期末 Project

114 學年度第 1 學期

老師：朱守禮　老師

組別:18

學生：

11327144 莊有隆

11327137 林宇宸

一、背景

目的：

熟悉 ARM 組合語言語法、大數據量記憶體存取（Frame Buffer）與數學演算法實作。

核心任務：

在樹莓派模擬器的 Console 模式下，透過組合語言計算並繪製動態的 Julia Set 分形圖形。

二、方法

工具：

Notepad：撰寫程式。
WinSCP：下載專案檔案。
QEMU 模擬器：使用 Code::Blocks 執行組合語言檔及檢索暫存器等

1. 程式說明

name.s ：印出組別及三名組員的英文名字。

id.s ：讀取三名成員的學號，並根據讀入的控制指令判斷是否輸出隨後依序輸出學號，並將學號加總後輸出。

main.s ：負責控制動畫循環、開啟/dev/fb0 設備節點、並將 frame 陣列寫入顯示記憶體。

drawJuliasSet.s：主要用來計算 julia set 動畫的每個 pixel 顏色，並將結果存入 frame 二維陣列，最後由 main.c 寫入 Frame Buffer 顯示。

2. 設計重點說明

應避免使用 r0 至 r3 暫存器來存取欲存放的資料，因為 printf 及 scanf 等函式常使用此四個暫存器運算，故存放內容極易發生變化。

- name.s:

  需分配三段記憶體空間存放名字字串內容，並靈活運用 bl printf 指令。首先使用 ldr 將 r0 暫存器載入欲輸出內容，並執行 printf。bl 指令中，b 表示進入 branch (分支)，l 表示將 PC (執行下一道指令的位址) 存進 LR (函式返回地址)，以便返回主程式繼續執行，避免位址錯亂。

- id.s :

  除了輸出基本字串格式外，則需要使用 bl scanf 指令進行資料讀入。但在讀入前需做好前置工作：準備 r0 暫存器作為讀入資料格式，準備 r1 暫存器作為存放欲讀入資料的記憶體段位址。

- enter_student_number 讀入段：

  共 3 筆資料，r11 暫存器用於計算次數。若次數大於 3，則跳出 label load_number。後續進行 scanf 前置動作，並使用 strle r4, [r7], #4 post-index 將整數資料存進去，並位移 4 位元組 (byte) 偏移量，以存取另外 2 筆資料，持續迴圈。

- 輸出 student_number 段：

  再將 r7 暫存器內的三筆資料取出，進行輸出及總和運算。

- Id.s 特殊要求：

  mov r12, sp：備份堆疊指標 (SP) 至 r12。
  rsbs sp, lr, pc：透過逆減法擾亂 SP 記憶體。
  mov sp, r12：將備份資料放回 SP。

  運用 mov 將 SP 暫存器內資料直接備份至 r12 暫存器，便無需擔心後續指令是否會擾亂 SP，只需最後將 r12 備份資料放回 SP 即可。

- main.c :

  程式主要透過呼叫 name()、id()與 drawJuliaSet()三個函式來整合組員資料與 julia Set 動畫功能。name()會印出組員姓名及組別名稱，id() 則讀入三個學號並將其寫入全域陣列 id_store，同時計算總和存入 id_sum_val。由於資料已存入全域變數，main.c 僅需載入所需資料即可進行顯示，而無須再次計算。

  使用者輸入學號後，程式即時計算總和，並透過 printf 將完整組員資訊與總和分行印出。為避免影響 printf 與 scanf 所使用的 r0–r3 暫存器，資料皆透過全域變數或其他暫存器存取，確保計算與顯示正確。

  動畫部分，程式直接操作/dev/fb0 的 Frame Buffer，將 drawJuliaSet() 計算出的畫面逐幀寫入硬體，並使用雙層迴圈控制 cY 變化，實現 Julia Set 動態畫面。透過按鍵控制動畫開始與結束，提供使用者直覺操作介面。

- drawJuliaSet.s:

  核心功能是計算 JuliaSet，並將每個 pixel 的顏色寫入 frame 陣列中。程式使用兩層迴圈分別控制 x 與 y 座標，對應畫面每個像素的位置。

  數值運算方面，將畫面座標轉換為 JuliaSet 的複數值 zx 與 zy，透過整數運算方式（如乘以 1000 再除以常數)來模擬浮點運算，兼顧精度與效能。__aeabi_idiv 函式被用於整數除法，確保運算正確。

