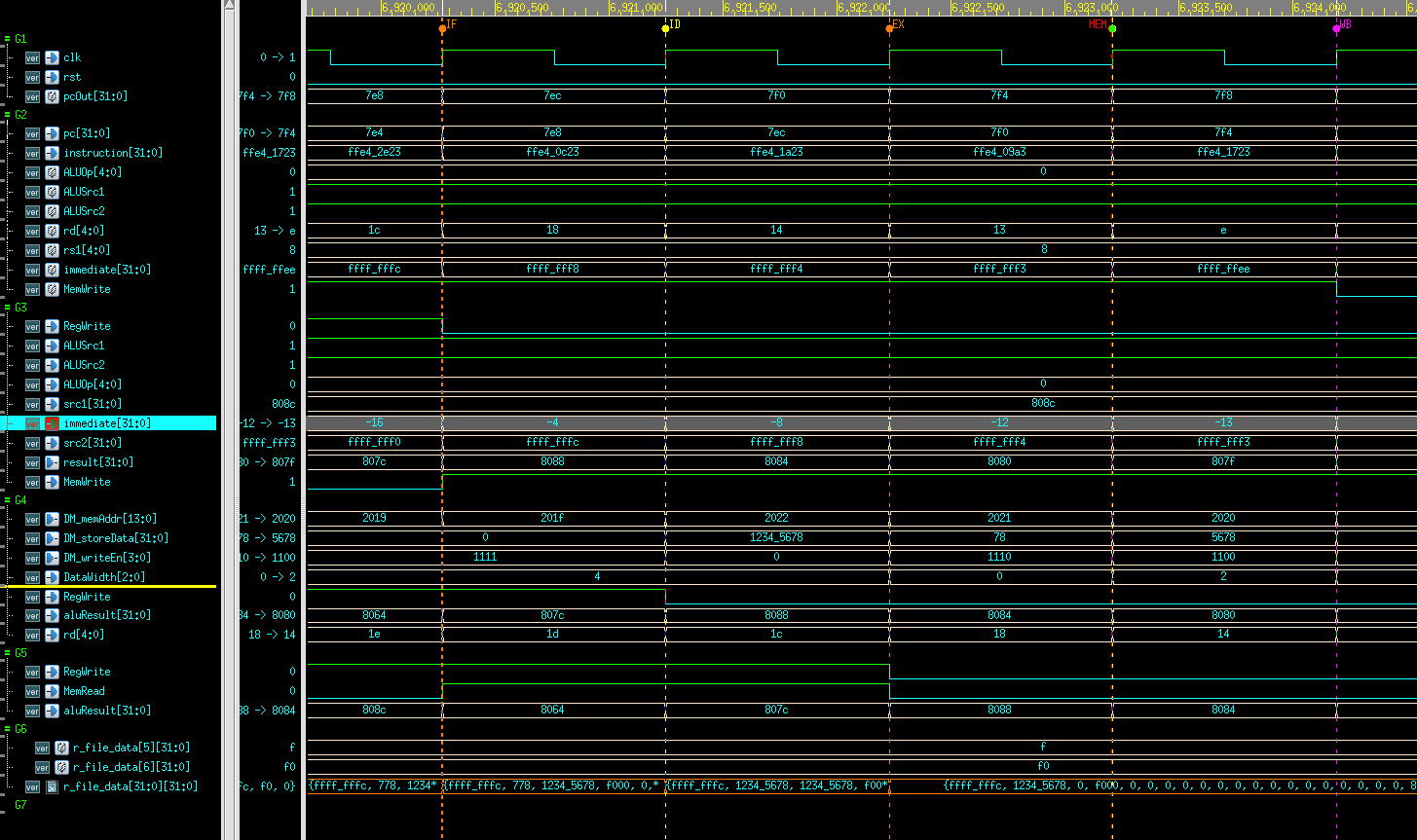
S-Type指令和上方的Load指令處理差不多，只差在因為透過Write enable，存入的最小寬度可以到byte，因此不需要處理sign bit。



