

Lab1 ARM Assembly I

實驗一 ARM Assembly I

Group28 0616084林柏均 0616095曾敏峰

Lab1-1. Hamming distance

1.Code

```
1 .data
2 » result:.byte 0
3 .text
4 » .global main
5 » .equ X, 0x55AA
6 » .equ Y, 0xAA55
7
8 hamm:
9 » //TODO
10 » ldr R1,=#1
11 » and R1,R0
12 » add R4,R4,R1
13 » asr R0,#1
14 » cmp R0,#0
15 » bne hamm
16 » bx lr
17 main:
18 » ldr R0,=X
19 » ldr R1,=Y
20 » ldr R2,=result
21 » eor R0,R0,R1
22 » movs R4,#0
23 » bl hamm
24 » str R4,[R2]
25 L:.b.L
26
```

2.algorithm abstract

在資訊理論中，兩個等長字符串之間的漢明距離是兩個字符串對應位置的不同字符的個數。換句話說，它就是將一個字符串變換成另外一個字符串所需要替換的字符個數。

漢明重量是字符串相對於同樣長度的零字符串的漢明距離，也就是說，它是字符串中非零的元素個數：對於**二進位字符串**來說，就是1的個數，所以11101的漢明重量是4。

演算法十分簡單，只要將兩數 XOR後所得的結果，以二進制的方式檢視此結果共有幾個1，即為漢明距離。將結果依次操作 & 1，加總操作所得至某一暫存器，且對結果shift right by 1 bit，至結果為零，檢視上所提及暫存器之值，即為兩數的漢明距離。

3.Project -> Build All

18、19行movs會出現編譯問題，因為movs's #immediate 12bit，代表接受的即值範圍 $0 \sim (2^{12}) - 1$ ，但很明顯0x55AA、0xAA55都超過了此範圍，因此改用ldr's #immediate 32bit，就可以編譯了！

4.Result

i.

X=0x55AA

Y=0xAA55

(x)= Variables Breakpoints 1010 0101 Registers I/O Registers Modules		
Name	Value	Description
General Registers		General Purpose and FPU Reg...
1010 0101 r0	0	
1010 0101 r1	1	
1010 0101 r2	536870912	
1010 0101 r3	134218157	
1010 0101 r4	16	

536870912 <Hex>		536870912 : 0x20000000 <Signed Integer>			
Address	0 - 3	4 - 7	8 - B	C - F	
0000000020000000	16	0	0	536871668	
0000000020000010	5...	5...	0	0	
0000000020000020	0	0	0	0	
0000000020000030	0	0	0	0	
0000000020000040	0	0	0	0	
0000000020000050	0	0	0	0	

將X XOR Y所得數字為0xFFFF，X Y 的漢明距離為16。如register 中r4及memory 中0x20000000，答案正確。

ii.

```
.text
>> .global .main
>> .equ X, 0x5500
>> .equ Y, 0x0000
```

(x)= Variables Breakpoints Registers I/O Registers Modules		
Name	Value	Description
General Registers		
r0	Error: Target not available.	General Purpose and FPU Reg...
r1	Error: Target not available.	
r2	536870912	
r3	134218157	
r4	4	
Name : r4		
Hex:0x4		
Decimal:4		
Octal:04		

將X XOR Y所得數字為0x5500，X Y 的漢明距離為4。如register 中r4及memory 中0x20000000，答案正確。

Lab1-2. Fibonacci serial

1.Code

```
1 » .syntax unified
2 » .cpu cortex-m4
3 » .thumb
4 .text
5 .global main
6 .equ N, 49
7 fib:
8 » cmp r0, #101
9 » bge return
10 » cmp r0, #0
11 » ble return
12 » movs r4, #1
13 » cmp r0, #1
14 » beq return0
15 » cmp r0, #2
16 » beq return0
17 » movs r5, #2
18 » movs r1, #1
19 » movs r2, #1
20 » loop:
21 » » adds r4, r1, r2
22 » » add r5, r5, #1
23 » » bvs ovfreturn
24 » » movs r1, r2
25 » » movs r2, r4
26 » » cmp r5, r0
27 » » blt loop
28 » return0:
29 » » bx lr
30 » return:
31 » » movs r4, #0
32 » » subs r4, r4, #1
33 » » bx lr
34 » ovfreturn:
35 » » movs r4, #0
36 » » subs r4, r4, #2
37 » » bx lr
38 main:
39
40 » movs r0, #N
41
42 » bl fib
43 L: b L
..
```

