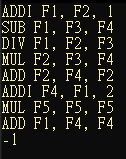
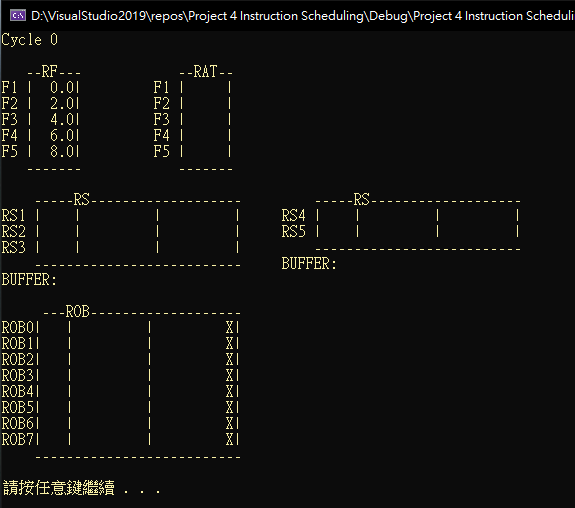
Input ex: 重點在最後要換行輸入-1

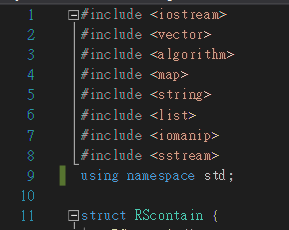


Output ex:

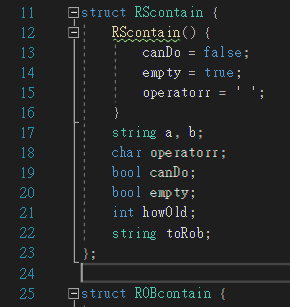


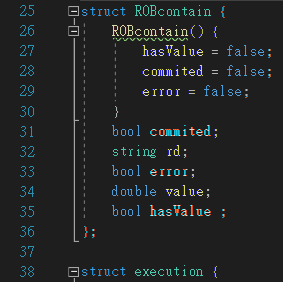
分別是 cycle RF RAT RS BUFFER ROB

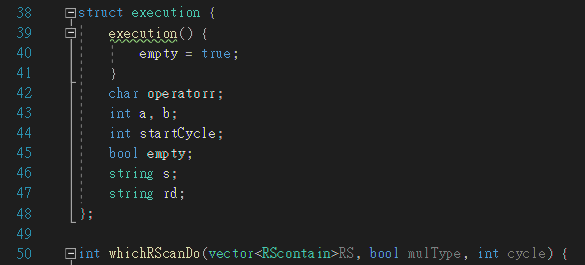
Code解說:

 這個地方引進所有需要的資料

Struct 的部分

另外新增一個RScontain來包含每個RS需要的東西

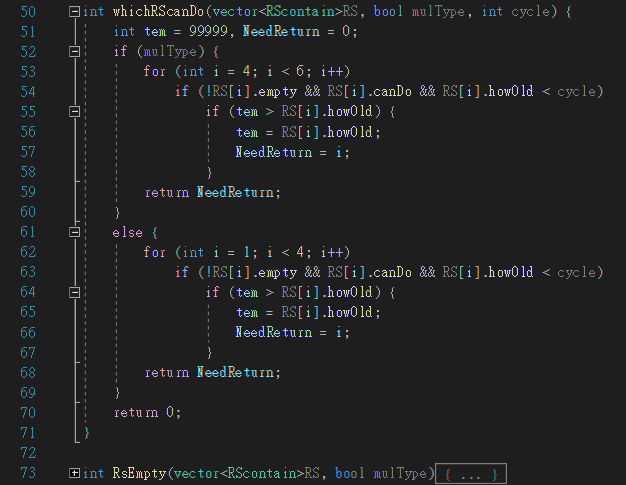
跟RS的同理，包含每個ROB需要的東西



這裡是每個execution所需要的東西

上面三個struct內用到的之後會解釋

Function的部分

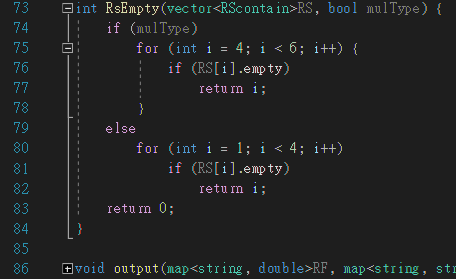


這個用來決定哪一個RS可以被傳到exection去運作

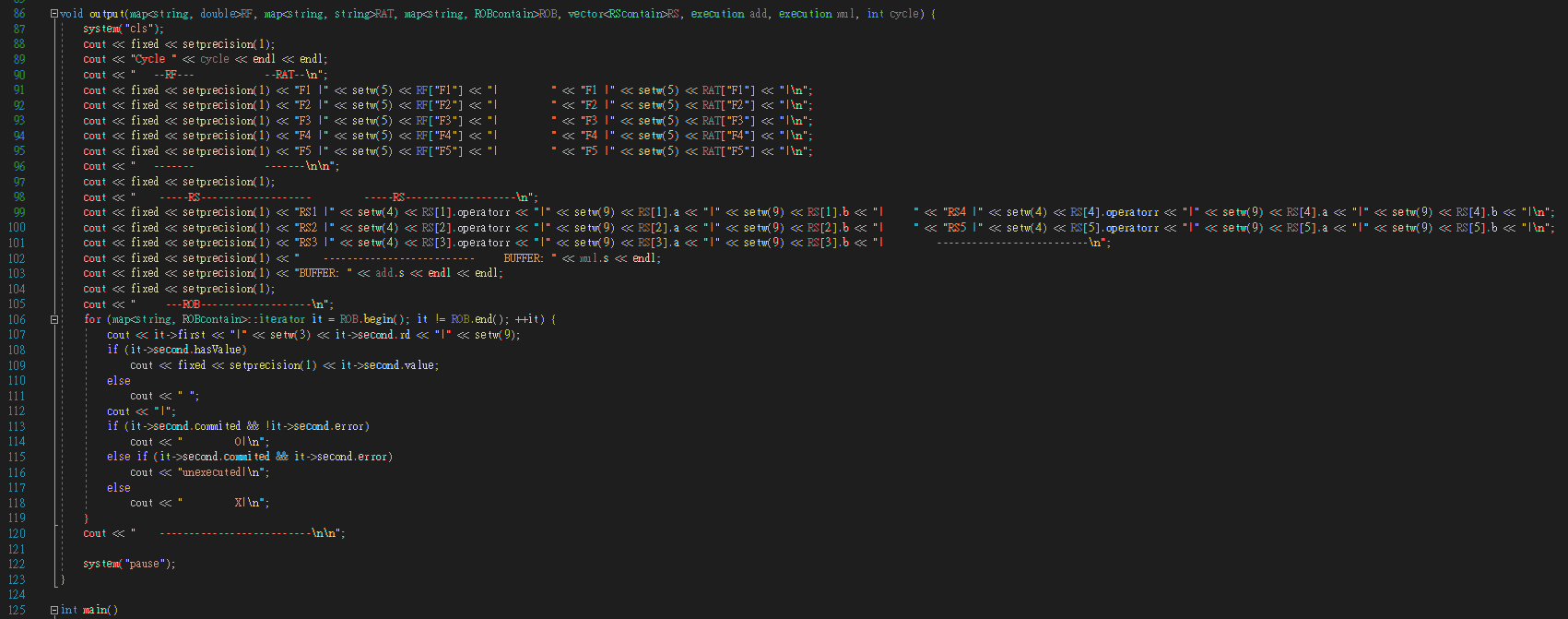
根據現在是指向加法還是乘法器來決定是跑加法的RS部分還是乘法的RS部分

(bool mulType) (RS1~3) (RS4~5)