(圖) SW指令：pc = 0x7c4



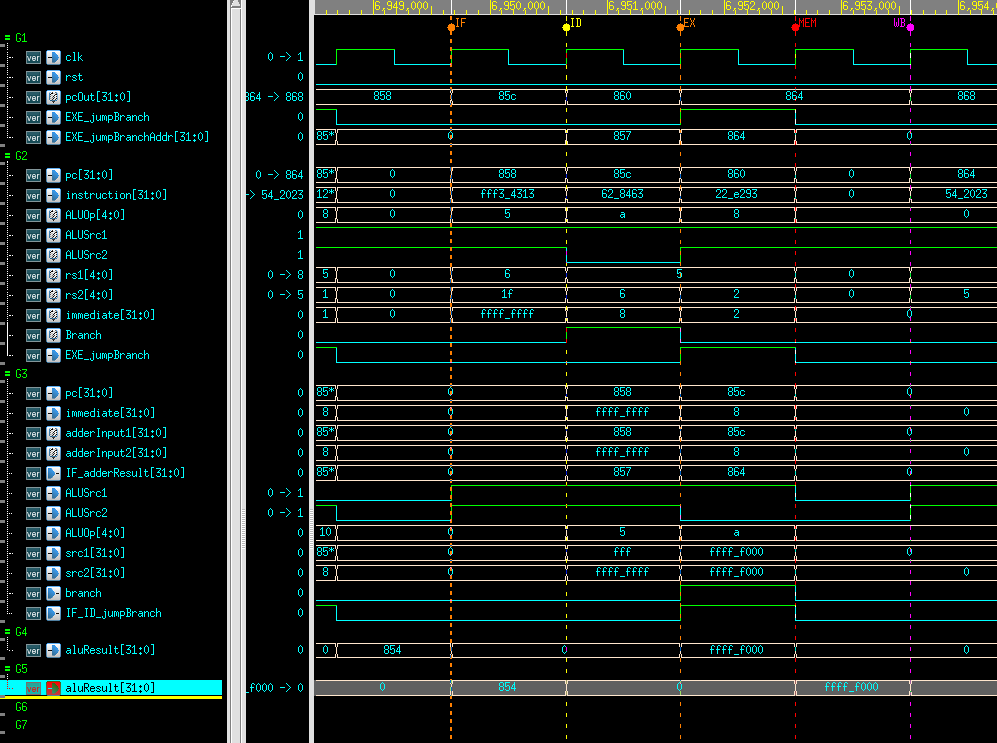
(圖)SB指令：pc = 0x7e8



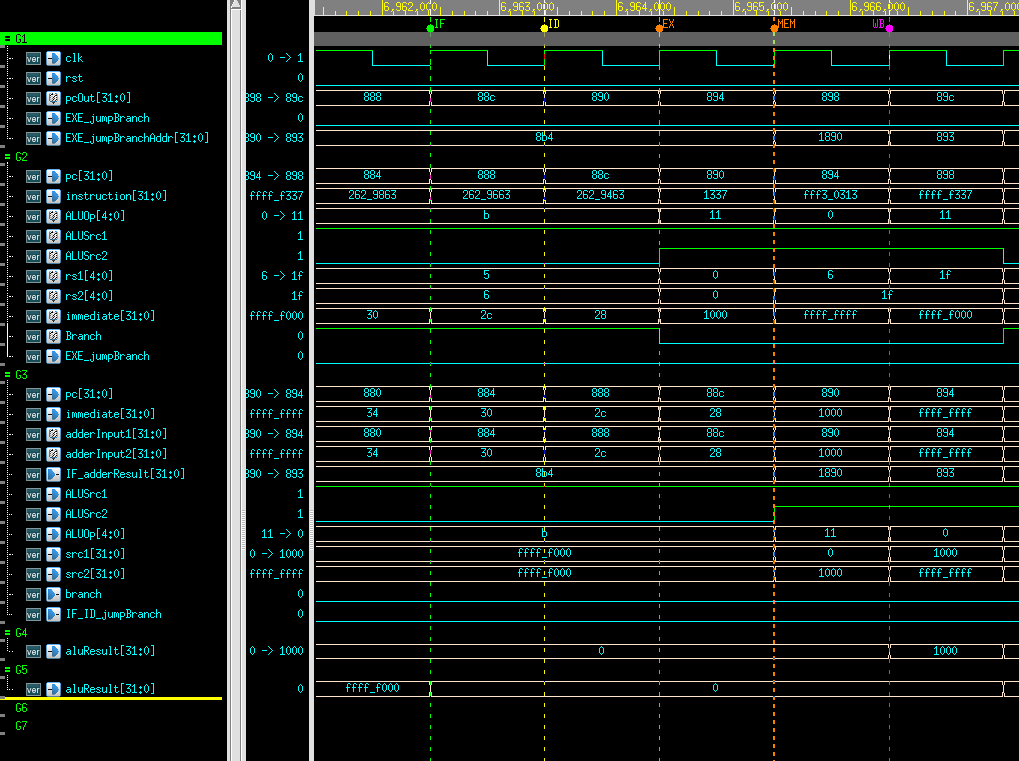
(圖) SH指令：pc = 0x7ec

B-Type