2.algorithm abstract

算式如下：

$$F1 = 1$$

$$F2 = 1$$

$$Fn = Fn-1 + Fn-2, \text{ for } 1 \leq n \leq 100$$

演算法主要為設定初始值（分別在r1,r2），而後依次將兩數相加，結果存在r4，並且使用movs r1,r2 及 movs r2,r4，更新r1及r2的值，並

使用r5紀錄當前費氏數列的n為多少。直到r5內的值相等於r0（也就是N值）時，離開loop，此時r4即為所求Fib(N)。

需注意的部分是 $1 \leq N \leq 100$ ，因此使用

```
cmp r0, #101
bge return
cmp r0, #0
ble return
```

此四項指令來達成N若超出範圍，則跳到return，達到回傳-1。

若 Fib[N] 結果 overflow 的話回傳 -2，使用

```
adds r4,r1,r2

bvs  ovfreturn
```

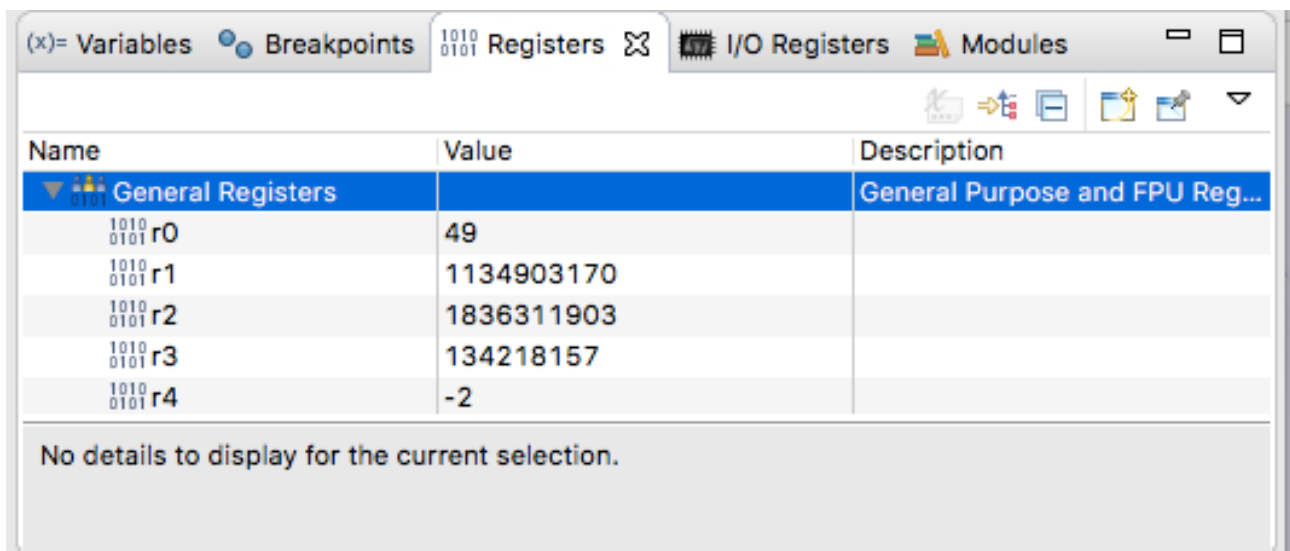
bvs來檢查是否產生overflow，若是則跳到ovfreturn這個標籤，達到回傳-2。






