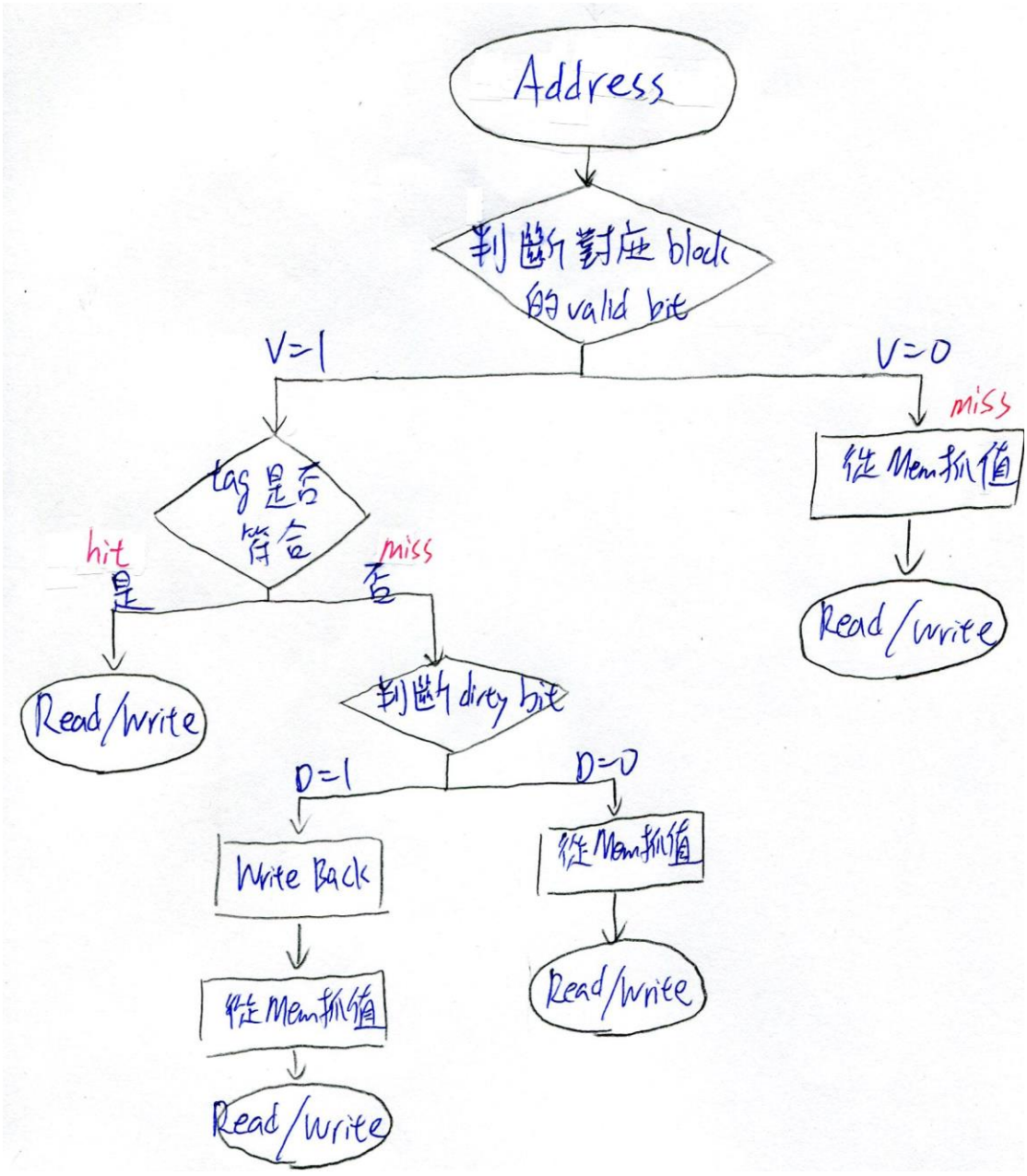
計算機結構 HW4

姓名：莊育權
學號：B03901142
系級：電機三

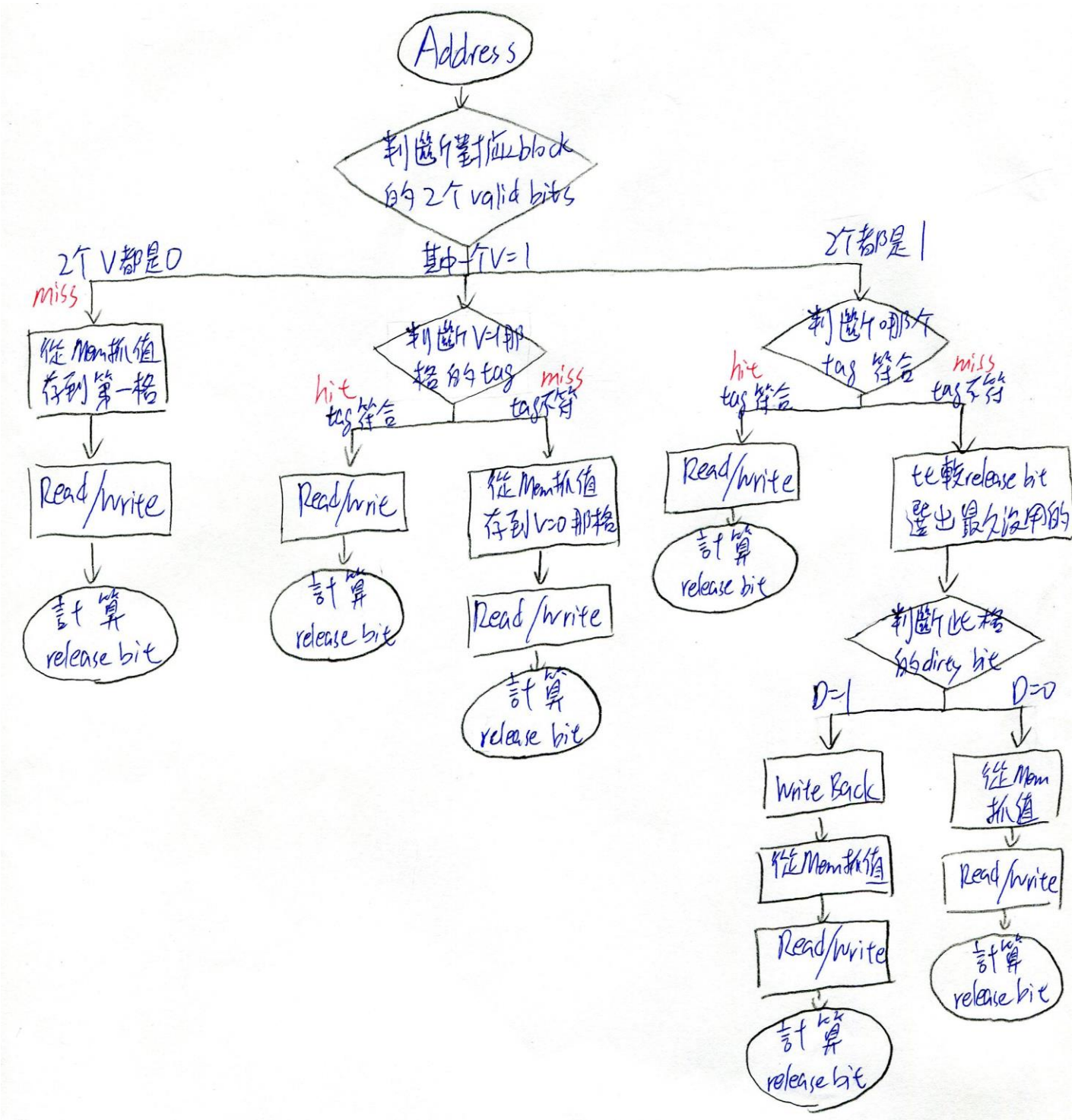
I、作業報告

