VLSI System Design (Graduate Level)

Fall 2023

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: 吳欣航

Student ID: N26124395

**RISC-V 架構**

1. RISC-V 規格

這個作業要求使用RISC-V 32IM指令集作為CPU的基礎，並設計一個具有45個不同功能的指令集，包括4個控制和狀態暫存器（CSR）相關指令。該CPU內部擁有32個32位元暫存器，一個32位元的指令計數器，以及兩個64位元的CSR寄存器。採用了5級流水線架構，包括指令取得（IF）、指令解碼（ID）、執行（EX）、記憶（MEM）、以及寫回（WB）等五個階段。此外，它使用兩個外部SRAM模塊作為指令記憶體（IM1）和數據記憶體（DM1）。在硬體設計方面，它考慮了解決data hazard、control hazard和structural hazard的需求。

1. RISC-V 設計

一張含有 寫生, 圖表, 圖畫, 方案 的圖片

自動產生的描述

1. Instruction Fetch (IF) :

第一個階段是取指階段（Instruction Fetch）。在這個階段CPU 從iSRAM(IM1) 中讀取指令，這是透過Program Counter的值來實現的，PC 會不斷增加4，指向下一條要執行的指令，有一個mux可以選擇是要pc+4或是跳轉的位址。將PC接進SRAM裡面就可以得到instruction，取得的指令會存放在Reg\_IF\_ID中，以備後續處理。由於本次作業的SRAM會延遲一個週期，這對設計CPU造成困難點，我的解決方法是將IM1讀出來的值直接接到下一個階段，同時讀出來的instruction也會同時記錄到Reg\_IF\_ID裡面，並在下一個階段使用一個mux將他們做選擇，常理來說直接將SRAM讀出來的值接到DECODER就好，並不用再接到pipeline register，這樣的做法是為了stall設計，因為若是stall，有時需要保留上一個clk的instruction，因此需要存到暫存器裡面。

1. Instruction Decode (ID) :

接下來是指令解碼階段（Instruction Decode）。CPU 對取得的指令進行解碼，識別出它是什麼類型的指令以及需要執行哪些操作，將解碼出來的東西傳遞到RegFile和Controller。本次作業總共有32個32bit暫存器，這部分的時現在RegFile，可以將我們的解碼得到的rs1\_index和rs2\_index透過RegFile取得實際上的數值，可以看到有兩個選擇器，這邊的功用是為了解決forwarding，MUX另外一組數值為第五個階段回傳回來的值。此外還有一個module 為IMM\_Ext，這個module的用途是因為RISC-V有不同的type，不同的type所對應到的immediate位置都不一樣，需要補0的地方也不一樣，因此針對規格書的定義去訂製此模組，達到效果。最後再將需要紀錄的數值存進REG\_ID\_EXE中。

1. Execute (EXE) :

這個階段是比較複雜的階段，可以看到一開始有兩個三對一的選擇器，這邊也是為了解決Forwarding的部分，會有三個可以選擇的原因是它可以是MEM階段的回傳值，也可以是WB階段的回傳值，因此需要三對一的選擇器。之後接到兩個二對一的選擇器，由於這階段不單單只有rs1、rs2的邏輯運算，也牽扯到了jump的地址運算以及與immediate的運算，因此需要使用MUX做選擇，爾後會有兩個值進到ALU和MUL做運算，其中ALU針對規格書，不同op code會有不一樣的動作，乘法器裡面也是，尤其正負數問題困擾我許久，必須先了解verilog的乘法原理，會將運算子定義為32+32=64bit，並做位元填充，若是sign則填充operand[31]，若是unsign則填充0。CSR主要是計算cycle數量以及instruction數量，要輸入當下階段的pc，若是發現pc不為32’b0則instruction+1，cycle則是不管怎麼樣就+1。JB unit的部分是用來處理地址的跳轉計算，計算完會傳回第一階段。最後將需要紀錄的數值存進REG\_EXE\_MEM中。