之後透過每個RS裡面的 cando(bool)代表是不是可以被傳下來做，再透過一個tem來比較誰先傳進來誰比較晚傳進來，並根據較先的傳到execution 另外由於issue的同時不能做dispatch所以還傳入了一個當前cycle跟RS裡面每個被放進去時的cycle做比較，要大於RS的cycle才能dispatch，就不會有同時issue跟dispatch的部分



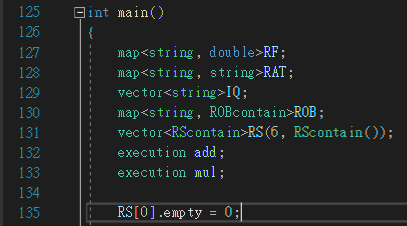
用每個RS裡面的empty來判斷當前這個RS是不是空的，空的代表可以放進去

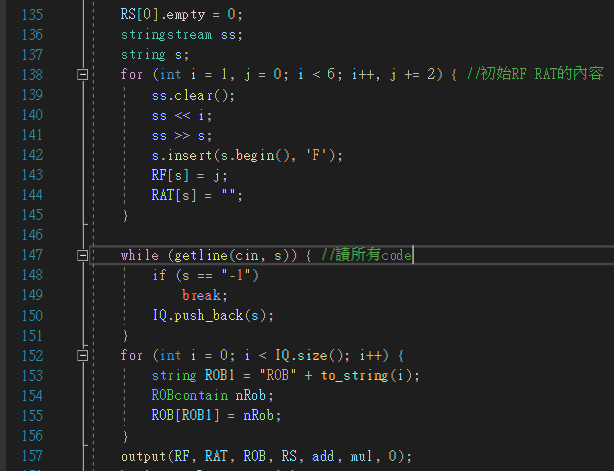


Output function就純粹是拿來輸出像output ex那樣

接下來是 main function得部分

Main function又可以分成

先宣告好所有需要的



138~145是把RF跟RAT的初始值設好

在147~151讀入所有的code

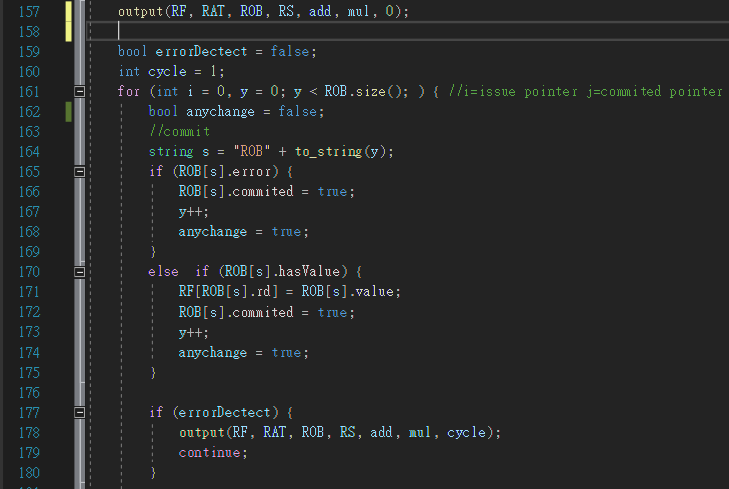
並根據讀入的code Size來把ROB的size設跟code size一樣  
之後先輸出初始的狀態(cycle=0)的時候