一、 設計結構

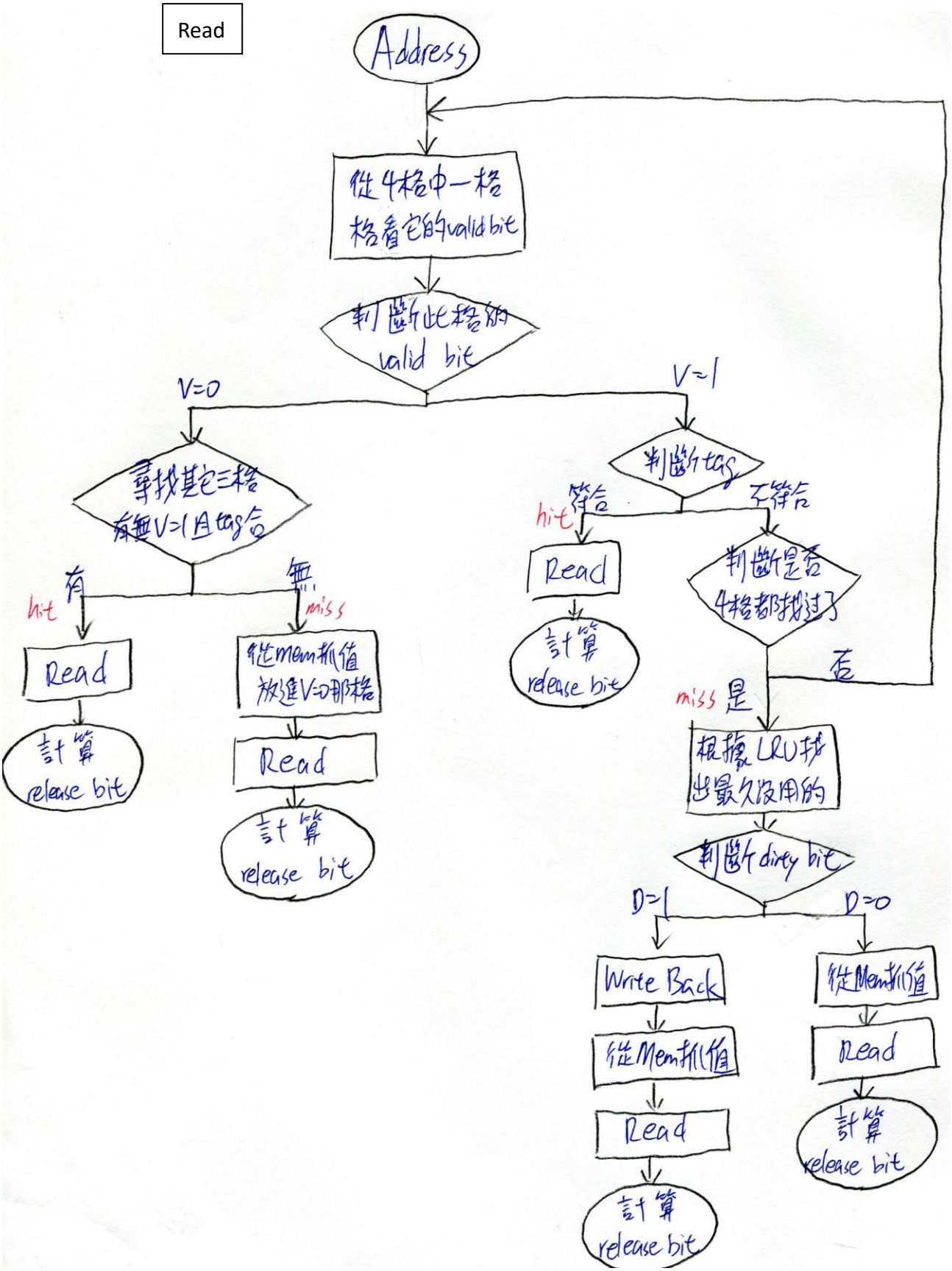
1. Direct-mapped

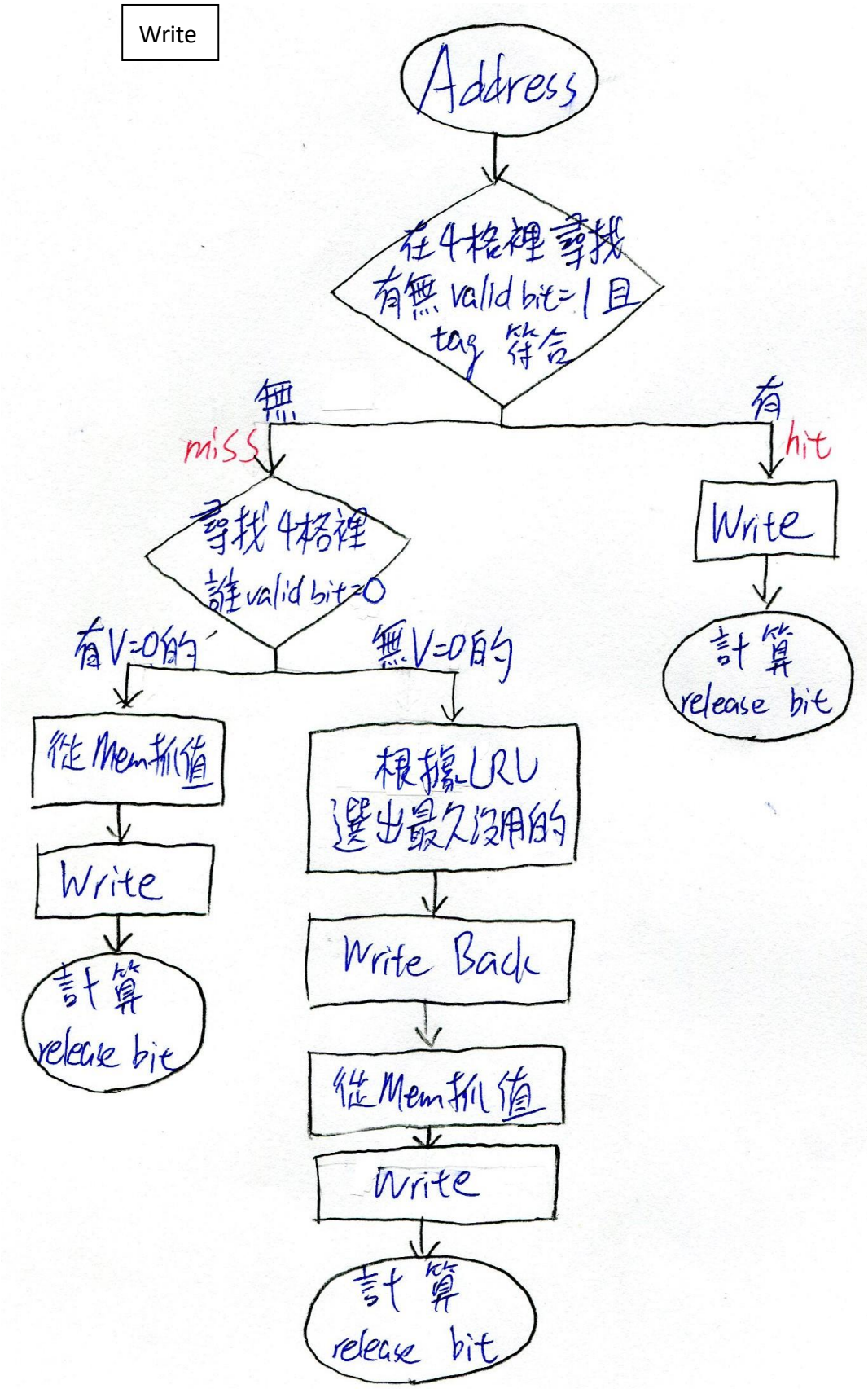


2. two-way