1. Memory (MEM) :

在這個階段相對單純，只是將計算出來的地址以及數值寫進DM1中，比較需要注意的是SRAM是active low，跟想像中的相反，需要特別注意。此外由於本次做也是word address，也就是memory中的單位是word，相對byte address，輸入DM1的地址只有14bit，也就是說將byte address的地址除以4，然而算出來的地址不一定都是4的倍數，像是SB、SH指令，因此需要針對餘數為0、1、2、3的情況去做位移對齊，因為word address和byte address的起始位置會不一樣，若不處理對齊的情況，會造成數值的錯誤，因此在DM1的前面有設計了一個wen\_shift的module來處理此問題，原理也是將memory enable signal和address位元填充。DM1會延遲一個週期，因此直接將他的輸出接到下一個階段。

1. Write Back (WB) :

這個階段主要的功能為處理指令計算結果寫回RegFile。會先經過一個Ld\_Filter用來處理LB、LH的問題，ld\_filter與wen\_shift為相對的關係，針對讀值做memory enable訊號的處理 。爾後將wb\_data傳回ID、EXE階段，寫回rd或是forwarding。

1. HAZARD
2. Structure hazard

在本次作業不會遇到，因為使用兩個SRAM(IM1, DM1)，確保structure hazard不會發生。

1. Data hazard

本次作業設計的CPU架構，發生資料來不及存入暫存器就要使用的情況，這種狀況就稱為data hazard。解決方法是採用forwarding的技術，分別為WB到ID，WB到EXE，MEM到EXE。Forwarding在pipeline CPU中有一個重要的觀念是Forwarding，可以直接將pipeline registors的值傳遞到需要的地方，主要有ID階段和EXE階段可以選擇，這邊就需要透過mux來選擇所要的訊號。

ID階段主要有兩個控制訊號，D\_rs1\_data\_sel和D\_rs2\_data\_sel。D\_rs1/2\_data\_sel訊號是rs1/2 data與wb data在選擇，判斷的條件大致是這樣，判斷W的rd有沒有被使用，判斷D的rs1/2有沒有被使用，在判斷W\_rd和D\_rs1/2是不是有重複，如果有則選擇wb\_data。

EXE階段則會有E\_rs1\_data\_sel和E\_rs2\_data\_sel控制訊號。判斷的條件是判斷M的rd有沒有使用，判斷W的rd有沒有使用，判斷E的rs1/rs2有沒有使用，如果有，判斷他們有沒有重複，如果E\_rs1/2與M\_rd有重複，那就選擇MEM的值，若是E\_rs1/2與W\_rd有重複，那就選擇WB的值。

如果forwarding依然無法解決data hazard的問題，那只能透過輸出空指令來達成stall將pc記數暫停，以解決data structure。

1. Control hazard

jtype與btype會遇到的hazard，由於在EXE才會計算出是否跳躍與跳躍的位址。因此必須Flush前面IF與ID已經進入CPU的指令，也是通過輸出空指令完成。才不會執行到被跳過的指令。

1. 困難點

在這次作業設計5級CPU中，我遇到了以下幾個設計問題。

1. Memory delay:

由於本次作業的SRAM讀值會delay一個cycle，因此需要針對此問題做設計，以往在課本中學到的SRAM是不會delay的，我這邊採用的做法是直接將IM1和DM1的ouput拉到下一個階段直接對齊，然而這在IM1會遇到問題。在stall的時候Reg\_ID\_IF需要輸出上一個cycle的值，若是直接IM1 output inst接到下一個階段，會沒有暫存器去存取上一個cycle的inst，因此我除了接到下一個cycle，也接到Reg\_ID\_IF裡面，爾後用mux去選擇。

一張含有 圖表, 文字, 方案, 工程製圖 的圖片

自動產生的描述一張含有 圖表, 工程製圖, 方案, 圖解 的圖片

自動產生的描述

1. Multiplier

另外一大挑戰是新增了乘法的指令，由於乘法器設計較為複雜，因此需要特別考慮週期的問題，分析timing report會發現critical path總是mul，路徑包括了從WB forwarding回來的值，然後到EXE，然後經過乘法器運算，如下圖橘線，這個路徑會經過兩個最主要的delay，分別是SRAM取值的delay和乘法器運算的delay。我的解決辦法是將MUL移到MEM的一開始運算，這樣可以避免乘法器和SRAM delay同時發生。