程式開始跑cycle從1開始，跑到當所有code都可以從ROB commit回去

i代表當前issue pointer/ y代表當前commited pointer

errorDectect表示之前有沒有divide by 0的問題

並把程式拆成了 commit issue dispatch execution的四個部分



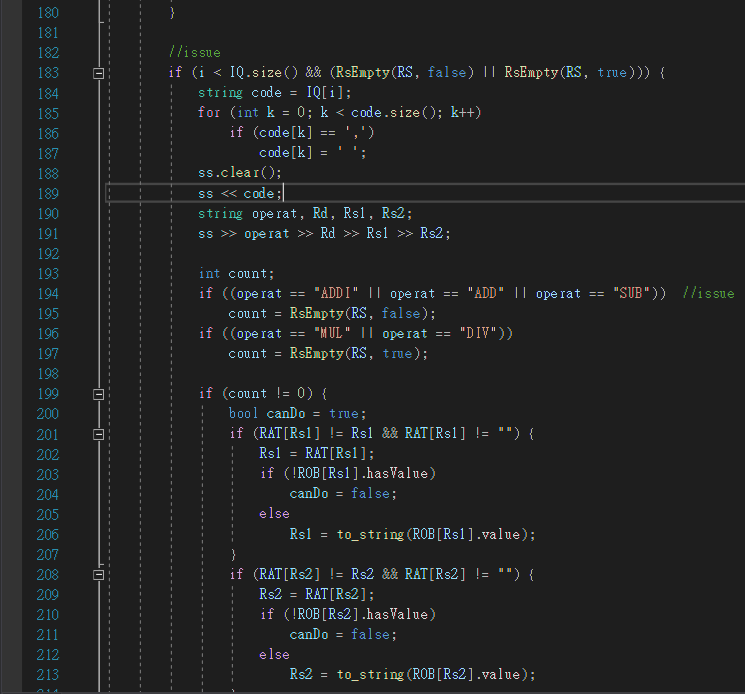
首先是commit的部分 因為她要到下一個cycle才能commit所以透過先commit在做其他的可以達到處理這個問題

並且透過當前這個ROB是不是有error來讓他commit 跟 y++ (165~169)

或者透過當前ROB的 hasValue來判斷是不是已經做完了可以commit了來commit跟y++ (170~175)

之後以前有過error 那之後的都不用execution了 (177-180)

再來是先做issue的部分

透過RsEmpty來判斷有沒有RS是空的才能issue

(183)

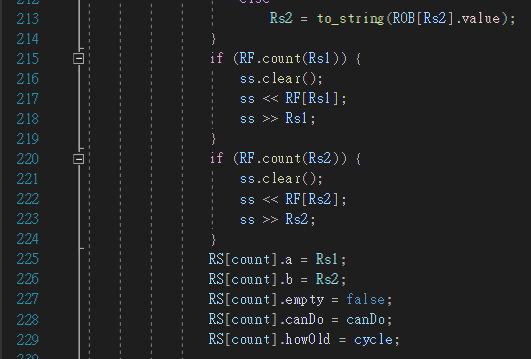
如果有空的位置就透過讀取當前這個issue pointer來讀取下一個code

並且做個初步處理 把逗號轉成空格，並用stringstream來分割出當前的指令跟三個位置  
之後透過當前指令來分析要去哪一個RS討空間(透過RsEmpty的回傳)

如果count不等於0代表在RS[count]這個位置是空的，可以issue (194~199)

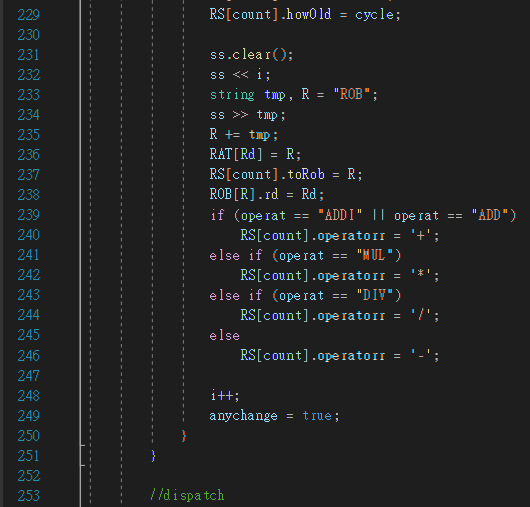
再來透過確認RAT裡是不是有存放Rs1,Rs2,Rd的東西

如果有代表他跟之前的code有dependency 在RAT裡有指向ROB的其中一格，並且代表這個RS當前是不可dispatch的(bool canDo) (200~213)