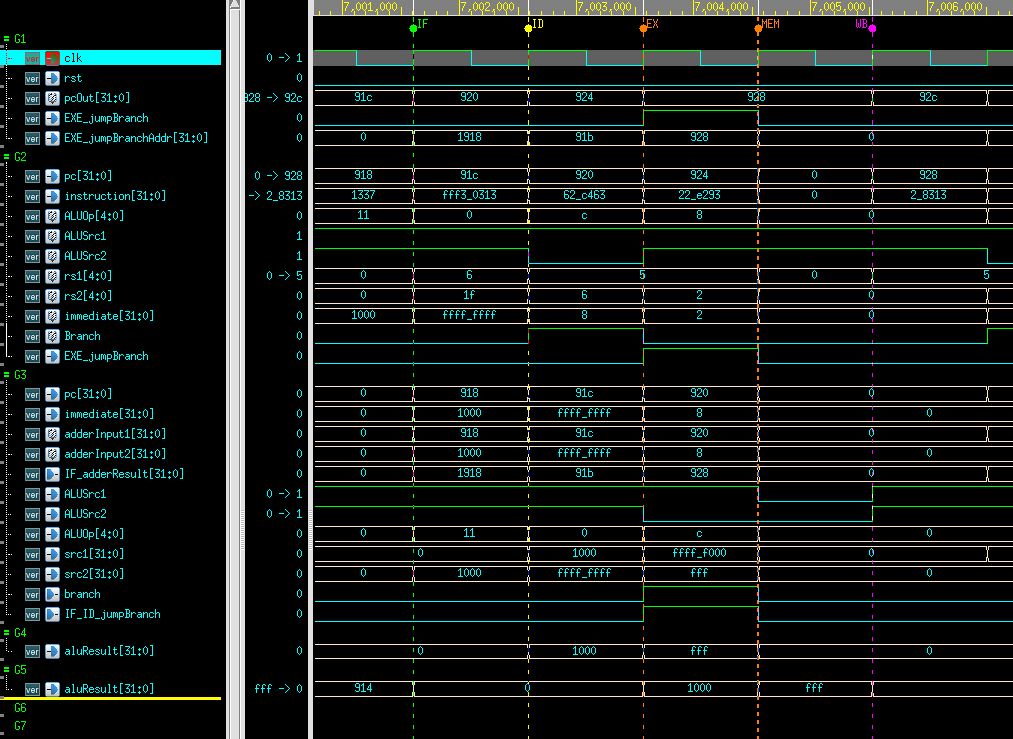
B-Type指令則和Design Explanation介紹的差不多，可以從pc值觀察到會造成1個cycle的delay。EXE stage的IF\_adderResult是指令的目標地址，IF\_ID\_jumpBranch則是是否要進行跳轉。



