Student Confirmation of Academic Integrity Policy
I have read and understood the Academic Integrity Policy. I acknowledge that all submitted work is my own and acknowledge the consequences of violating this policy.

Student Name: _____
Student ID: _____
Signature: _____
Date: _____

1. Include screenshots of your simulation results as evidence of correctness.

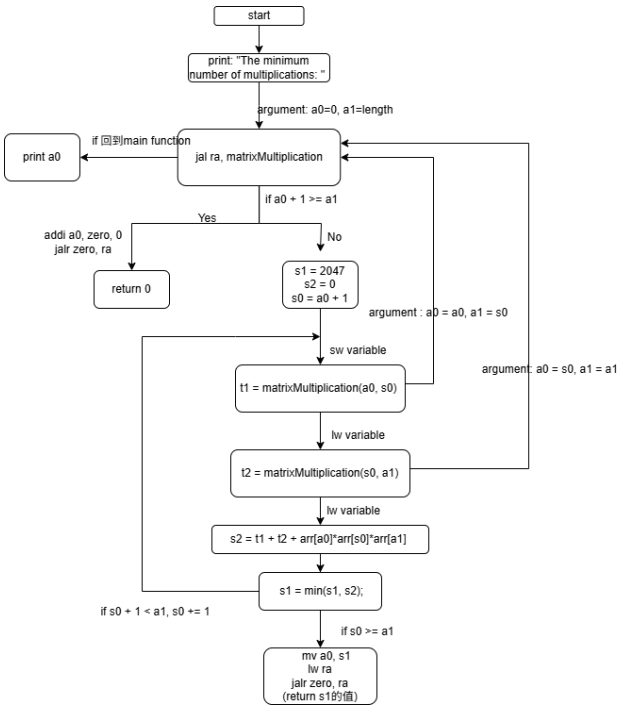
   Test1:

   Console
   ```
   The minimum number of multiplications: 166
   ```

   Test2:

   Console
   ```
   The minimum number of multiplications: 394
   ```

2. Provide a flowchart of your implementation.



3. Explain your algorithm clearly and concisely.

   此程式主要實作 Matrix Chain Multiplication 的最小乘法次數:

   開頭先印出說明字串 "The minimum number of multiplications: "，再用 register 把 length 的值和 arr 的 base address 記下來。接著，設好 argument a0 = 0, a1 = length - 1 並跳至 matrixMultiplication label。

   matrixMultiplication label 的功能是判定是否要進入 loop label，如果 a0+1<a1，說明沒有可切的矩陣，回傳 0。否則，就把之後會用到的 register 的值(a0:(i), a1(j), s0(k), s1(res), s2(curr))設好，並把這些 redister 的值

(上述和 ra，在這裡用 variable 代稱)都存入 memory。

接著我們往下進入 loop label，先跳到 matrixMultiplication label(argument :a0 = a0, a1 = s0)，再重複上述步驟(matrixMultiplication label 後的 code)，如此形成遞迴。遞迴完後回到剛才的呼叫處的下一行，並把結果存入 memory。接著我們讀出 variable(這樣可以避免 register 的值是錯誤)後，再跳到 matrixMultiplication label(argument :a0 = s0, a1 = a1)，其過程和上面提到的跳回一致。

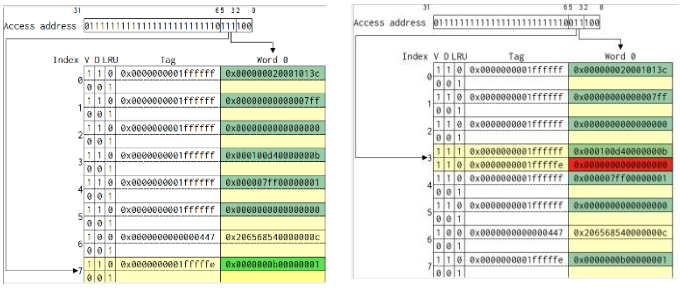
然後，我們再讀出 variable 和上一次跳回的結果(這樣可以避免 register 內的值是錯誤)，並加總左 + 右子問題 + 分割左右的成本。分割左右的成本為 arr[i] * arr[k] * arr[j](計算方法如下，將 i、k、j 轉為 offset → 加到 t0 = arr base 得到位址 → 讀值 → 相乘)，加總後的總成本就是 s2 的值，如果 s2 < s1 則更新 s1 的值。最後，如果 s0 + 1< a2，就 s0 += 1，並回到 loop label 的起始 address，否則，脫離 loop 回到 ra(呼叫時的位置的下一行)，並回傳 s1 的值。

等遞迴函數結果，回到_start label 內時，被 return 的值就是我們要的最小成本，這個程式的邏輯就是反覆遞迴切割 arr，最終找到所有可能成本的最小值。

4. question

(a) (3%) Provide a case of cache insertion at an index (a set) where exactly one way is already occupied. Be sure to explain how the LRU bits of both ways in that set are updated.