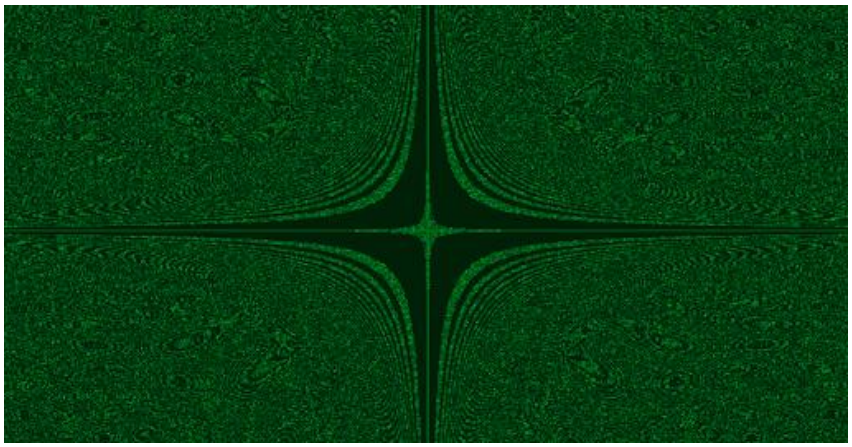
  在函式開頭，透過 stmfd sp!, {r4-r11, lr} 保存暫存器內容，並在結尾使用 ldmfd sp!, {r4-r11, pc} 恢復，確保函式呼叫返回後暫存器值不受影響。整體設計兼顧暫存器管理、計算精度與記憶體存取，能正確產生 Julia Set 動態畫面。