另外一個問題則是搞懂verilog在乘法的規則，在verilog中做乘法運算會將較少位的數擴充成最多位的數，因此正負問題就此產生，也就是說若是sign，要擴充成原本的sign bit，若是unsign，填充0即可。在最一開始，我使用最基礎的補碼的概念做正負的處理，但是這樣會使得整個MUL運算週期變得更長，因此以verilog基礎語法設計一個乘法器。

一張含有 圖表, 方案, 工程製圖, 圖解 的圖片

自動產生的描述

MUL

一張含有 圖表, 方案, 工程製圖, 圖解 的圖片

自動產生的描述

1. Wen\_shift

這裡遇到的問題是由上課提到的word address 和 byte address所引起的，SRAM的address input只有14bit，這一開始困擾我許久，後來才發現原來SRAM的input就是address的最小單位-word，也就是將原本的address右移兩位。然而這會發生一個問題，那就是當EXE階段計算出來的address不是4的倍數時會產生問題。這邊的設計是使用wen\_shift這個module對w\_en和address做對齊的動作。

一張含有 圖表, 文字, 方案, 工程製圖 的圖片

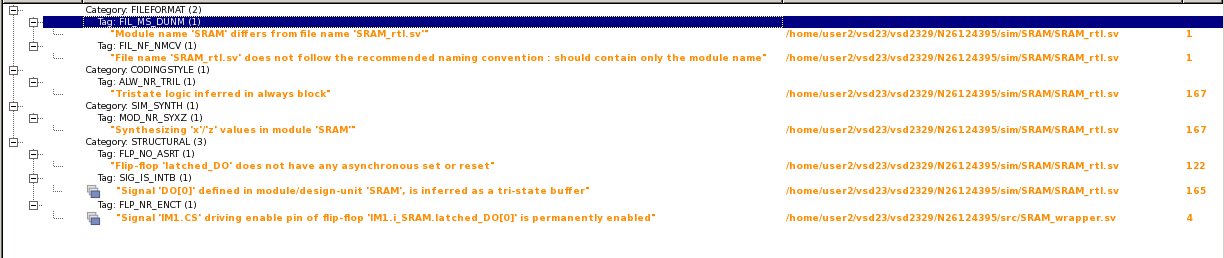
自動產生的描述

**Superlint**

這次作業有使用superlint來檢查語法，他是一個很好用的工具，除可以幫我們檢查基本的語法錯誤外，也可以幫助我們建立寫verilog的良好習慣，例如初始化、non-blocking寫法，也可以幫忙檢查有沒有甚麼線或暫存器是沒有用到的。這次作業有一個規定是superlint的warning不能超過全部行數的15%，我的CPU行數總共有1041行，而最終superlint的結果剩下7個，全部都是跟SRAM有關的警告，符合作業規定。以下說明常見的警告。