這邊代表如果RAT沒有，那就要去RF裡面拿取值

並且把當前RS裡面的內容設定好

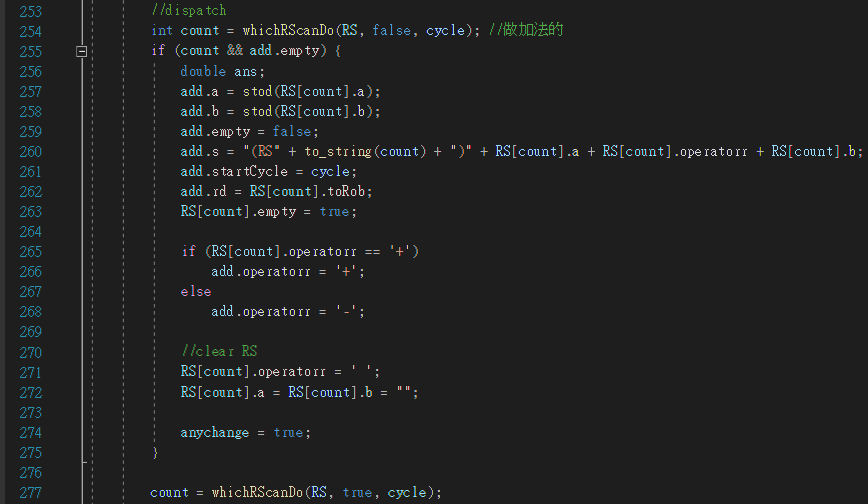


這邊則是處理當放進RS後 RAT跟ROB的值也要一同更改

並且同時代表了有個東西issue進去，代表有東西改變讓anychange=true

好讓之後輸出當前cycle

再來是dispatch的部分

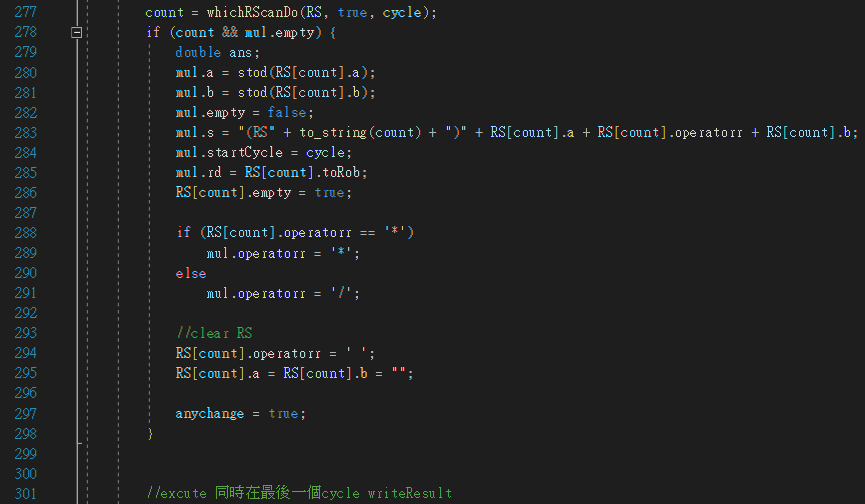
首先是加法的部分

如果add的execution不是空的就不會dispatch而且必須要有一個count代表在RS[count]可以做加法(254-255)  
之後透過一些基本function來轉換string跟double把值放到execution裡面