(圖)BEQ指令：pc = 0x85c



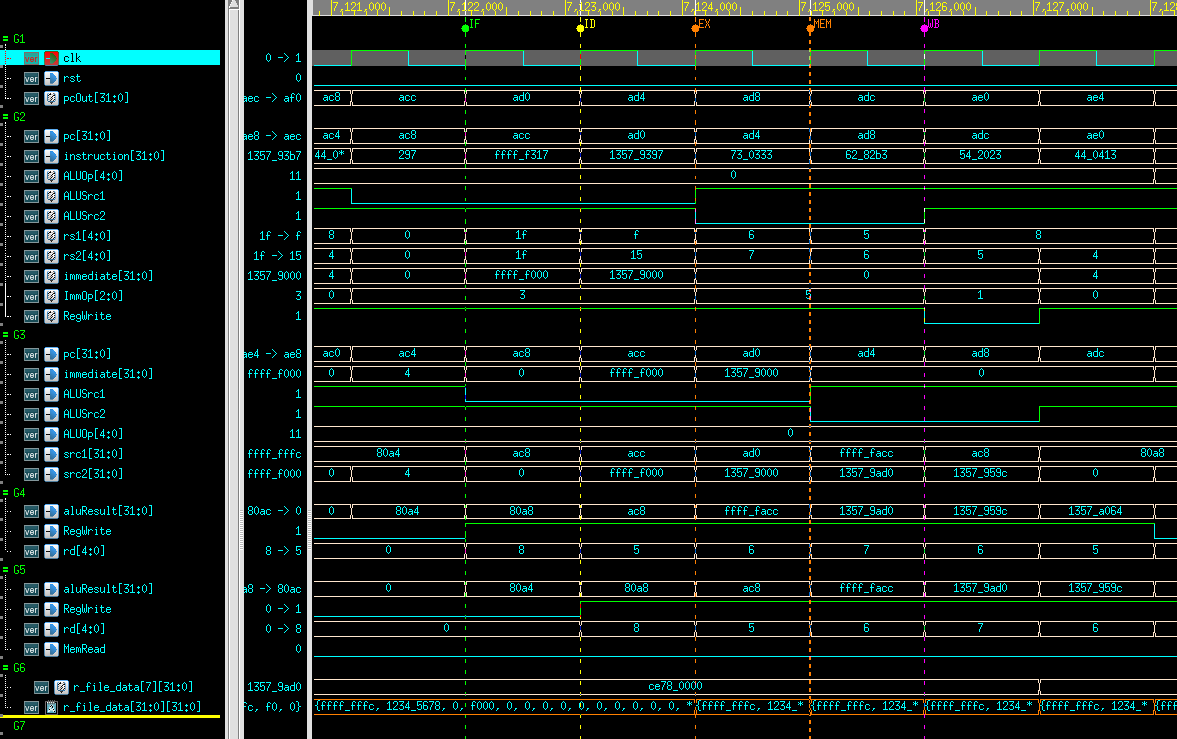
(圖)BNE指令：pc = 0x88c



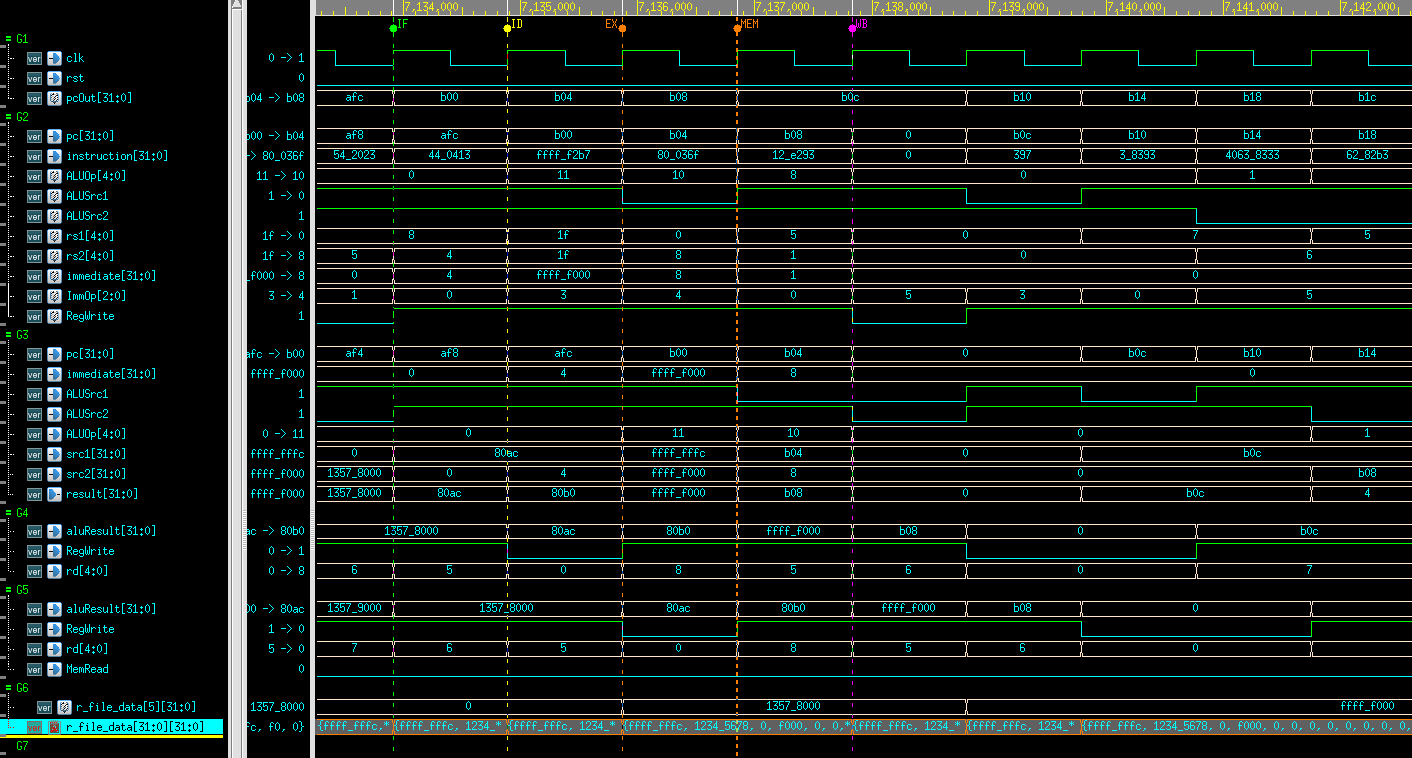
(圖)BLT指令：pc = 0x920

U-Type

U-Type指令只有兩條：AUIPC和LUI，原則上和I-Type指令差異不大。