3.Problem I met

一開始不知道可以用多層標籤來實作費氏數列，寫起來略為複雜一些，而後知道了變輕鬆許多，整個結構更像C一點，顯得十分熟悉且可愛！

4.Result

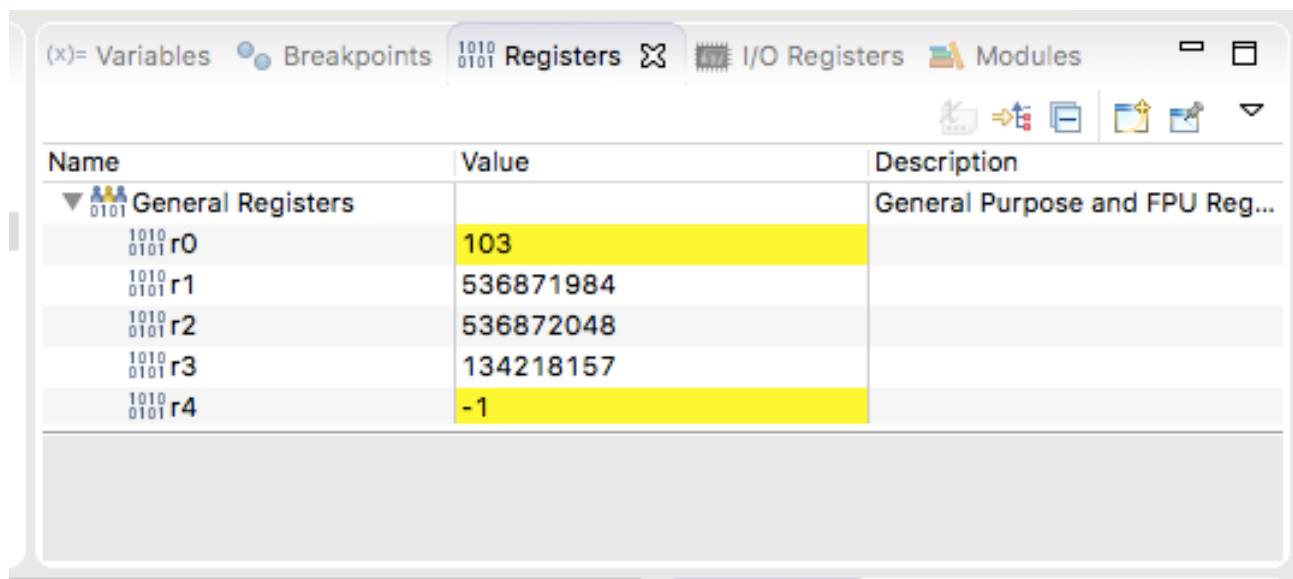
i. N=49時，overflow，r4=-2，答案正確。



Name	Value	Description
General Registers		General Purpose and FPU Reg...
 r0	49	
 r1	1134903170	
 r2	1836311903	
 r3	134218157	
 r4	-2	

No details to display for the current selection.

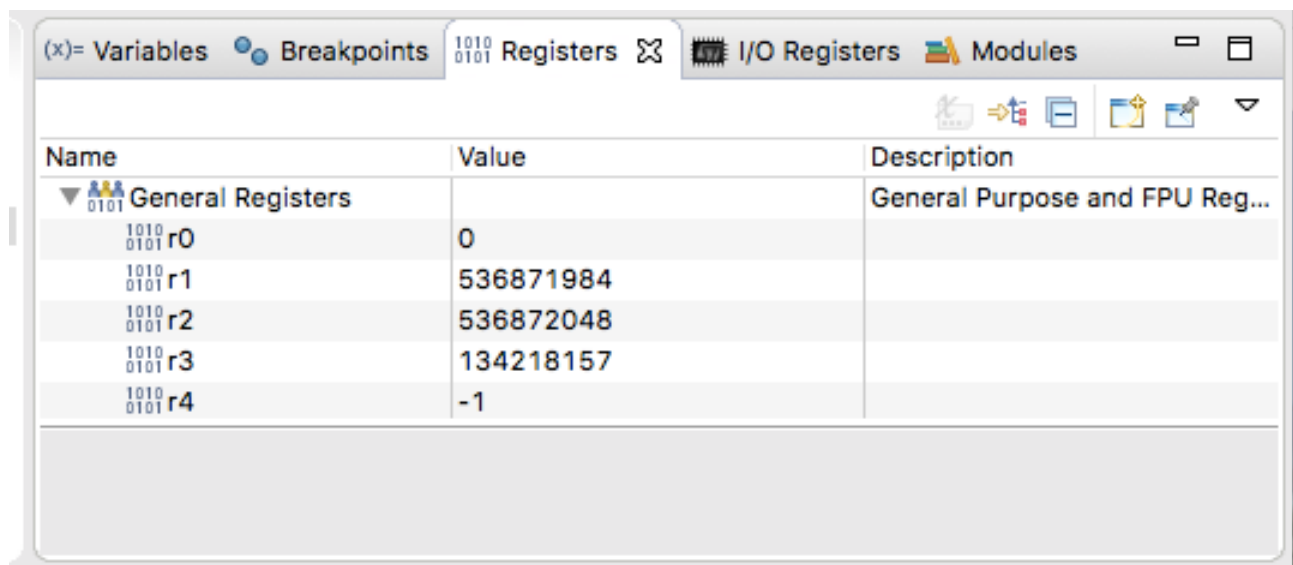
ii. $N = 103$ 時， $N > 100$ ， $r4 = -1$ ，答案正確。