- drawJuliaSet.s 特殊要求：

  指定: adds lr, sp, pc
  解決: mov r0, lr 備份再存回

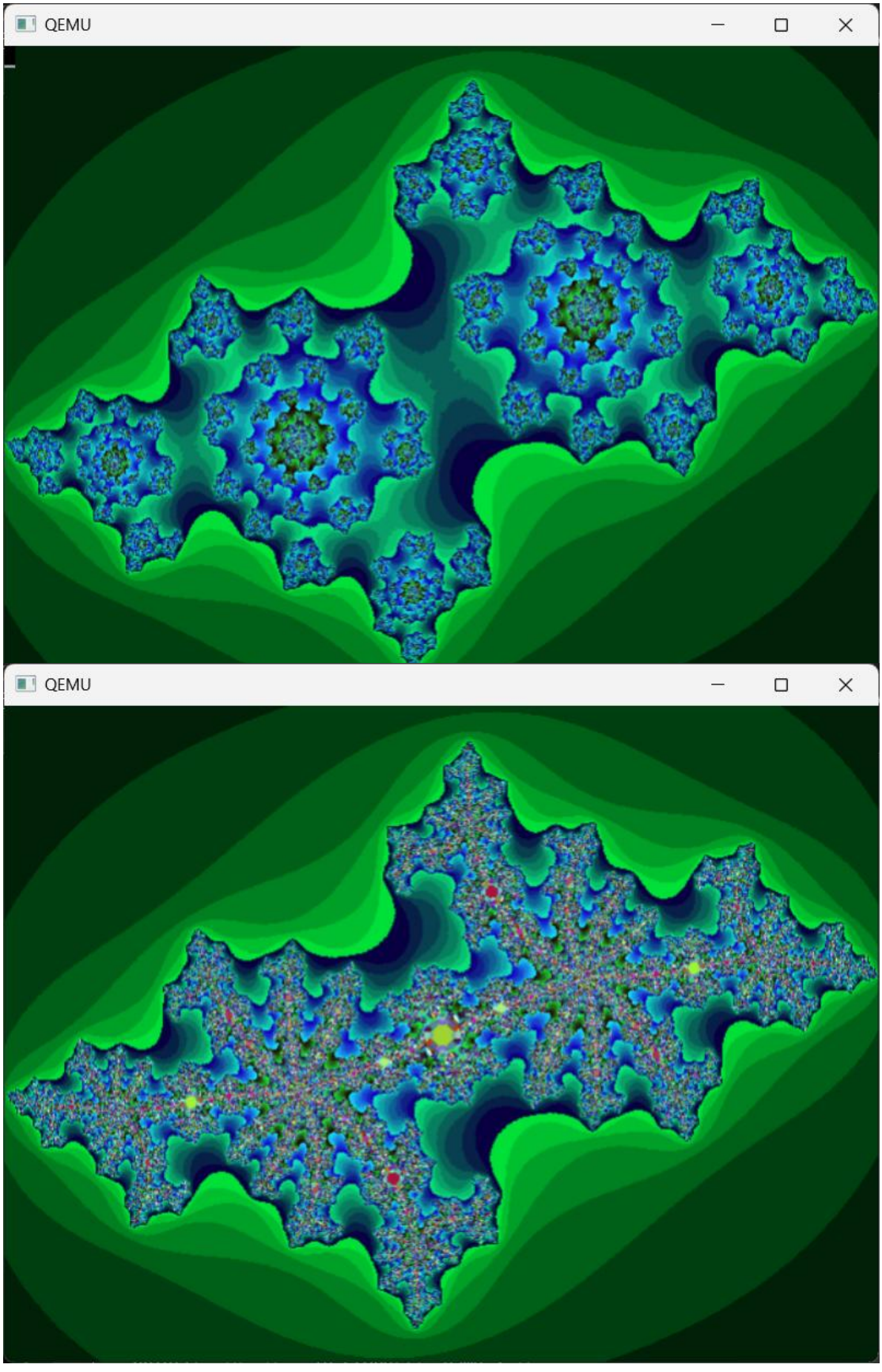
- 除法實作：

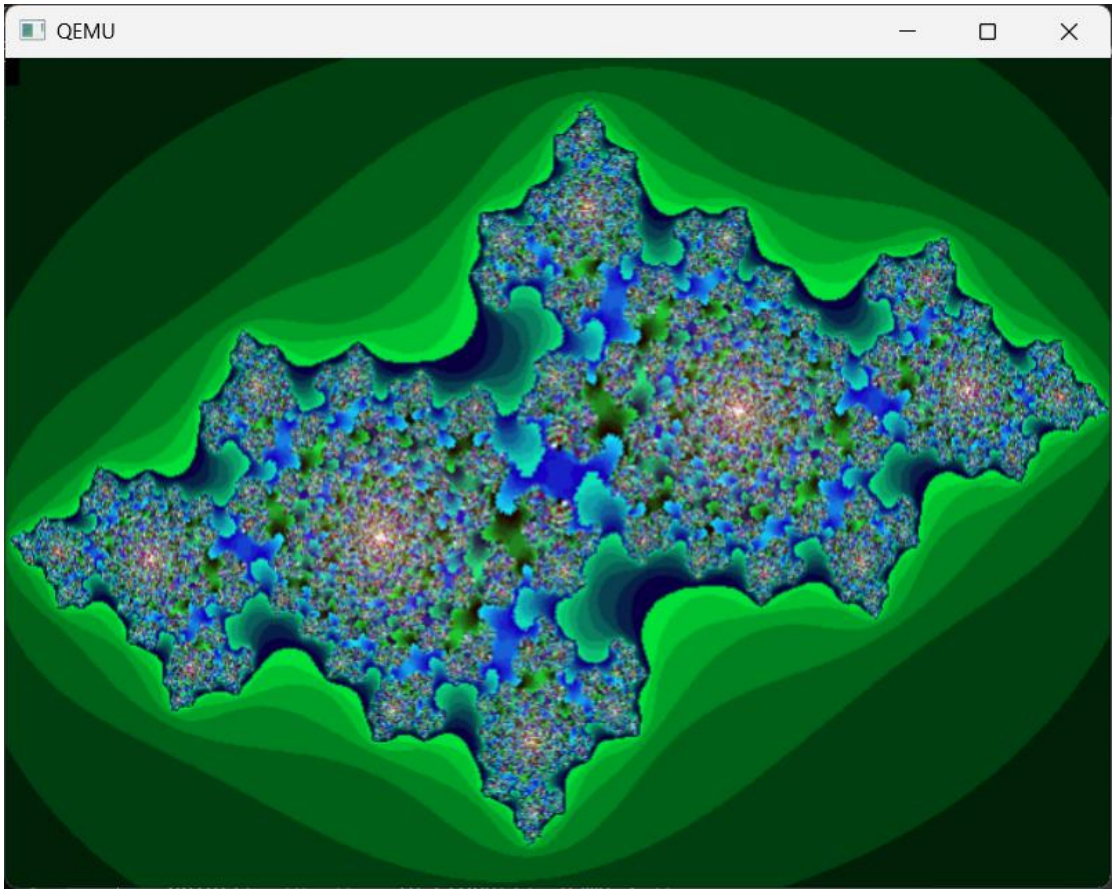
  專題中使用了除法函式庫(__aeabi_idiv)來完成整數除法運算，以符合計算 JuliaSet 所需的精度與效率。在使用除法函式庫時，仍需注意負數處理與運算順序，確保結果正確，避免畫面出現錯誤或破碎的像素。透過觀察結果，理解到即便使用函式庫，組合語言中暫存器保存與參數傳遞也很關鍵，錯誤的暫存器管理會導致除法結果不如預期，就如同下面的圖片。