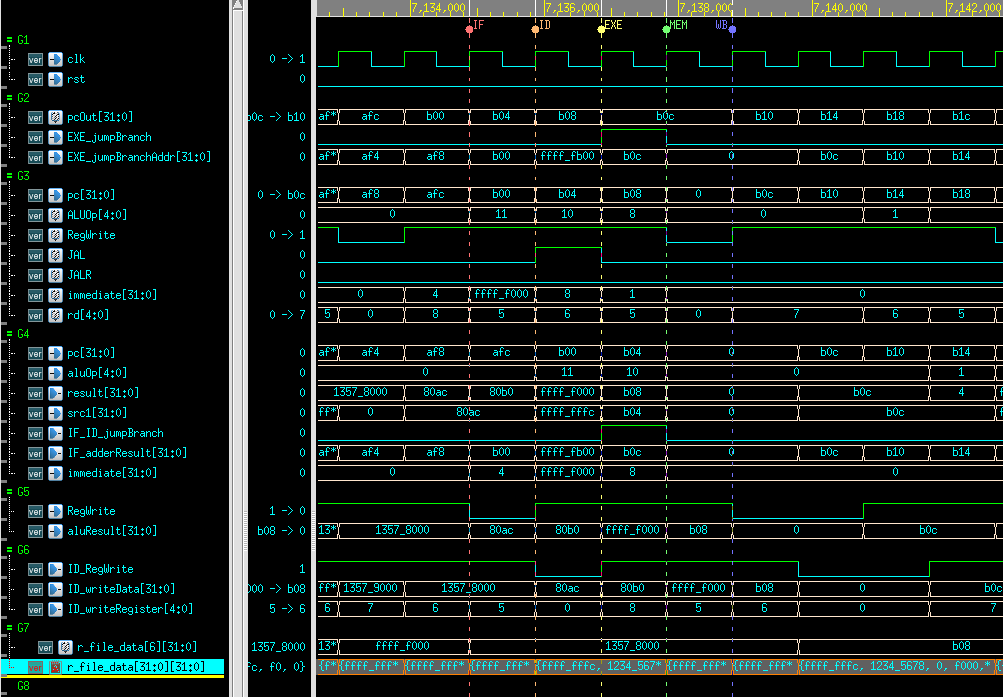
(圖)AUIPC指令：pc = 0xad0



(圖)LUI指令：pc = 0xb00

J-Type

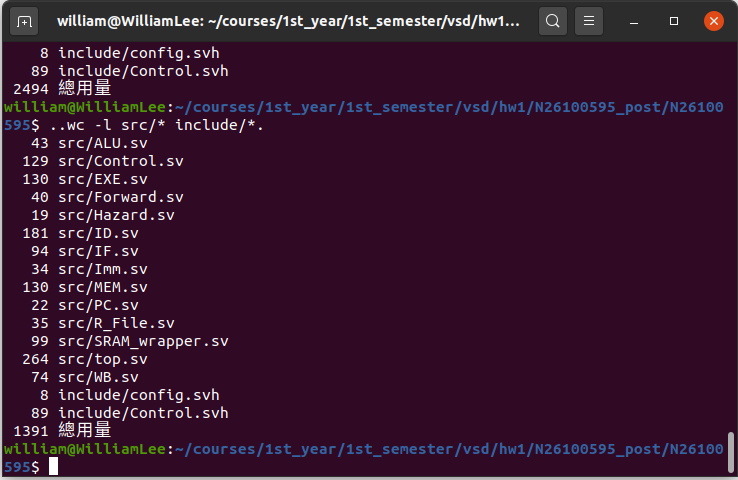
J-Type指令則和Branch指令差異不大，為了簡化設計，本CPU讓跳轉指令的目標地址都是由EXE stage的adder輸出。