The screenshot shows the 'Registers' window of a debugger. The 'General Registers' section is expanded, showing five registers: r0, r1, r2, r3, and r4. The values are: r0 = 103, r1 = 536871984, r2 = 536872048, r3 = 134218157, and r4 = -1. The registers r0 and r4 are highlighted in yellow.

Name	Value	Description
General Registers		General Purpose and FPU Reg...
r0	103	
r1	536871984	
r2	536872048	
r3	134218157	
r4	-1	

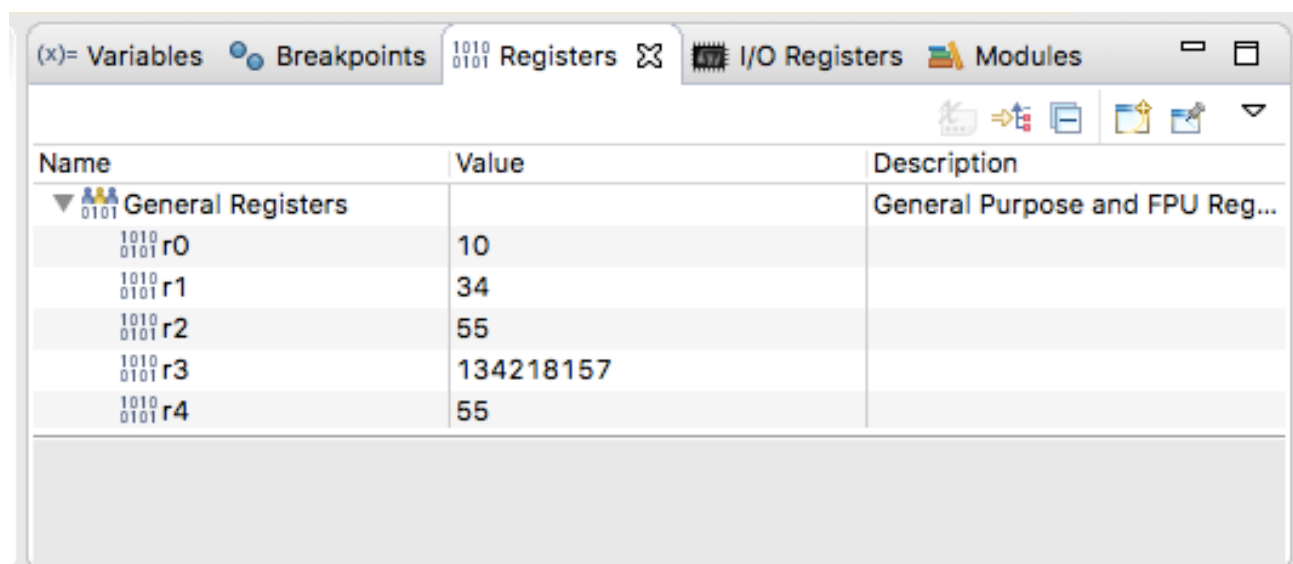
iii. $N = 0$ 時， $N < 1$ ， $r4 = -1$ ，答案正確。



The screenshot shows the 'Registers' window of a debugger. The 'General Registers' section is expanded, showing five registers: r0, r1, r2, r3, and r4. The values are: r0 = 0, r1 = 536871984, r2 = 536872048, r3 = 134218157, and r4 = -1.