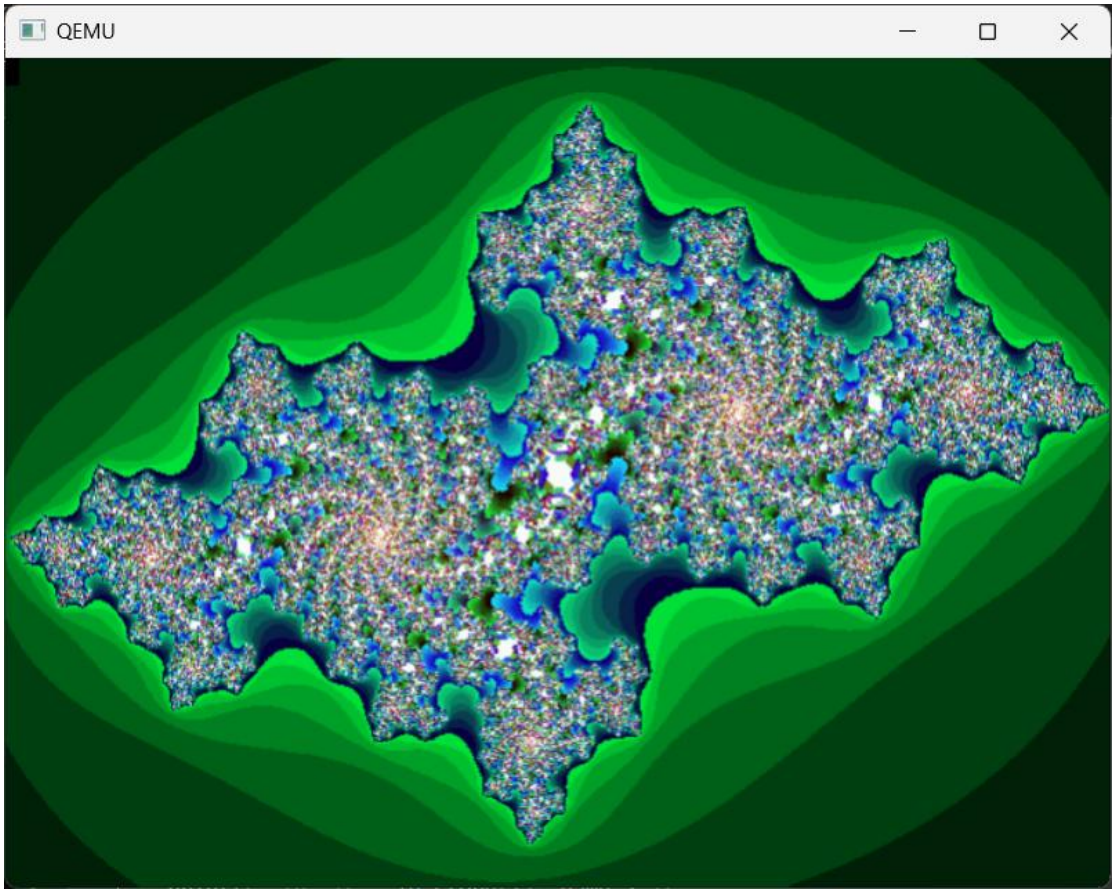


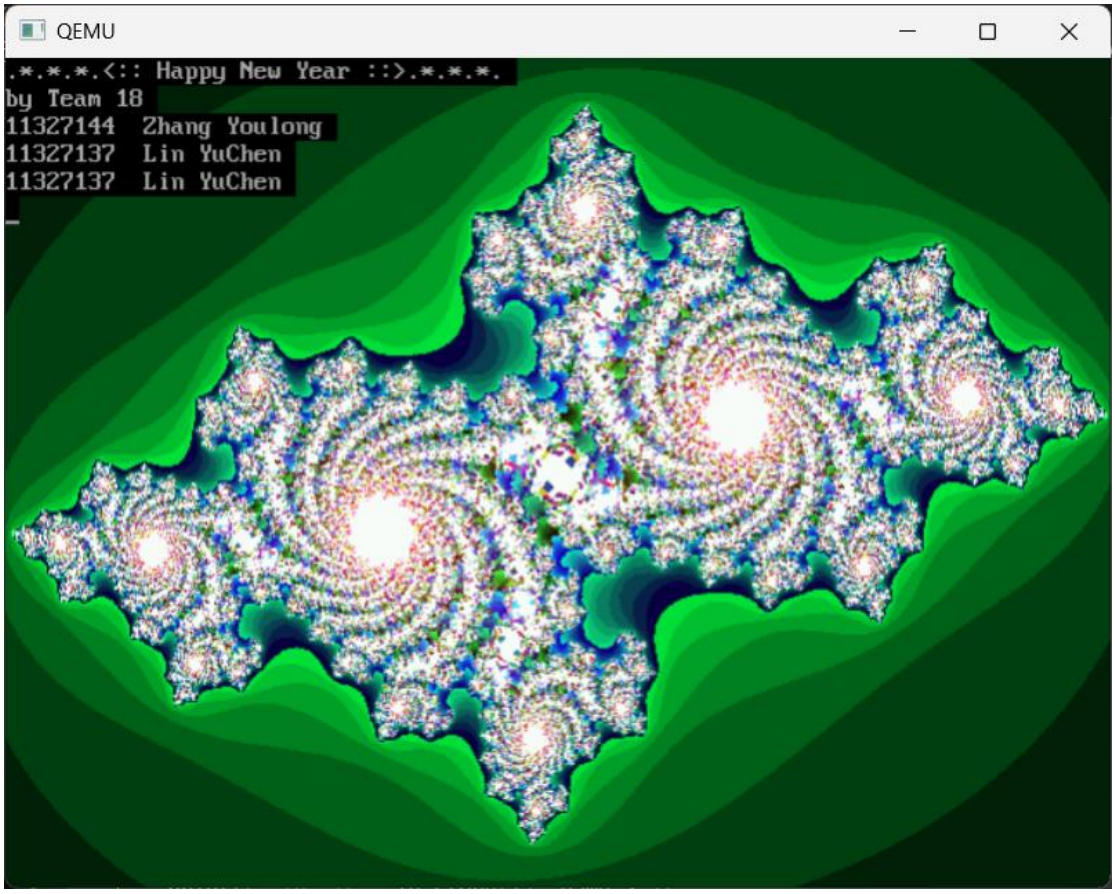
四、討論

成果圖:



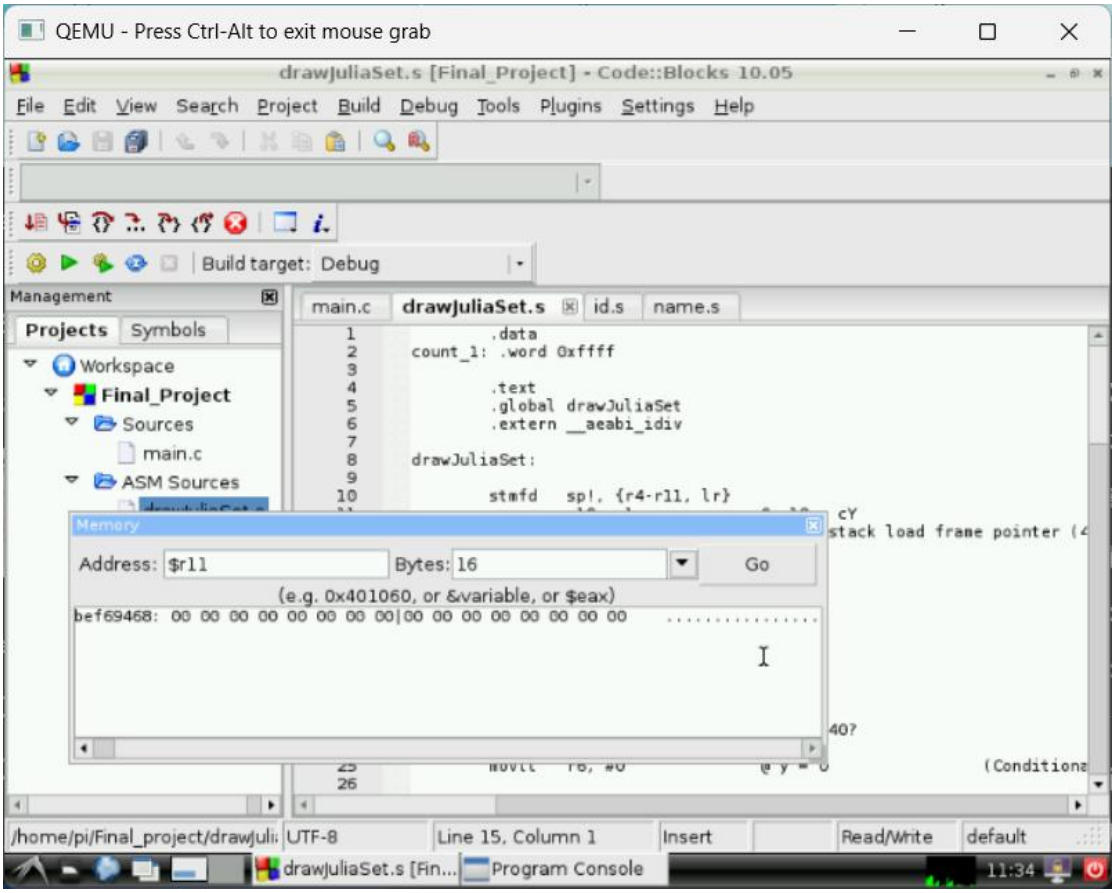

QEMU


QEMU


QEMU
.*.*.*.<:: Happy New Year ::>.*.*.*.
by Team 18
11327144 Zhang Youlong
11327137 Lin YuChen
11327137 Lin YuChen