(圖)JAL指令：pc = 0xb04

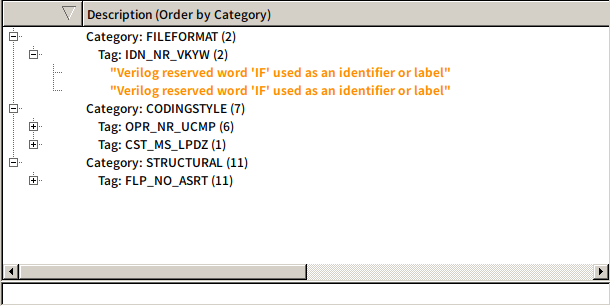
* Superlint Result

* + Total Line: 1391 lines

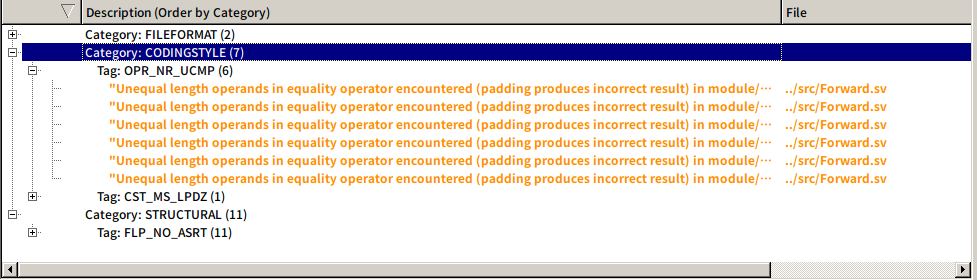


* + Most Frequent Warnings:

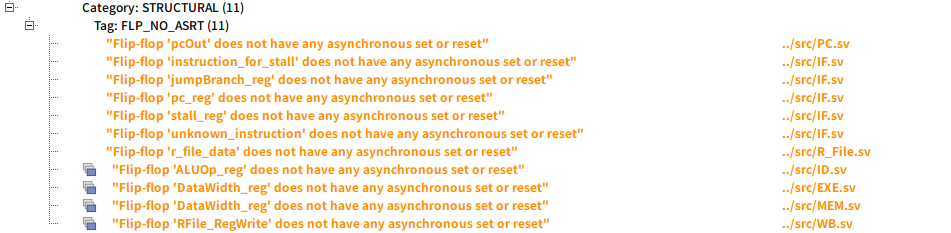
File Format: IDN\_NR\_VKYW



Coding Style: OPR\_NR\_UCMP



Structural: FLP\_NO\_ASRT



Modification:

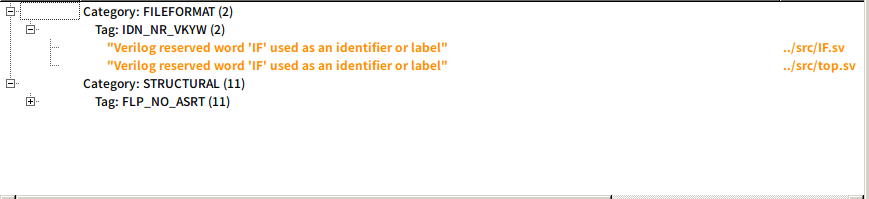
File format的部分，由於IF的Module名稱是設計所造成，就不特別修改。Structural的部分，同樣也是設計上不打算使用Asynchronous的reset，也不做修改。Coding Style的部分則將修改為下列表示法：(紅框為修改後)

一張含有 文字 的圖片

自動產生的描述

Result:

Superlint comply 100% - (13/1391) = **99%** of RTL code.



* Synthesis Result

|  |  |
| --- | --- |
| Clock Period | Cell Area |
| **9.8** ns | **5586700.873177** |

Timing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Report : timing

        -path full

        -delay max

        -max\_paths 1

Design : top

Version: O-2018.06

Date   : Mon Oct 11 21:12:29 2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: WCCOM   Library: fsa0m\_a\_generic\_core\_ss1p62v125c

Wire Load Model Mode: enclosed

  Startpoint: DM1/i\_SRAM (rising edge-triggered flip-flop clocked by clk)

  Endpoint: ID/readData2\_reg\_reg[2]

            (rising edge-triggered flip-flop clocked by clk)

