

## 專題二報告

課程主題：計算機結構

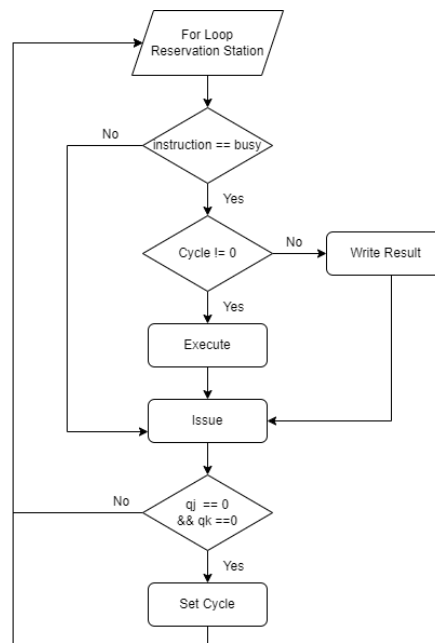
學號：A1095551

姓名：廖怡誠

本次專題的內容為設計 Tomasulo algorithm 的模擬器，根據題目的需求設置 Reservation Station、Register Result status、Cycle time 等參數，並且能夠避免 RAW、WAW、WAR 三種 Hazard。

### 1. 架構設計

Tomasulo algorithm 所使用的架構較多，如 Reservation Station、Register Result Status、Register、Memory 等，此外，還需要管理進入 instruction queue 的指令，所以我分別建立 Reservation Station、Register Result Status、instruction unit 與 Tomasulo 四種 Class，目的是後續在建立陣列時，方便管理與呼叫使用。



圖一、Tomasulo Algorithm 的流程圖

圖一為 Tomasulo Algorithm 的流程圖，1)首先，尋訪 Reservation Station 中的 Adder、Multipliers、Load buffer、Store buffer 是否有 Busy 狀態的指令，2)接著，判斷該指令的執行 cycle 是否已經完成，如果 cycle 為 0，則代表執行完成，執行 Write Result 階段；反之則會執行 Execute 階段，並更新 cycle 的數值，不同類型的指令會執行對應的動作，3)結束 Reservation Station 的尋訪之後，再進入 Issue 階段，從 Instruction queue 中取出 instruction，並設定後續判斷需要使用的參數，

4)最後，會檢查放入 Reservation Station 的指令是否有  $Q_j$  與  $Q_k$  均為 0，代表能夠進入 Execute 階段，因此設定對應的 Cycle 值。

由於優先執行 Issue 之後，會出現同一個 Cycle 內，剛放入 Reservation Station 的指令會進入 Execute，因此需要將 Execute 與 Write Result 的順序提前，且一指令在相同的 cycle 中不會同時進入 Execute 與 Write Result，此外，Write Result 的 Reservation Station 會等到 Issue 結束後才執行初始化，避免在相同的 Cycle 中 Write Result 的硬體空間，馬上被使用。

## 2. 遭遇問題

### 2.1. Store 指令 Write Result 時機

Store 指令進入 Write Result 階段的條件與其他指令不太相同，當  $RS[r].Q_k$  為 0 且執行完成才會進入 Write Result，所以如果根據上述的執行順序(Adder、Multipliers、Load buffer、Store buffer)，Store 指令可能會因為前面的指令 Write Result 使 Store 的  $Q_k$  變成 0，導致在同個 Cycle 會直接進入 Write Result 階段，但由於 CDB 傳送 Data 需要花費 1 個 cycle，因此實際上 Store 指令要到下個 Cycle 才可以進入 Write Result 階段，為了避免發生這個問題，我將執行順序反轉變成 Store buffer、Load buffer、Multipliers、Adder，因為優先執行 Store buffer 就不會提早判斷。

```
for (int i = 3; i >= 0; i--){
    vector<ReservationStation> &reservation_ = *all_reservation_station_[i];
    for (int j = 0; j < reservation_.size(); j++){
        ReservationStation &instruction_ = reservation_[j];
```

圖二、執行順序反轉

### 2.2. Function Unit 管理

原先在書面推導 cycle 時，可以用看得確認每個指令對應到 Reservation Station 與 Register Result Status 的位置，但實際以程式實作時，除了需要管理 instruction queue 的執行順序與其對應到的位置，還要記錄該指令以便彼此之間溝通，例如，記錄 cycle 需要找到對應的指令和 Reservation Station 要在 Register Result Status 中找對應的指令存值，因此，我將 Reservation Station 的每個位置都加入一個編號，方便在 instruction queue 與 Register result status 中找出對應的指令，此部分是書面推導不會遇到的問題，也不太確定實際上的運作會是如何運作。

Reservation Station								
Time	Name	func_unit	Busy	Op	Vj	Vk	Qj	Qk
-1	Add1	-1001	0		0	0	0	0
-1	Add2	-1002	0		0	0	0	0
-1	Add3	-1003	0		0	0	0	0
-1	Mul1	-1004	0		0	0	0	0
-1	Mul2	-1005	0		0	0	0	0
-1	load1	-1006	0		0	0	0	0
-1	load2	-1007	0		0	0	0	0
-1	Store1	-1008	0		0	0	0	0
-1	Store2	-1009	0		0	0	0	0