執行結束並成功輸出。

- Frame 陣列記憶體區塊:



#起始位置:0xbef69468

#結束位置:0xbefff468

運算方式:

640(weight) * 480(height) = 307200

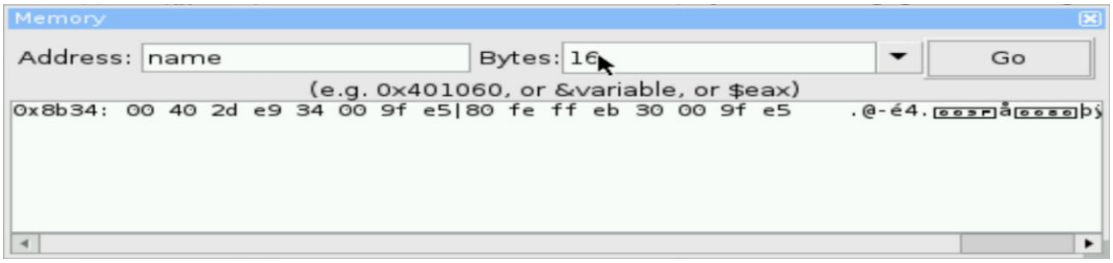
#256 色中: 1 像素 = 1 byte, 但本次宣告 int16_t = 2 byte

30720 * 2 = 614400

614400 轉 16 進位 = 0x96000

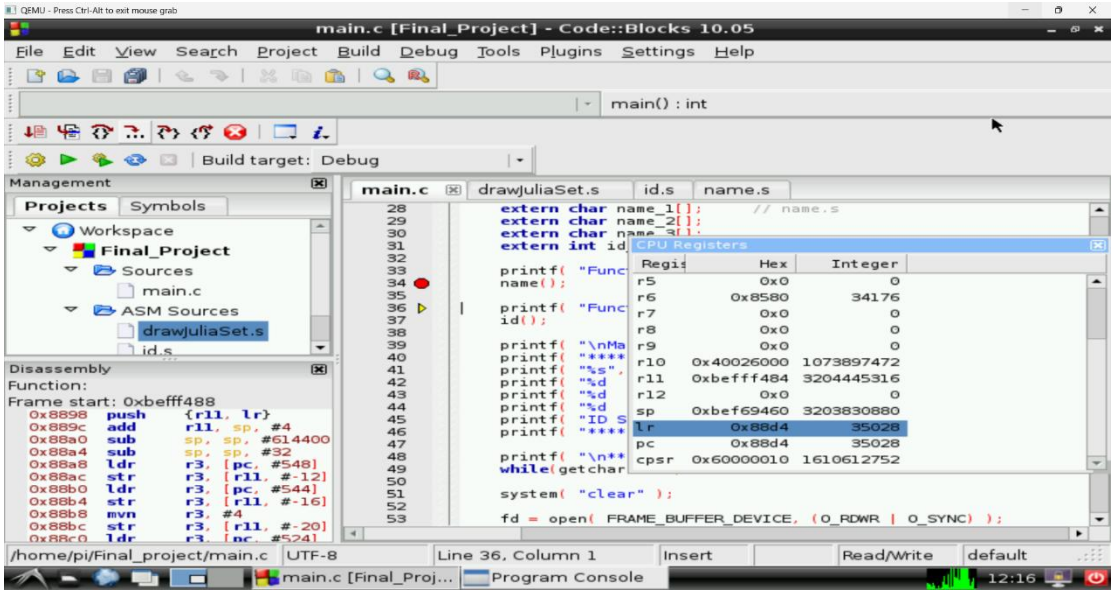
0xbef69469 + 0x96000 = 0xbefff468(結束位置)