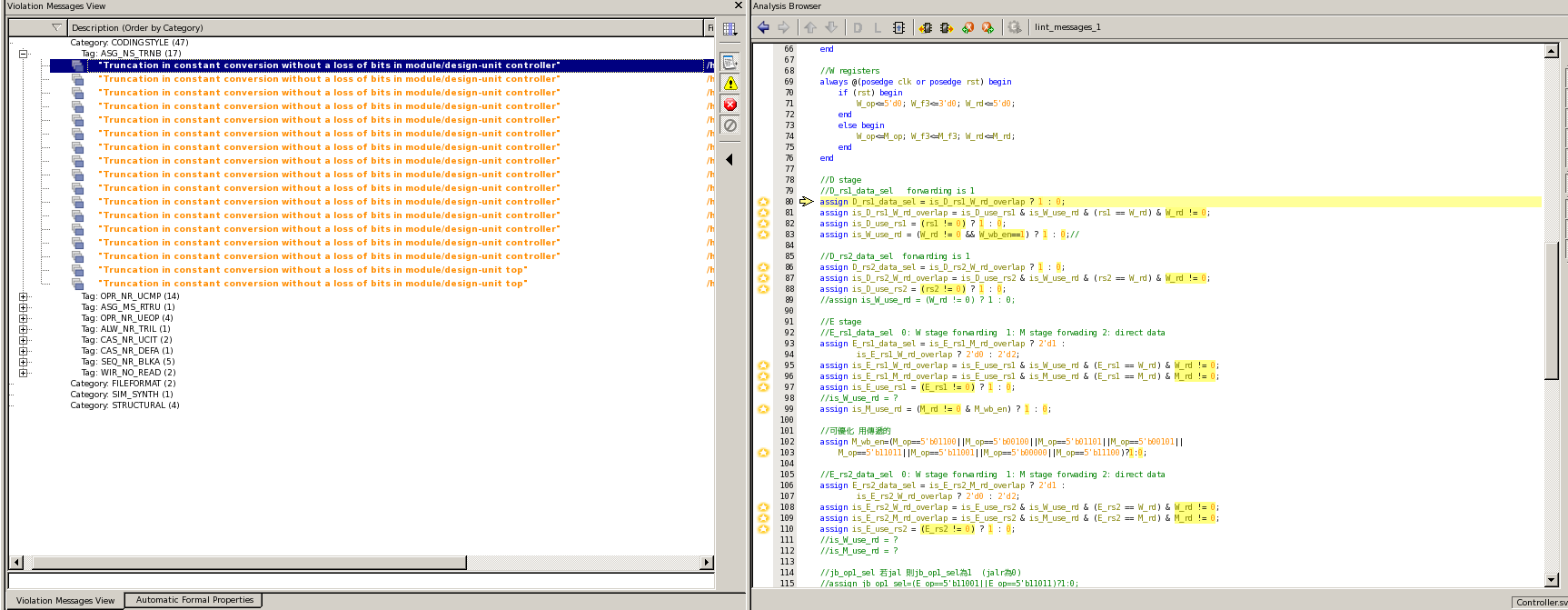
一張含有 文字, 螢幕擷取畫面, 陳列 的圖片

自動產生的描述



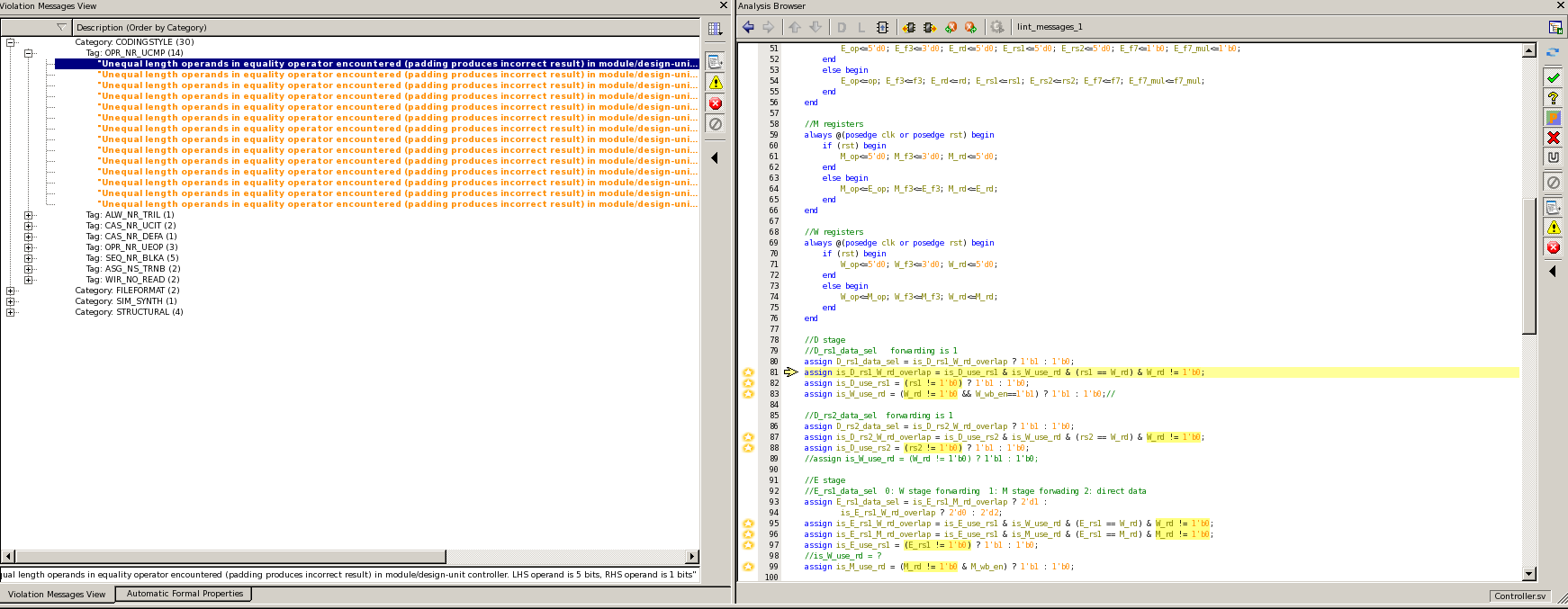
1. ASG\_NS\_TRNB

這個警告是在說我的位元描述的不夠完整，可以看到以下的assign三原運算子，1:0的部分，此部分較好的寫法應該是要將位元也寫上去變成1’b0:1’b1;