Name	Value	Description
General Registers		General Purpose and FPU Reg...
r0	0	
r1	536871984	
r2	536872048	
r3	134218157	
r4	-1	

iv. $N = 0$ 時， $N < 1$ ， $r4 = -1$ ，答案正確。



The screenshot shows the 'Registers' window of a debugger. The 'General Registers' section is expanded, showing five registers: r0, r1, r2, r3, and r4. The values are: r0 = 10, r1 = 34, r2 = 55, r3 = 134218157, and r4 = 55.

Name	Value	Description
General Registers		General Purpose and FPU Reg...
r0	10	
r1	34	
r2	55	
r3	134218157	
r4	55	

Lab1-3. Bubble sort

1.Code

```
1 » .syntax unified
2 » .cpu cortex-m4
3 » .thumb
4 »
5 .data
6 » arr1:.byte 0x19, 0x34, 0x14, 0x32, 0x52, 0x23, 0x61, 0x29
7 » arr2:.byte 0x18, 0x17, 0x33, 0x16, 0xFA, 0x20, 0x55, 0xAC
8 .text
9 .global main
10
11 do_sort:
12 » //TODO
13 » ldr r1,=#8
14 » b outer
15 »
16 » outer:
17 » » sub r1,#1
18 » » ldr r2,=#0
19 » » cmp r1,#1
20 » » bgt inner
21 » » bx lr
22 » » inner:
23 » » » ldrb r3,[r0,r2]
24 » » » add r2,#1
25 » » » ldrb r4,[r0,r2]
26 » » » cmp r3,r4
27 » » » bgt swap
28 » » » b inner_check
29 » » » swap:
30 » » » ldrb r5,[r0,r2]
31 » » » strb r3,[r0,r2]
32 » » » sub r2,#1
33 » » » strb r5,[r0,r2]
34 » » » add r2,#1
35 » » » b inner_check
36 » » » inner_check:
37 » » » cmp r2,r1
38 » » » blt inner
39 » » » b outer
40 »
41 main:
42 » ldr r0,=arr1
43 » bl do_sort
44 » ldr r0,=arr2
45 » bl do_sort
46 »
47 » mov r10,1
48 »
49 » L:.b.L
```

2.algorithm abstract


演算法由兩層迴圈組成，分別由outer 及 inter標籤實作而成，outer 索引從7 -> 0，outer內由inner實作而成，inner索引從0 -> outer 索引，inner內比較r3 r4 是否為遞增，若否則跳到swap標籤，如此則可正確將8個分別為1byte的數字遞增排序完成。

另外參考講義第四章在assembly中實作If-else 的方法，再建一個inner_check標籤，成功達到非遞增跳swap，遞增則不跳的效果，

```
if (counter > 10) then
    counter = 0
else
    counter = counter + 1
```

```
CMP R0, #10      ; compare to 10
BLE incr_counter ; if less or equal, then branch
MOVS R0, #0      ; counter = 0
B     counter_done ; branch to counter_done
incr_counter
    ADDS R0, R0, #1    ; counter = counter +1
counter_done
...
```

3.Result

536870912 : 0x20000000 <Hex>  New Renderings...					
Address	0 - 3	4 - 7	8 - B	C - F	
0000000020000000	14192329	32345261	16171820	3355ACFA	
0000000020000010	00000000	FC020020	64030020	CC030020	
0000000020000020	00000000	00000000	00000000	00000000	
0000000020000030	00000000	00000000	00000000	00000000	
0000000020000040	00000000	00000000	00000000	00000000	
0000000020000050	00000000	00000000	00000000	00000000	

arr1: 0x41 0x19 0x23 0x29 0x32 0x34 0x52 0x61
arr2: 0x16 0x17 0x18 0x20 0x33 0x55 0xAC 0xFA

答案正确！