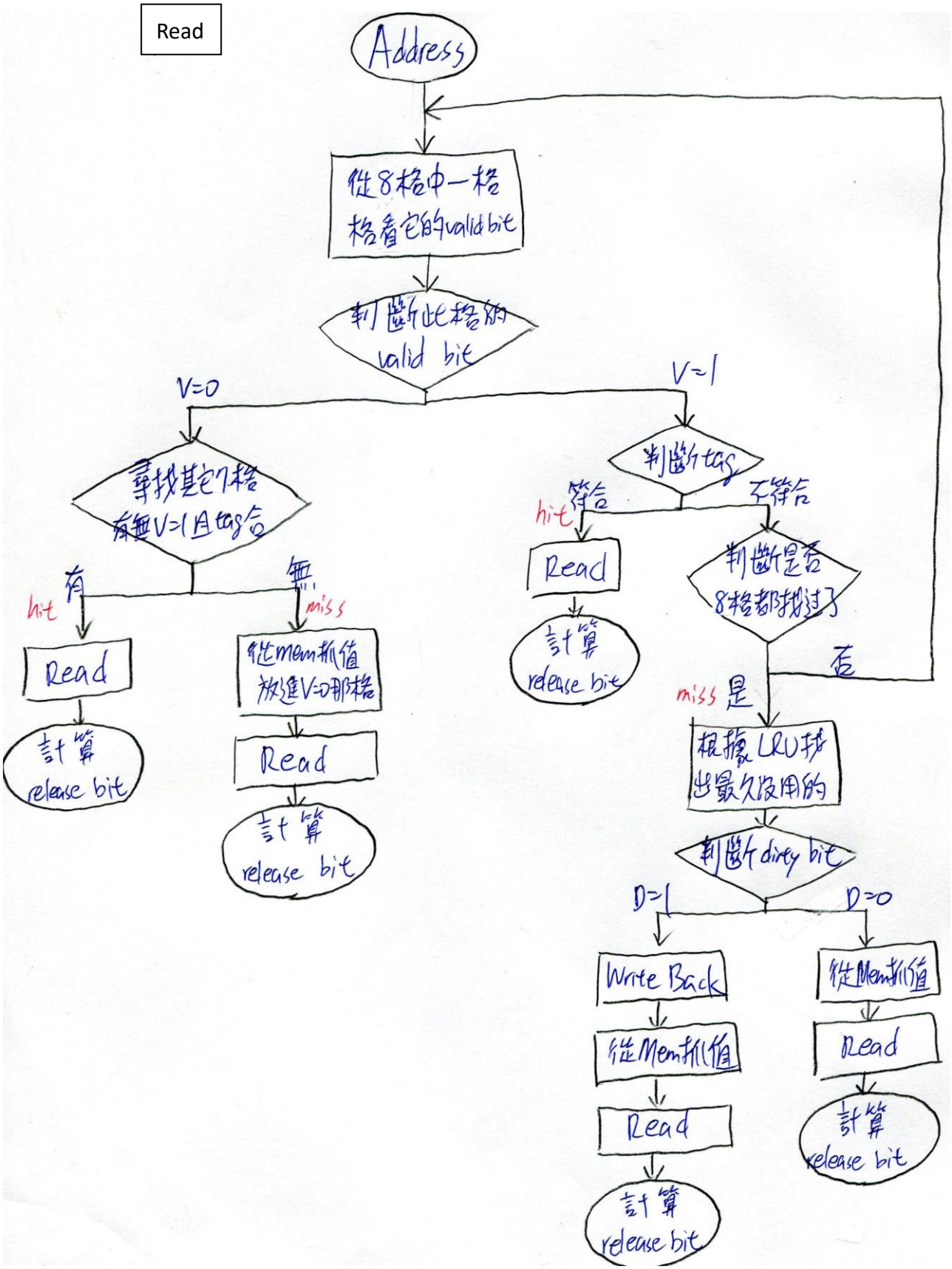

Address
判斷4對應block的2个valid bits
2个V都是0
其中一个V=1
2个都是1
miss
從Mem抓值存到第一格
Read/Write
計算release bit
判斷V=1那格的tag
hit
tag符合
miss
tag不符
Read/Write
計算release bit
從Mem抓值存到V=0那格
Read/Write
計算release bit
判斷4哪个tag符合
hit
tag符合
miss
tag不符
Read/Write
計算release bit
te較release bit選出最久沒用的
判斷此格的dirty bit
D=1
D=0
Write Back
從Mem抓值
Read/Write
計算release bit
從Mem抓值
Read/Write
計算release bit