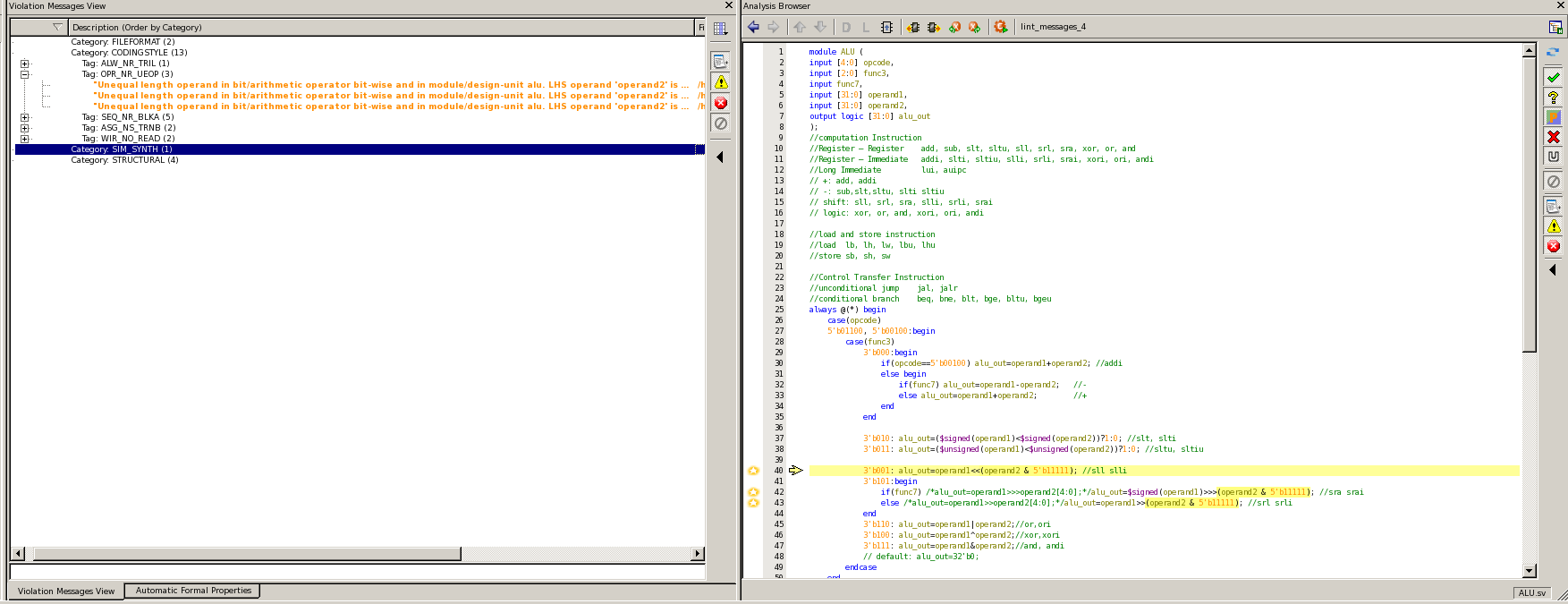
1. OPR\_NR\_UCMP

這個警告說明了我的變數對應到的位元數量是有錯誤的，可以看到下圖，W\_rd是5bit，但我卻讓他等於1bit，因此有了錯誤。



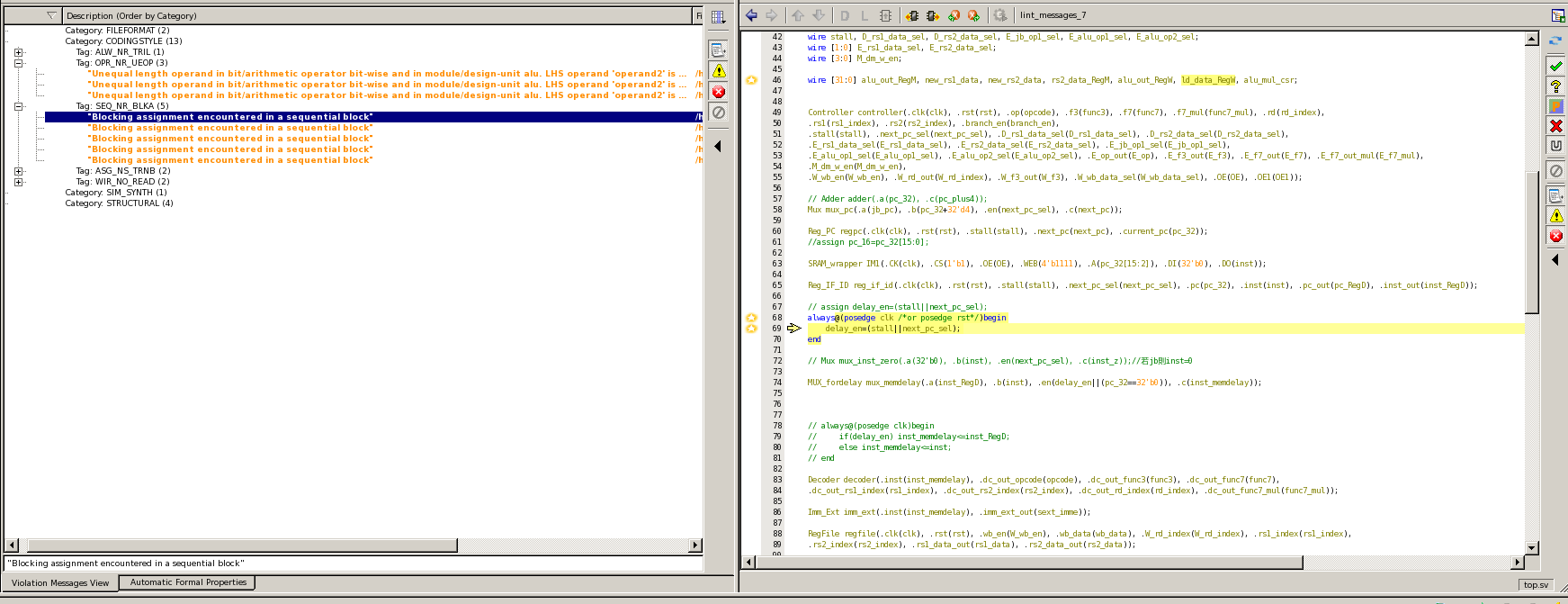
1. OPR\_NR\_UEOP

這個警告跟上一個警告類似，都是在說位元沒有一致，然而這邊是針對兩個運算元作出的警告，上一個則是針對等號兩邊做出的警告，如下圖所示，operand2&5’b11111的地方，正確寫法應該是要將兩個運算元對齊，變成operand2&32’b11111。



1. SEQ\_NR\_BLKA

這邊是說明在sequential電路時，使用blocking的寫法，我原先認為因為只賦了一個值，因此有沒有blocking沒有影響，但顯然superlint認為這是習慣的問題，此外我發現我沒有做初始化的動作，因此將這需問題一併改善。



