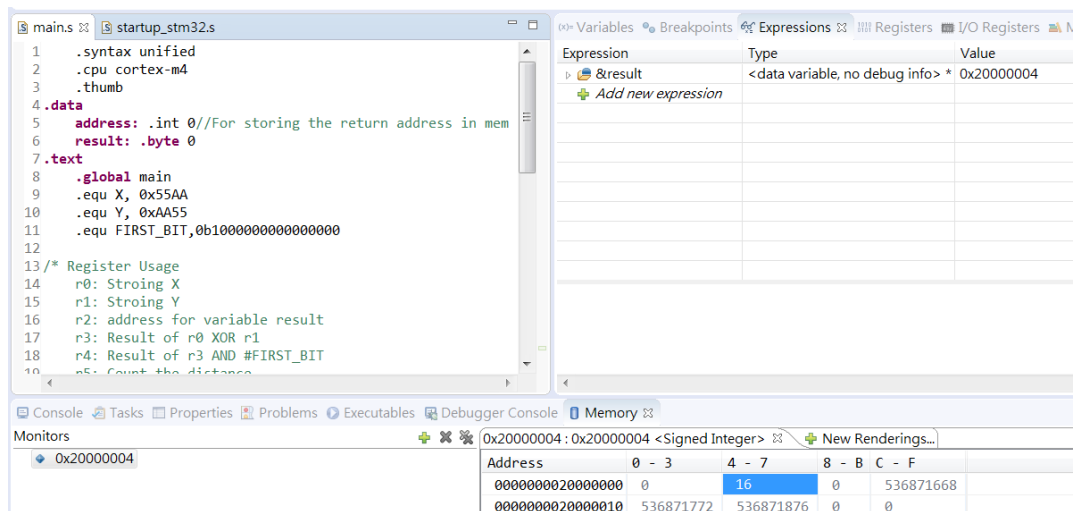


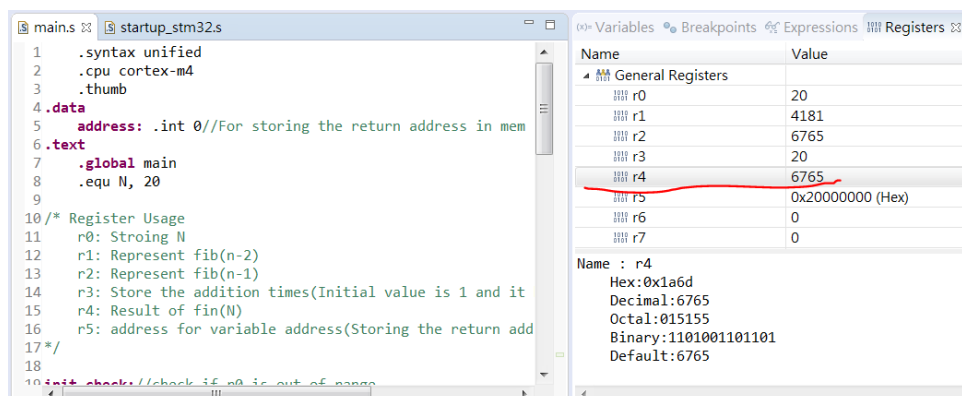
微處理機 Lab2 Report

- 一、 實驗名稱：ARM Assembly I
- 二、 實驗目的：熟悉基本 ARMv7 組合語言語法的使用。
- 三、 實驗步驟：
 1. Hamming distance。
 2. Fibonacci serial。
 3. 簡易算數與基本記憶體指令操作。
- 四、 實驗結果與分析：