並讓這個execution在做啥表示到add.s去 之後輸出BUFFER用

另外記錄當前的cycle來準備做cycle延遲，並且把RS dispatch的清空

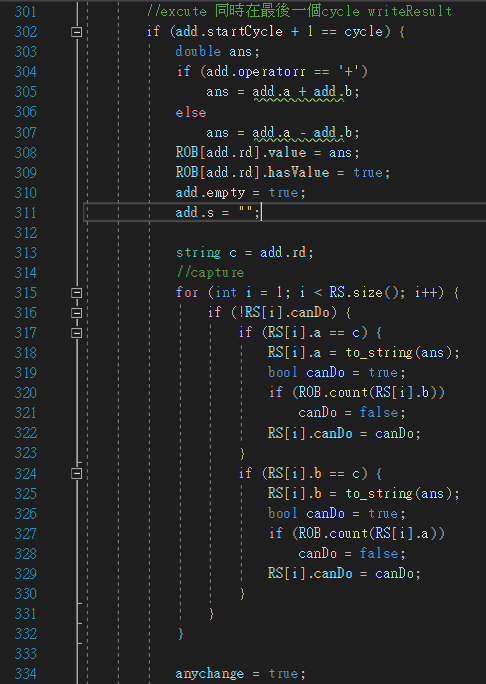
anychange跟之前同理



乘法部分跟加法同理

最後是execute的部分(同時包含writeResult)

首先是加法的execution

只有當跑完cycle延遲才做，並根據裡面的資料來判斷要+還-

之後把值放進ROB去

並讓ROB hasValue=true

之後清空execution的s跟讓add.empty=true

之後把capture做好

透過每個RS是否canDo來看裡面指向的ROB是不是現在指向的ROB

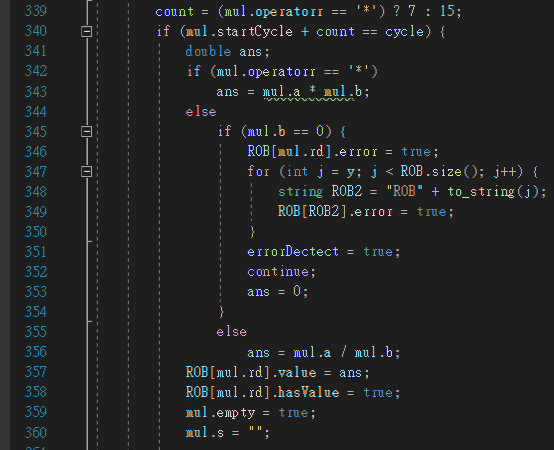
是就改值

不是就不管

最後是如果當前RAT指向同個ROB那就清空他

anychange同理

再來是乘法execution的部分，跟add大同小異

 根據operator的不同有不同的cycle延遲要跑

另外如果divide by zero

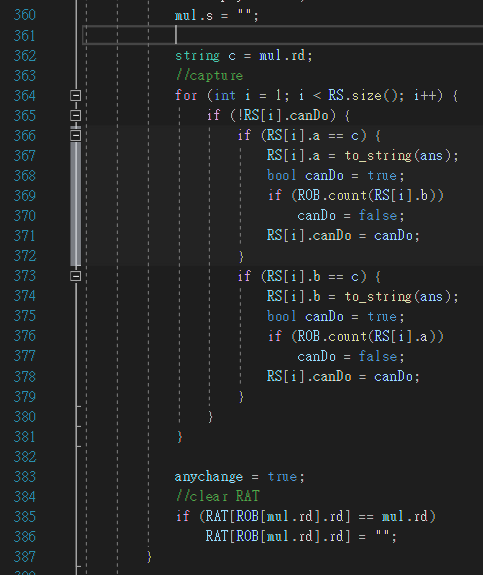
就讓之後的所有ROB.error=

true

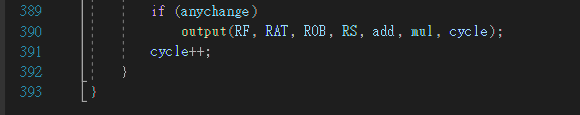
errorDected=

true

讓之後的所有code都不用跑了(前面有提到)  
剩下的東西跟ADD的execution一樣



最後就是 判斷有沒有任何改變來決定要不要跑 output()



並且準備跑下一個cycle