3. four-way

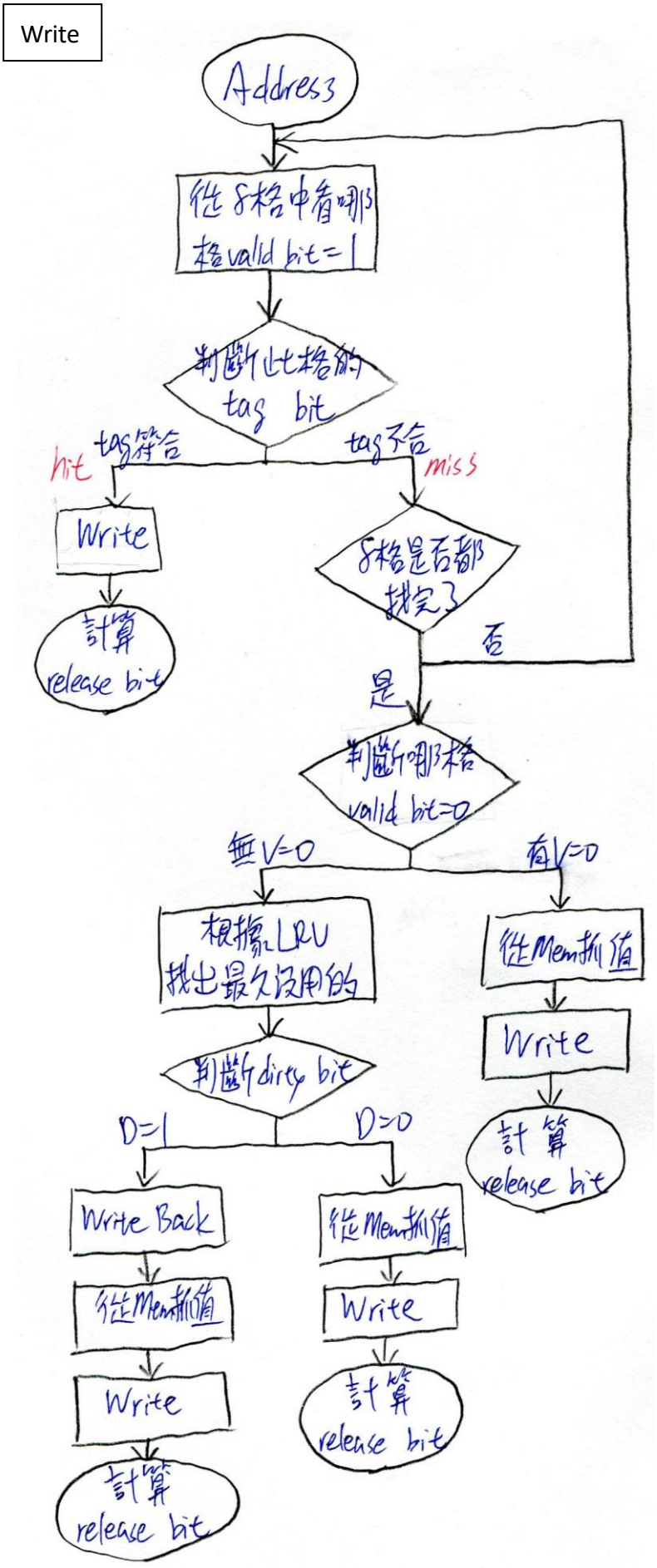

Read
Address
從4格中一格
格看它的valid bit
判斷此格的
valid bit
V=0
V=1
尋找其它三格
有無V=1且tag合
判斷tag
hit
符合
不符合
有
無
hit
miss
Read
計算
release bit
從mem抓值
放進V=0那格
Read
計算
release bit
Read
計算
release bit
判斷是否
4格都找過了
否
miss
是
根據LRU找
出最久沒用的
判斷dirty bit
D=1
D=0
Write Back
從Mem抓值
Read
計算
release bit
從Mem抓值
Read
計算
release bit


Write
Address
在4格裡尋找
有無 valid bit=1 且
tag 符合
無
miss
有
hit
尋找4格裡
誰valid bit=0
Write
計算
release bit
有V=0的
無V=0的
從Mem抓值
Write
計算
release bit
根據LRU
選出最久沒用的
Write Back
從Mem抓值
Write
計算
release bit

4. Fully-associative



Write


Address
從S格中看哪格valid bit=1
判斷此格的tag bit
hit
tag符合
tag不合
miss
Write
計算release bit
S格是否都找完了
否
是
判斷哪格valid bit=0
無V=0
有V=0
根據LRU找出最久沒用的
從Mem抓值
判斷dirty bit
Write
計算release bit
D=1
D=0
Write Back
從Mem抓值
從Mem抓值
Write
Write
計算release bit
計算release bit

5. Release bit 的用法

在 2-way、4-way 與 fully-associative 中，當在同一 block 中，裡面有一個 set 被使用時(Read/Write)，自己的 release bit 設成 0，其餘的 release bit 都減 1。