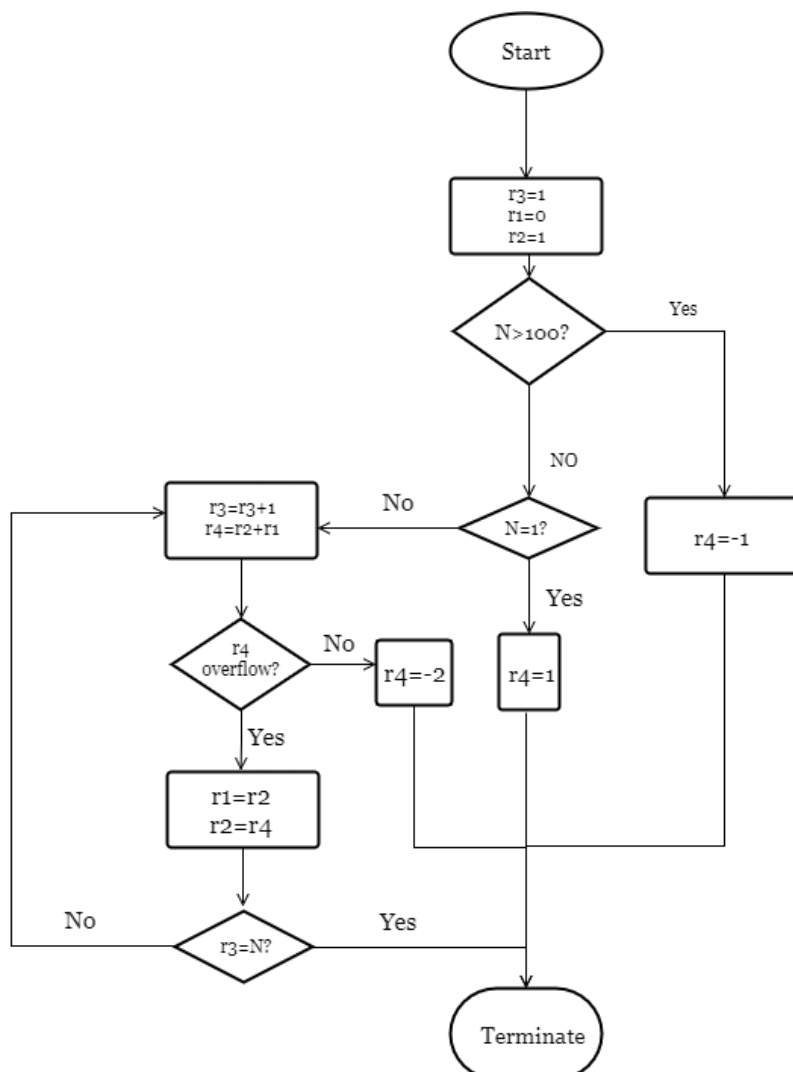
1. Hamming distance：



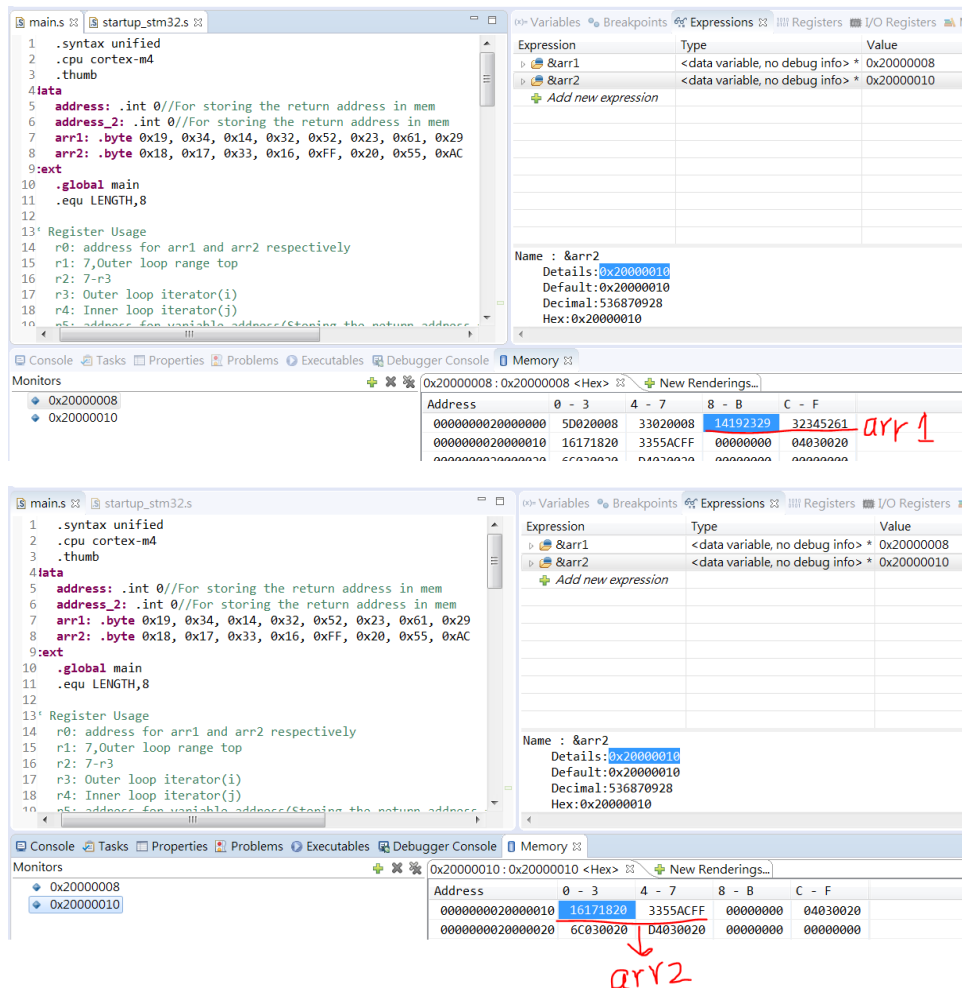
2. Fibonacci serial：



我們這題用的是迴圈的做法，在這種做法中有個規則是當 $N \geq 2$ 時，需要算出 $f(N)$ 的總加法次數是 $N-1$ 次，這就是拿來控制要 branch 幾次的依據。最初先檢查 N 有沒有在給定範圍內，沒有的話直接讓 $r4=1$ 後結束執行；接下來的算法就是先拿兩個暫存器存 $\text{fib}(0)$ 及 $\text{fib}(1)$ 的值，依序把每項的費波那契函數值算出來，當每算一個新的費波那契數值後，先檢查有無 overflow(利用 condition filed)，有的話也是直接讓 $r4=-2$ 後結束執行，再更新那兩個暫存器的值，更新成當下的費波那契函數值及前一項的費波那契數值，接著檢查加法次數是否已到頂，沒有的話就繼續跳入迴圈，有的話就跳出。最後 $\text{fib}(N)$ 的數值可在 $r4$ 裡面看到。