最後將這些警告一一解決後，所剩的警告都是與SRAM有關係的，有規定不能更動SRAM的部分，因此忽略到那些警告。

**模擬結果**

在這個部分，我會針對Prog0測資，將每一種type的指令挑選出一或兩個指令搭配波型圖做說明，波型的部分已針對每個stage做分類。

**R type (ADD)**

4c8: 01c383b3 add t2,t2,t3

從這個波型圖可以看到從IF階段pc 為4c8，經過ID將rs1和rs2的值取得到為90f0和90ec，之後傳到EXE，經過判斷沒有forwarding的發生因此operand1 = 90f0，operand2 = 90ec，相加之後得到的結果為000121dc，之後傳到MEM階段，ADD並不會經過DM1，會直接傳到WB將算除來的值存到rd的位置。

**一張含有 螢幕擷取畫面, 行, 平行 的圖片

自動產生的描述**

**一張含有 電子產品, 螢幕擷取畫面, 電子工程, 電路 的圖片

自動產生的描述**

**MUL extension (MUL, MULHSU)**

3bc: 026282b3 mul t0,t0,t1

本次作業新出的MUL extension，就流程來看其實跟ADD很像，差別是在我的CPU設計將MUL乘法器放到MEM階段的一開始，這樣做可以有效降低cycle time，因此可以看到乘法1234\*f的結果在MEM階段才會算出來是0001110c，這個結果同樣不會經過DM1，直接傳到WB寫回register file。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

444: 0262a2b3 mulhsu t0,t0,t1

MULHSU的指令其實跟MUL差不多，差別在於正負號的關係，可以看到operand1為f0f0f0f0，operand2為f0f0f0f0，在我的乘法器設計，會先將這兩個operand擴展成64位元，若是sign則填充operand的sign bit，若是unsign則填充32’b0，因此這個情況下operand1會擴展成fffffffff0f0f0f0，operand2會擴展成00000000f0f0f0f0，爾後相乘。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

**I type (ADDI)**

160: 00440413 addi s0,s0,4

這個指令與ADD唯一不同的是他的ALU input的選擇不一樣，他會有immediate的選擇，在這個狀況immdiate會經過imme\_ext根據不同opcode做immediate的擴展，得到operand2=4，並和operand1=8000做相加，最後得到8004。

一張含有 螢幕擷取畫面, 電子產品, 電腦 的圖片

自動產生的描述

**S type(SW SB)**

在Store比較需要注意的是dm\_w\_en的處理，也就是控制記憶體是否能寫入的訊號，可以比較下方兩個指令的dm\_w\_en，第一個是4’b1111，第二個是4’b0001，這邊使用active high做表示，代表SW可以將4個byte全部寫入，SB只能將一個byte寫入。值得注意的是本次作業使用的sram是word address，因此需要針對地址非4的整數做shift的動作。

108: 00c52023 sw a2,0(a0)

將80fc，4個byte皆存到80f4位置裡面。

一張含有 螢幕擷取畫面, 軟體, 多媒體軟體, 電腦 的圖片

自動產生的描述

898: ffe40c23 sb t5,-8(s0)

將12345678的第0個byte，也就是00000078存到8098位置裡面。

一張含有 螢幕擷取畫面, 電子產品, 軟體, 電腦 的圖片

自動產生的描述

**B type (BEQ)**

這邊使用了beq作為branch的範例，挑出了兩種狀況，branch和不會branch做解釋，這邊branch的判斷，我將它設計在ALU中，針對不同的func可以做出不同的運算。

8cc: 02628c63 beq t0,t1,904

可以看到t0是fffff000，t1是00000ff0，若是相等則branch，顯然的這兩個數字並不相等，在alu\_out\_32也可以很清楚的看到結果為0。

一張含有 螢幕擷取畫面, 電腦 的圖片

自動產生的描述

90c: 00628463 beq t0,t1,914

這個例子可以看到t0是fffff000，t1也是fffff000，這兩個值相等，alu\_out\_32的結果為1，因此跳躍。

一張含有 螢幕擷取畫面, 電腦 的圖片

自動產生的描述

**U type (LUI)**

25c: f0f0f337 lui t1,0xf0f0f

LUI又稱為load upper immediate，將immediate的高20位存入rd中，也就是f0f0f000存入rd中，可以看到f0f0f的部分在ID的階段就已經被擴展成f0f0f000，隨後一直傳下去到WB階段。