當同一 block 中全部都被填滿，而且需要把其中一個 set 替換掉時，基於 LRU(Least Recently Used)原則，檢查每一個 set 的 release bit，release bit 最小的代表很久沒被用到，因此替換掉那一個 set。

二、 模擬結果

1. Table for Hit rate of quicksort/mergesort

	Direct-mapped way_nymber=1	2-way way_nember=2	4-way way_number=4	Fully associative way_number=8
Quicksort	88.99%	90.84%	89.15%	91%
Mergesort	80.5%	83.78%	83.27%	86.84%

2. Discuss why there is difference of hit rate b/w different structures

當 cache 的 block 裡面只有一個 set(Direct-mapped)時，很容易發生 data conflict 而造成 miss 的情況發生，當 associativity 上升時，一個 block 擁有更多 set，會使得 data miss 的情況下降。因此，通常 associativity 上升時，hit rate 也會上升。

3. Is hit rate keeps going up as the way_number goes up

從 Mergesort 可以看出來，當 associativity 漸漸上升的時候，hit rate 有很明顯的改善，原因就如上一題所述。

然而對於 Quicksort 而言，associativity 漸漸上升，雖然 hit rate 也有上升，但其實上升的幅度並不高，到最後幾乎都差不多。因為 Quicksort 本身對於空間的占用量並不高，因為它只需要額外多一個空間讓兩個數值去做交換，如下一題解釋，因此當 associativity 上升時，原本的 set 已經足夠讓 Quicksort 去做排了，因此對於它的 hit rate 影響不會太大。

三、 問題與討論

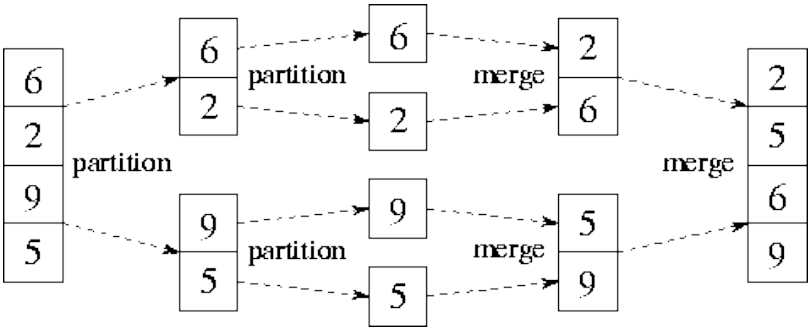
1. What the computational complexity of quicksort/mergesort?