- Name 函式的所在位置



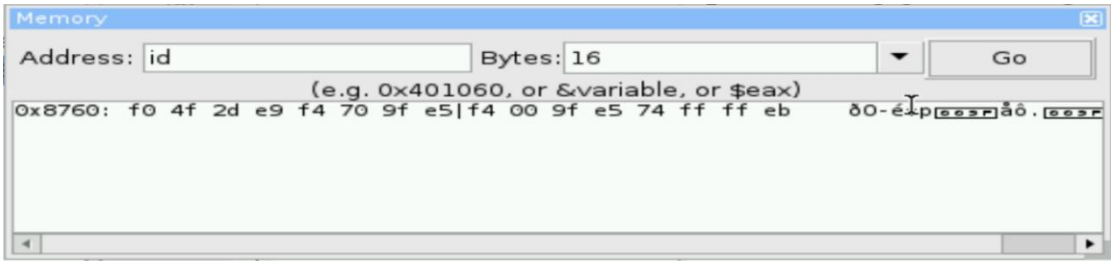
Name 函式的所在位置: 0x8b34

- Name 函式的 Return Address



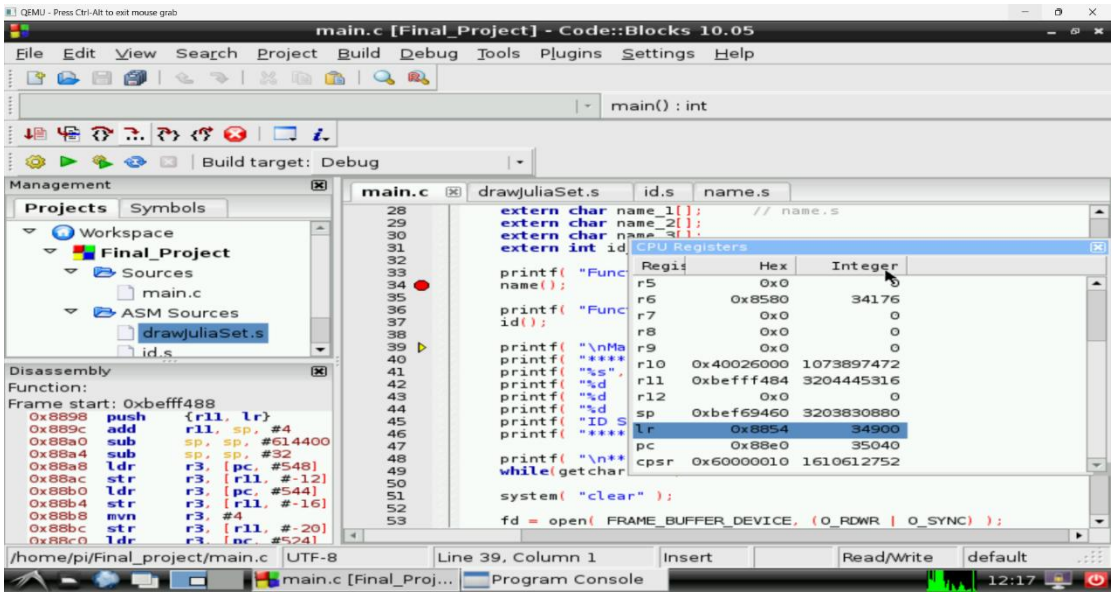
Name 函式的 Return Address: 0x88d4

- Id 函式的所在位置



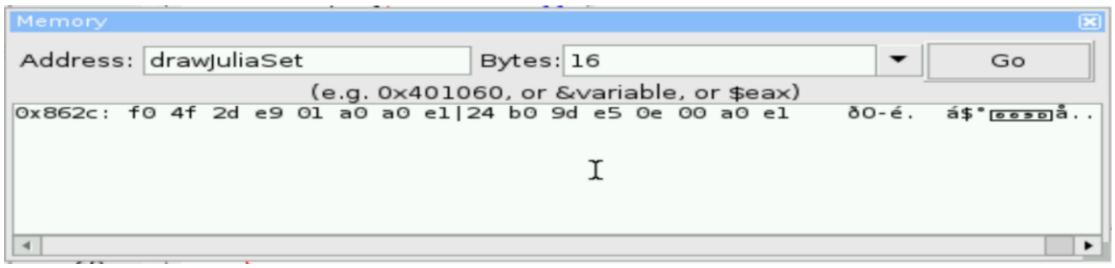
Id 函式的所在位置: 0x8760

- Id 函式的 Return Address



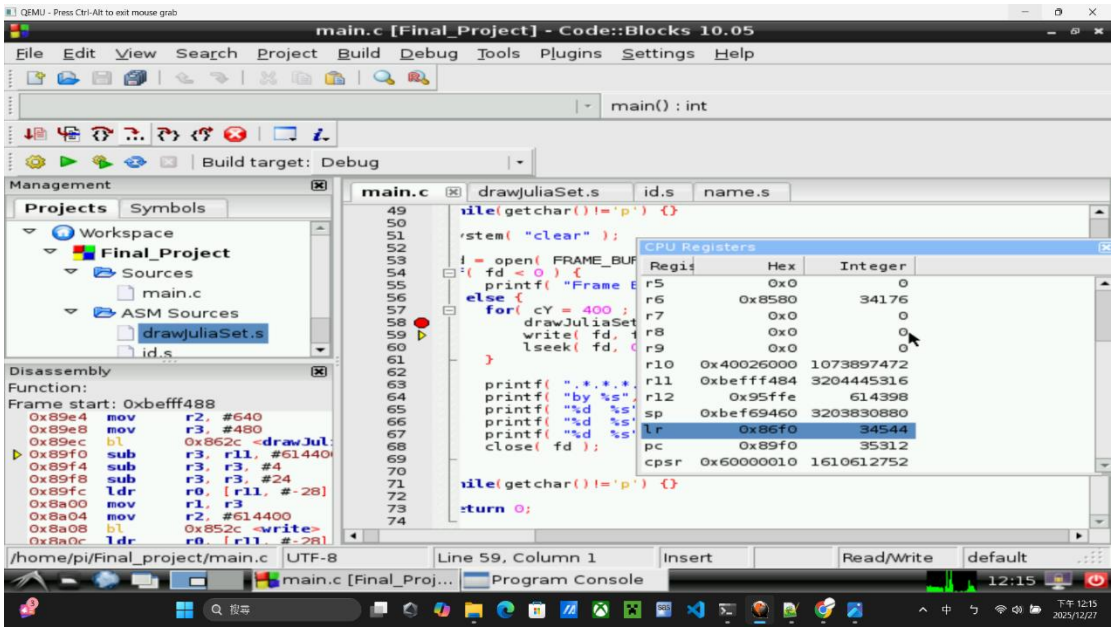
- Id 函式的 Return Address: 0x8854

- drawJuliaSet 函式的所在位置



drawJuliaSet 函式的所在位置: 0x862c

- drawJuliaSet 函式的 Return Address



drawJuliaSet 函式的 Return Address: 0x86f0

五、結論

1.整合 C 與 ARM 組合語言能力提升:

成功將 Midterm Project 的 name 與 id 函數整合到期末專題中，並以組語實作 drawJuliaSet 函數，熟悉函式呼叫、暫存器管理與 stack 使用。

2.掌握低階運算與暫存器管理:

透過 Julia Set 的雙層迴圈與 while 迴圈運算，實際操作資料處理指令與 conditional execution，理解暫存器保存、stack offset 計算的重要性。

3.除錯與實作經驗:

在撰寫過程中解決了暫存器覆蓋、記憶體位址錯誤與除法運算結果不正確等問題，加深對組合語言程式流程的理解。

4.專題設計與系統整合能力:

經歷 midterm project 到 final project，理解大型專題的程式模組化設計、資料傳遞、與多函式整合的重要性。

六、未來展望

- 精準度提升：

  目前為了運算效率多採用定點數運算，這在某些縮放層級下會出現失真。未來展望能導入高精度的浮點數運算邏輯，解決在極細微觀察下的圖形破碎問題，使分形圖形的細節更加精緻。

- 互動式介面開發：

  預計加入中斷處理機制，讓使用者能在程式執行期間透過鍵盤即時輸入不同的參數值，動態觀查 Julia Set 在不同複數平面上的連續演化過程，提升程式的教育與互動價值。

七、心得感想

在這次期末專題中，要實作一個具有實際圖形輸出的程式。與期中只在組合語言中撰寫小型功能不同，這次的專題需要考慮整體程式架構、函式介接方式，以及與作業系統硬體裝置的互動，也是一個相當具有挑戰性的挑戰。

在實作過程中，我最大的收穫是對 ARM EABI 呼叫慣例有更實際的理解。過去只是在課堂上知道參數會放在 r0 到 r3，其餘放在 stack，但在實際撰寫 drawJuliaSet 時，必須自己正確地保存暫存器、計算 stack offset 來取得 frame buffer 的位址，否則畫面就會顯示錯誤甚至造成程式當掉。這讓我們真正體會到暫存器管理與 stack 操作在組合語言中的重要性。

在除錯方面，也遇到不少問題，例如暫存器值被覆蓋、stack 位址計算錯誤，或是除法運算結果不如預期。這些錯誤在 C 語言中較容易被編譯器保護，但在組合語言中必須完全由程式設計者自行負責。透過一次次的測試與修正，我對程式實際在 CPU 中執行的流程有了更清楚的認識。

八、各組員分工方式與負責項目

11327137 林宇宸:
報告撰寫、drawJuliaSet.s 校正、main.c 實作

11327144 莊有隆:
報告撰寫、drawJuliaSet.s 實作、main.c 校正