3. Bubble sort :



這題我們是照著維基上的 bubble sort 來寫的，把 C 語言一行一行翻成組合語言，因為有兩層迴圈，所以控制跳出那邊比較難寫。這題還有一個陷阱是因為資料都是 8 位元的，一開始在使用 ldrb 的時候用成 signed 的版本，導致怎麼排出來都怪怪，最後是發現原來 signed 版本的會把那一個 byte 當成 signed integer 處理，所以 load 到 register 裡面就會變成負數，改成使用 unsigned 的版本後解決此問題了。

4. Enabling FPU (Floating Point Unit) and Floating Point Manipulation

i. Q3.4.1.1：如果 enable_fpu 留空，程式會停在哪裡？為什麼？

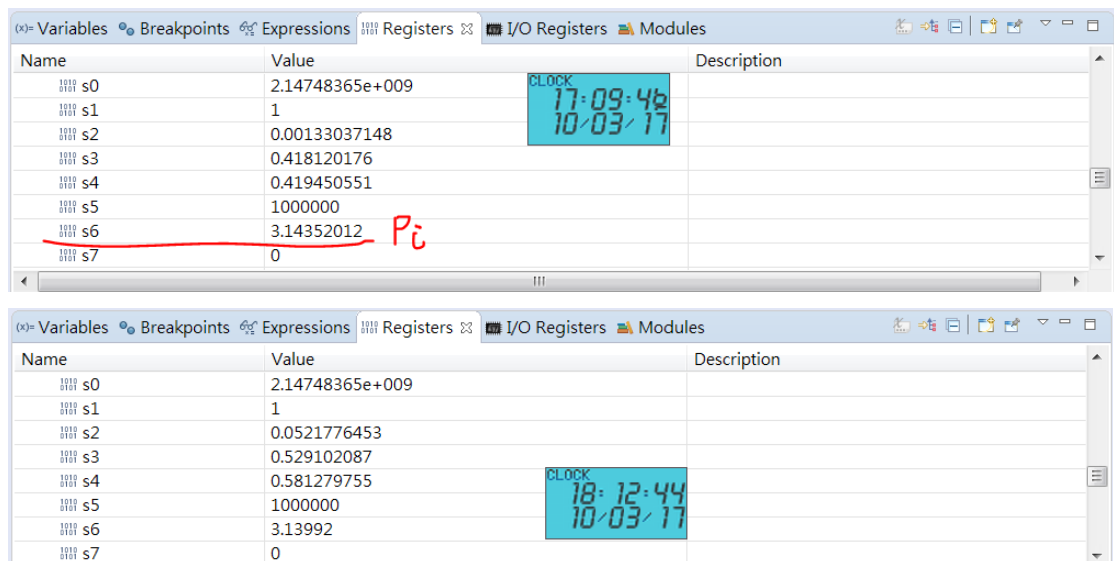
程式會停在 main.s 的 vldr.f32 s0,[r0]這一行(第一個利用 float point 的指令)，更精確來說程式會停在 startup_stm32 的第 96 行，會在那邊形成一個無限迴圈。那造成這樣的原因，