一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述

**J type (J)**

110: ff5ff06f j 104 <fill\_block>

顧名思義，直接跳轉到immediate的地方。經過JB unit的運算可以得到跳轉地址為104，此時ID以後的stage需要做flush (nop)的動作，因為前面執行的pc是錯誤的。

一張含有 螢幕擷取畫面, 電子產品, 軟體, 電腦 的圖片

自動產生的描述

NOP

**CSR instruction (RDINSTRETH)**

bdc: c82023f3 rdinstreth t2

看當下的instruction執行了幾個並記錄到rd中。

一張含有 螢幕擷取畫面, 行, 電腦 的圖片

自動產生的描述

以下這個表格可以看到registor在不同狀況會發生的不同事。

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

**Jump**

一張含有 螢幕擷取畫面, 行, 平行 的圖片

自動產生的描述

NOP

**stall**

一張含有 螢幕擷取畫面, 電子產品, 電腦 的圖片

自動產生的描述

NOP

**合成結果**

1. Timing Report

一張含有 文字, 數字, 文件 的圖片

自動產生的描述

一張含有 文字, 文件, 螢幕擷取畫面, 數字 的圖片

自動產生的描述 透過timing report可以看到DC的cycle設為11，slack大於等於0，沒有發生time violation的情況。同時可以針對timing report做分析，乘法器由於運算最為複雜，所以critical path是他非常的合理，原先的設計我是將乘法器放在EXE階段，但是這樣會產生一個問題，要先透過memory取值，造成delay時間拉很長，cycle time會在13附近，因此我將Multiplier直接拉到MEM才做，這樣MEM取值就不會與Multiplier同時做，有效的降低cycle time。

1. Area Report

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述 這邊可以看到合成後的面積為5736035，這次設計沒有特別針對面積去做壓縮，主要針對timing做優化。

1. Post Simulation

在合成階段，DC script裡面的cycle time是設為11，然而在後模擬階段若是將CYCLE設為11會造成模擬錯誤，因此以下CYCLE設為11.5，成功通過Prog0~5。

Prog0: 規格書上的每一個指令在Prog0都會run過一遍，基本上都利用Prog0在debug，是最基本的功能測試。後模擬通過。

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

Prog1: 根據作業要求撰寫一隻main.c檔來完成排序演算法，我使用的是bubble sort，按照題目規定從小排到大，寫了一個function來做bubble sort，兩相鄰的數互相比較，若左邊大於右邊則交換，並歷變每一個數。後模擬通過。

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

Prog2: 不使用MUL指令完成multiplication，此部分由助教提供。後模擬通過。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

Prog3: 作業要求實現最大公因數的計算，算法就是兩數比大小，大的減小的，減到他們相等為止。後模擬通過。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

Prog4 :使用階乘C code測試rdinstret, rdinstreth, rdcycle, rdcycleh，此部分由駐要提供，後模擬通過。

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

Prog5: 使用MUL指令完成multiplication，此部分由助教提供，後模擬通過。

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

**Summary**

本次作業成功設計出一個符合作業規格書規定的5級pipeline CPU，設計分為IF、ID、EXE、MEM、WB五個階段，利用硬體解決了structure hazard, data hazard以及control hazard，同時解決SRAM取值會delay的問題。本次作業的合成使用11ns進行，在後模擬階段使用11.5ns進行後模擬，Prog0~5全數通過模擬結果，RTL模擬亦同，合成出來的面積為5736035。

**Lesson Learning**

在課堂上的部分老師講解了許多system verilog的語法，像是logic可以不用考慮它到底是reg還是wire的問題，大大提升寫程式的方便度，還有一些於法如unique，保持良好習慣，將你心中所想的電路以最詳盡的方式表達出來，system verilog可以做到這一件事。

在完成本次作業後，除了實作的能力大幅上升外，我覺得debug的能力也得到非常大的進步，一開始一個bug總是要抓很久，但慢慢的越來越抓到感覺，跟著pipeline，一步一步trace code，最終完成作業。此外，優化我覺得也是一件很有趣的事，如何在不要大改設計的前提讓cycle time變短或是壓低面積，這次作業我有針對timing做優化，然而我認為還不夠，期盼往後課程能夠學習到更多。做完這次作業後，對硬體設計有了不一樣的理解，也領略到其中困難點和有趣點。