  Path Group: clk

  Path Type: max

  Des/Clust/Port     Wire Load Model       Library

  ------------------------------------------------

  top                enG1000K              fsa0m\_a\_generic\_core\_ss1p62v125c

  WB                 enG5K                 fsa0m\_a\_generic\_core\_ss1p62v125c

  EXE                enG10K                fsa0m\_a\_generic\_core\_ss1p62v125c

  ALU\_DW01\_cmp6\_4    enG5K                 fsa0m\_a\_generic\_core\_ss1p62v125c

  ALU                enG5K                 fsa0m\_a\_generic\_core\_ss1p62v125c

  ID                 enG30K                fsa0m\_a\_generic\_core\_ss1p62v125c

  Point                                    Incr       Path

  -----------------------------------------------------------

  clock clk (rise edge)                    0.00       0.00

  clock network delay (ideal)              1.00       1.00

  DM1/i\_SRAM/CK (SRAM)                     0.00 #     1.00 r

  DM1/i\_SRAM/DO7 (SRAM)                    5.71       6.71 r

  DM1/DO[7] (SRAM\_wrapper\_1)               0.00       6.71 r

  WB/memData[7] (WB)                       0.00       6.71 r

  WB/U40/O (OR3B2)                         0.18       6.88 f

  WB/U35/O (AN4P)                          0.35       7.23 f

  WB/U94/O (OR3B2)                         0.18       7.41 r

  WB/U93/O (BUF6)                          0.27       7.68 r

  WB/ID\_writeData[9] (WB)                  0.00       7.68 r

  EXE/WB\_writeData[9] (EXE)                0.00       7.68 r

  EXE/U6/O (INV8)                          0.05       7.73 f

  EXE/U255/O (OAI112HT)                    0.19       7.93 r

  EXE/alu/src2[9] (ALU)                    0.00       7.93 r

  EXE/alu/r322/B[9] (ALU\_DW01\_cmp6\_4)      0.00       7.93 r

  EXE/alu/r322/U215/O (INV2)               0.11       8.04 f

  EXE/alu/r322/U213/O (XNR2HP)             0.20       8.24 f

  EXE/alu/r322/U211/O (OAI12H)             0.19       8.42 r

  EXE/alu/r322/U216/O (AOI12H)             0.09       8.52 f

  EXE/alu/r322/U242/O (OAI12HP)            0.17       8.69 r

  EXE/alu/r322/U217/O (AOI12HP)            0.07       8.76 f

  EXE/alu/r322/U251/O (OAI12HP)            0.19       8.95 r

  EXE/alu/r322/U253/O (AOI12HT)            0.11       9.05 f

  EXE/alu/r322/U252/O (OAI12HT)            0.19       9.25 r

  EXE/alu/r322/GE (ALU\_DW01\_cmp6\_4)        0.00       9.25 r

  EXE/alu/U76/O (ND2F)                     0.08       9.32 f

  EXE/alu/U73/O (OAI112HP)                 0.08       9.41 r

  EXE/alu/U70/O (AN4B1P)                   0.14       9.54 f

  EXE/alu/branch (ALU)                     0.00       9.54 f

  EXE/U26/O (AOI12HT)                      0.19       9.73 r

  EXE/U25/O (INV12)                        0.10       9.83 f

  EXE/IF\_ID\_jumpBranch (EXE)               0.00       9.83 f

  ID/EXE\_jumpBranch (ID)                   0.00       9.83 f

  ID/U16/O (OR2T)                          0.23      10.06 f

  ID/U15/O (INV12)                         0.26      10.32 r

  ID/U22/O (AN2S)                          0.28      10.61 r

  ID/readData2\_reg\_reg[2]/D (QDFFN)        0.00      10.61 r

  data arrival time                                  10.61

  clock clk (rise edge)                    9.80       9.80

  clock network delay (ideal)              1.00      10.80

  clock uncertainty                       -0.10      10.70

  ID/readData2\_reg\_reg[2]/CK (QDFFN)       0.00      10.70 r

  library setup time                      -0.09      10.61

  data required time                                 10.61

  -----------------------------------------------------------

  data required time                                 10.61

  data arrival time                                 -10.61

  -----------------------------------------------------------

  slack (VIOLATED: increase significant digits)       0.00

Area

Area

-----------------------------------

Combinational Area: 163358.294174

Noncombinational Area: 78848.079002

Buf/Inv Area: 33026.010726

Total Buffer Area: 22820.41

Total Inverter Area: 10205.60

Macro/Black Box Area:

5344494.500000

Net Area: 0.000000

-----------------------------------

Cell Area: 5586700.873177

Design Area: 5586700.873177

* Summary

在本次作業中，我完成了針對RISC-V指令集中37條指令的CPU設計，並完成了4個program的測試，以及Superlint comply 99%的結果。在報告中，我介紹了本設計各個stage的功能，以及Jump/Branch指令、Hazard Detection、Forwarding、Memory address alignment … 等相關機制。

在合成方面，CPU的Clock period可以最低至9.8 ns，面積為5586700，並可通過4個Program測試。

* Lessons Learned

雖然大學時有修過李昆忠老師的VLSI，期末project也是做pipelined CPU，但當時是和一群人一起合作，我認為最大的差別就是，總是會有人能夠發現某些設計的盲點，因此最後做出來並沒有太多需要Debug的部分。但自己完成一個設計，會有很多部分必須要自己後來Debug才能發現，過程十分痛苦，而東修一點，西修一點的方式也會讓整個設計到後面變得很凌亂，因此在下個作業前，還有一些設計要被調整。

話雖如此，自己獨力完成一個設計，對於整個CPU的架構更加理解，對Hazard Detect, Jump/Branch的機制也更加清楚，也學習到如何使用架構圖先具象化自己的設計，也讓後來的修改相對輕鬆。後面還有更加困難的挑戰，希望自己可以一一克服。

-------------------------------End--------------------------------