ii. Q3.4.1.2：為什麼需要將 U32 轉成 F32 格式再相加？如果想直接 load 一個值代表 20 到 s2 中不需轉換就能運算，應該將 z 修改成多少才能得到相同答案？

- 因為 vldr 這個指令會直接來源暫存器內的東西直接存在 s2 暫存器，但問題就是整數與小數的二進位表示法明顯不同，s2 裡面存的不是浮點數“20”的二進位表示式，所以要經過轉換，這樣 s2 裡面存的才是正確浮點數“20”的二進位表示法。
- 由前一題可以知道 vldr 指令是直接把來源暫存器的內容搬到目的暫存器，如果要想不經轉換就可運算，只要將來源暫存器內容改存成浮點數“20”的二進位表示式就可以了。浮點數“20”的二進位表示式為 01000001101000000000000000000000，將來源暫存器內存這個數值，vldr 做完之後就不用再轉換即可直接運算。

5. Estimation of Pi

i. 三次結果：



Name	Value	Description
s0	2.14748365e+009	
s1	1	
s2	0.00133037148	
s3	0.418120176	
s4	0.419450551	
s5	1000000	
s6	3.14352012	
s7	0	

Name	Value	Description
s0	2.14748365e+009	
s1	1	
s2	0.0521776453	
s3	0.529102087	
s4	0.581279755	
s5	1000000	
s6	3.13992	
s7	0	

Name	Value	Description
s1	1	
s2	0.0991835073	
s3	0.886285126	
s4	0.985468626	
s5	1000000	
s6	3.14118409	
s7	0	
s8	0	

ii. 遇到的問題與解決辦法

- 開啟 RNG 功能，一開始完全不知道怎麼存取 RNG 這個暫存器做修改，看了很久才在助教給的 template 裡面的 set_flag 中看到相似的做法，才發現原來是要用類似把記憶體裡面的值讀出來做修改後存回記憶體的方式來開啟 RNG 這個暫存器。
- 浮點數的比較問題，因為要比較浮點數誰大誰小，很直覺的就使用 vcmp 這個指令了，但用了之後發現，估算出來的 π 值都接近 4，懷疑就是在比大小這邊錯了，導致判斷在圓內的點過多。之後發現原來 vcmp 改的是 fpscr 這個暫存器，並不是一般常用的 xPSR 暫存器，所以 vcmp 後面用的帶有 condition field 的算數指令其實都不知道真正的 condition flag 是什麼。這個問題的解決方法就是每次做完 vcmp 指令後就把 fpscr 暫存器的內容搬到 xPSR 暫存器內，這樣接下來帶有 condition field 的算數指令就知道真正的 condition flag 是什麼了，也可以正常判斷點有無在圓內了。

五、心得討論與應用聯想：

這次的作業期實並不難，但我們仍然在熟悉語法的使用，像是第一題我們當初並不知道為何 blne 無法使用，是上網看懂了 IT block 後才了解到他的用法，其次讓我們最困惱的就是第四題了，當初由於 RNG 開啟位置錯誤所以一直找不到，在一番努力研究下，發現要將 RNG 往後放，我們總算可以生成

亂數，但卻又因忘記 VCMP 不更新 APSR 而再次卡關，在逐行檢查下，才想起要將 ZNCV 丟入 APSR，這才將本次作業完成。