   Quicksort: O(nlogn) / Mergesort: O(nlogn)

2. Which algorithm has the higher hit rate? Why?

   (1)Quicksort 有較高的 hit rate

   (2)由於 **Mergesort 的演算法**，是將一個序列分成兩段，然後一直不斷分下去，直到只剩下 1 個 data 或是 2 個 data，然後將他們排序比大小，比完之後再回到上一層去跟另外一組再做一次排序比大小，如下圖。



   但這對於 cache 而言較沒有 temporal locality，因為當最下面那層排序完，cache 必須要一直去抓另外一層不屬於現在這層的序列去做 mergesort，因此較沒有時序性，也易造成 miss。而對於 **Quicksort 的演算法**，也是分成兩段，但在分的時候是根據 pivot 去做分類，因此當最後一層排列完的序列，就是已經是按照整個序列的大小順序排列好，並不像 Mergesort 即使在這層排列好，還要回到上一層跟另外一層去重新做一次排列，因此 Quicksort 對於 cache 的使用較有時序性，因此 miss 較少。

   另一方面關於**額外空間的使用**，Mergesort 需要額外多開好幾個空間才能讓裡面的序列去做排序，而 Quicksort 只需要額外一個空間讓裡面的數值可以 swap 即可，因此 Mergesort 對於 Write miss 會比較大，因此 hit rate 比較低。

3. Hit/Miss

   一開始寫完 code 的時候，發現自己的 hit rate 都跟助教給的數據差很多。事後才發覺自己對於一個指令的 miss 和 hit 定義有誤。

一開始認為當 miss 時(miss+1)，需要從 memory 抓值回來放到 cache，之後再做一次 Read/Write，這邊這個重新 Read/Write 的行動，我又算一次 hit(hit+1)。事後，才發現一個指令對於 miss 和 hit 應該就是一體兩面的，當指令一開始是 miss 就 miss，一開始是 hit 就 hit，不會出現先 miss 後 hit，也就是一筆 data 只有 hit 或 miss 的可能，而且是由第一次 miss 或 hit 決定。

4. The way for release bit

最後結果發現 hit rate 在 way_number=4 的 mergesort 無法到達助教所說的正負 2%之內，有跟同學互相討論過後，覺得是因為當每個 set 的 release bit 一樣的時候，跟你決定要取哪一個 set 被替代有關。我的版本是取最後一個，有試過取第一個，但效果更差。**希望助教在這方面可以從寬，因為在 spec 裡面沒有特別規定是要怎麼如何使用 release bit。**

四、 心得

最近真的很能體會到教授為什麼會想把計算機結構結合數位系統設計。在寫這幾次作業的過程中，由於為了寫出一個完整的 CPU 或 Cache，你必須把這些的原理還有結構都要讀得很透徹，你才寫得出來，藉由這樣其實也幫助我自己複習教授上課所教的內容，而且又可以發現自己其實哪裡的觀念是模糊的，像是上面所提到 hit/miss 和 LRU 的問題。

這次作業是用軟體的方式實現 cache 的運作，但在打 code 的時候，其實發現如果要把軟體真的用 Verilog 硬體的方式做出來的時候，會發覺自己的方法會有些是不可行的，像是上面所提到 release bit 的方法，在硬體中並無法給 release bit 一個無限的 bit，應該會先給定 release bit 到底最多幾 bit，因此在下次作業要實作硬體的時候，就需要想另外一種方法達到 LRU 的目的，藉由這次作業也了解到軟體跟硬體之間實現的方法與邏輯其實是很不同的。