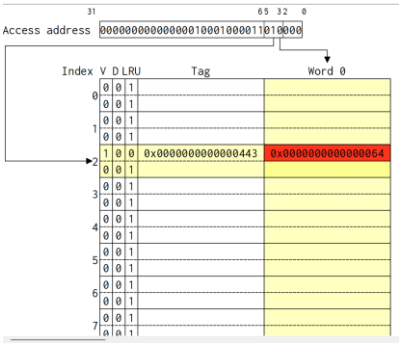
Instruction: 10108: 00a12023 sw x10 0 x2

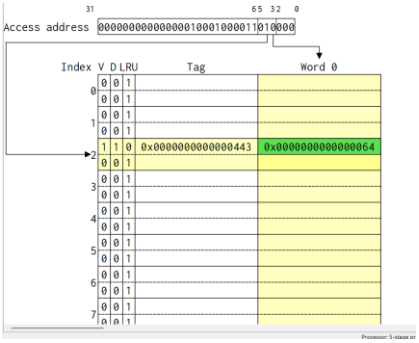


Set3 在左圖時 way0 有放東西，此時 way0 的 LRU = 0, way1 的 LRU = 1，這是因為 way1 比較久沒被 referenced 到。到右圖時，載入新的 block，因為 way1 的 LRU = 1，所以 block 放到 way1 內。同時 way0 的 LRU 被更新為 1，way1 的 LRU 被更新為 0，這是因為 way1 已經剛被 referenced 過了，現在是 way0 比較久沒被 referenced。

(b) (3%) Provide a case of a cache hit where the block becomes dirty as a result (i.e., it was not dirty before the hit).

Instruction: 100c0: 0072a023 sw x7 0 x5

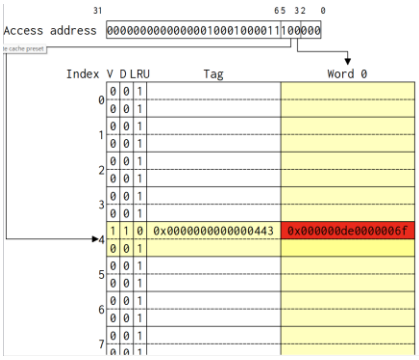


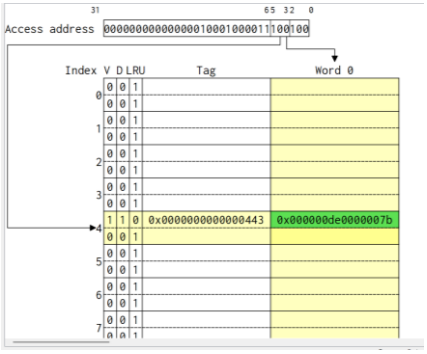


X5 存的 memory address 在這個指令前，已經被 lw 過。因此 Dirty bit 原本為 0，hit 後 dirty bit 變成 1。

(c) (3%) Provide a case of a cache block replacement where a write-back to memory is required.

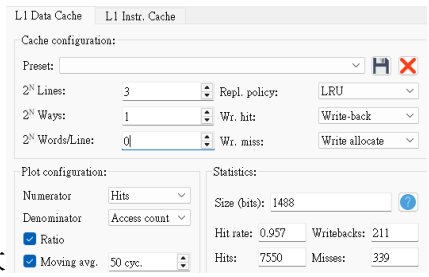
Instruction: 100cc: 01c3a023 sw x28 0 x7

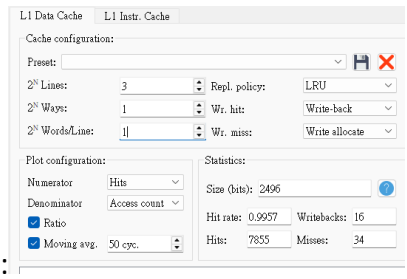




原本 dirty bit = 1 的 block，在被 replaced 後，這個 block 會被 write-back to memory.

(d) (5%) Improve the hit rate by designing your own cache configuration. You may adjust parameters such as associativity or cache size. Include the following: i. A screenshot of your custom cache design and configuration. ii. A screenshot showing the improved hit rate. iii. A brief explanation of why your configuration improves the hit rate.

原本 

customized: 

我修改的是 block size，從 1word 改成 2words，這同時也增加了 cache size。而之所以 hit rate 會上升主因有二。其一，我用了 local spacility 的特性，當程式存取某一個記憶體位置時，它很可能接下來會存取附近的記憶體位址。其二，cache

size 增加，使得有更多的 data 會被留在 cache 中。透過上述，我成功讓 hit rate 從原本的 95.7% 提升至 99.57%。