圖三、設置 Reservation Station 的 func\_unit 數值

### 2.3. Reservation Station 初始化問題

在 Reservation Station 的初始化中，以硬體架構來說，沒有值的時候，該位置的值應該要為 NULL，然而在實作中，由於要避免判斷時，讀取到 NULL 出現問題，且在 C 中無法給一個變數 NULL 的空值，因此只能都將其設值為 0，然而，這樣的處理方式會與判斷進入 Execute 階段雷同，所以我在判斷是否要進入 Execute 階段中，會加入該指令是否於 busy 與其 cycle 是否被設置，將沒有使用到的部分加入 cycle。

```

for (int i = 0; i < 4; i++){
    vector<ReservationStation> &reservation_ = *all_reservation_station_[i];
    for (int j = 0; j < reservation_.size(); j++){
        ReservationStation &instruction_ = reservation_[j];
        // check whether is first time setting cycle time.

        // load need to check qj.
        if(instruction_.op_ == "L.D" ){
            if(instruction_.busy_ && instruction_.qj_ == 0 && instruction_.count_cycle_ == -1){
                instruction_.count_cycle_ = op_to_cycle_number[instruction_.op_];
            }
        }
        // store need to check qk.
        else if(instruction_.op_ == "S.D"){
            if(instruction_.busy_ && instruction_.qj_ == 0 && instruction_.count_cycle_ == -1){
                instruction_.count_cycle_ = op_to_cycle_number[instruction_.op_];
            }
        }
        // FP
        else{
            if(instruction_.busy_ && instruction_.qj_ == 0 && instruction_.qk_ == 0 && instruction_.count_cycle_ == -1){
                instruction_.count_cycle_ = op_to_cycle_number[instruction_.op_];
            }
        }
    }
}

```

圖四、加入額外判斷參數

### 3. 執行結果

Sample1 result

Cycle 57			
Instruction	Issue	Comp	Result
L.D F6, 8(R2)	1	3	4
L.D F2, 40(R3)	2	4	5
MUL.D F0, F2, F4	3	15	16
SUB.D F8, F6, F2	4	7	8
DIV.D F10, F0, F6	5	56	57
ADD.D F6, F8, F2	6	10	11

Sample2 result

Cycle 50			
Instruction	Issue	Comp	Result
L.D F6, 8(R2)	1	3	4
L.D F2, 40(R3)	2	4	5
ADD.D F0, F2, F6	3	7	8
SUB.D F8, F6, F2	4	7	8
MUL.D F10, F0, F8	5	18	19
S.D F10, 0(R3)	6	7	20
DIV.D F8, F0, F2	7	48	49
S.D F8, 8(R3)	8	9	50

Sample3 result

Cycle 49			
Instruction	Issue	Comp	Result
L.D F6, 8(R2)	1	3	4
L.D F2, 40(R3)	2	4	5
ADD.D F4, F2, F6	3	7	8
DIV.D F8, F0, F4	4	48	49
SUB.D F6, F6, F2	5	7	8
MUL.D F10, F6, F4	6	18	19
S.D F10, 40(R3)	7	8	20

Sample4 result

Cycle 63			
Instruction	Issue	Comp	Result
L.D F6, 8(R2)	1	3	4
L.D F2, 40(R3)	2	4	5
ADD.D F4, F2, F6	3	7	8
DIV.D F8, F0, F4	4	48	49
MUL.D F6, F8, F4	5	59	60
SUB.D F10, F2, F4	6	10	11
SUB.D F14, F8, F4	7	51	52
SUB.D F10, F6, F4	9	62	63
ADD.D F12, F6, F4	12	62	63

Sample5 result

Cycle 71			
Instruction	Issue	Comp	Result
L.D F6, 8(R2)	1	3	4
L.D F2, 40(R3)	2	4	5
MUL.D F4, F2, F6	3	15	16
DIV.D F8, F0, F4	4	56	57
MUL.D F6, F6, F2	17	27	28
DIV.D F10, F6, F4	29	69	70
S.D F10, 40(R3)	30	31	71

Sample6 result

Cycle 88			
Instruction	Issue	Comp	Result
L.D F0, 0(R1)	1	3	4
L.D F2, 0(R0)	2	4	5
DIV.D F4, F0, F2	3	45	46
DIV.D F8, F6, F4	4	86	87
SUB.D F8, F0, F2	5	7	8
DIV.D F10, F8, F4	47	87	88

Sample7 result

Cycle 172			
Instruction	Issue	Comp	Result
L.D F2, 8(R1)	1	3	4
L.D F4, 16(R0)	2	4	5
ADD.D F4, F2, F4	3	7	8
DIV.D F4, F4, F4	4	48	49
DIV.D F4, F4, F4	5	89	90
DIV.D F4, F4, F4	50	130	131
DIV.D F4, F4, F4	91	171	172

## 4. 心得

在這次的實作中，實作了 Tomasulo Algorithm，了解到使用軟體模擬硬體的實際架構與程序仍然有一定程度的困難，需要考慮到執行操作通常是 1 個 cycle 為單位、或是硬體的線路傳資料使用軟體只能使用指標和標籤來代替，這些問題使我在實作的當下十分糾結，會好奇實際的情況會如何運作，如果用軟體模擬用甚麼樣的寫法能夠更接近實際的硬體運作，但兩者還是有差別，沒辦法實作到完美。除此之外，參照投影片上的 Issue、Execute、Write Result 三個流程的說明也十分有挑戰性，感覺像是一個人要實作 Pipeline 的複雜度，一步一步跟著說明實作，並發現其中的 Store 與 Load 不會有到 Register Result Status 確認 Qi 的步驟，因為通常 Store 與 Load 是到 Memory 拿資料，但是 Register Result Status 紀錄的 Double 型態的 Register，如果直接取值，會獲取並存在的位置。