

# 實驗一 實驗環境建立與 Debugger 操作

0410001 電資 08 陳宏碩

## 1. 實驗目的

測試實驗器材

熟悉開發環境

## 2. 實驗步驟

### 2.1. 專案建立與程式編譯

請依照助教給的 lab1\_note 教學，建立一個 STM32 eclipse project，新增一個內容如下的 main.s 程式碼並透過 debugger 觀察程式執行結果。

```
.syntax unified
.cpu cortex-m4
.thumb

.text
.global main
.equ AA, 0x55

main:
    movs r0, #AA
    movs r1, #20
    adds r2, r0, r1

L: B L
```

### 2.2. 變數宣告與記憶體觀察

將 main.s 修改成以下程式碼並編譯執行觀察程式執行結果，並透過 memory monitor 觀察 X 內容值變化與回答問題。

```
.syntax unified
.cpu cortex-m4
.thumb

.data
X: .word 100
str: .asciz "Hello World!"
.text
.global main
.equ AA, 0x55

main:
    ldr r1, =X
    ldr r0, [r1]
```

```
movs r2, #AA
adds r2, r2, r0
str  r2, [r1]

ldr  r1, =str
ldr  r2, [r1]
L: B L
```

### 2.3. 簡易算數與基本記憶體指令操作

這部分實驗需要同學在 **data section** 中宣告三個 X,Y,Z 長度為 4byte 的變數並利用 ARM 組合語言計算以下式子，找出這些變數的 **memory address** 並觀察程式執行結果。

```
X = 5
Y = 10
X = X * 10 + Y
Z = Y - X
```

**Note:** 該程式需使用到算數指令 MULS, ADDS, SUBS 及記憶體讀寫操作指令 LDR, STR

## 3.實驗結果與分析

### 2.1. 專案建立與程式編譯

Q: 程式執行結束後 R2 值為多少？如何觀察？

**movs r0, #AA // r0 = 0x55, r0 = 85 ('.' .equ AA, 0x55)**

**movs r1, #20 // r1 = 20**

**adds r2, r0, r1 // r2 = r1 + r0 = 85 + 20 = 105**

**故程式執行結束後 R2 值為 105(十進位)或 0x69(十六進位)**

**利用 Registers Window 我們可以觀察 registers 的變化，如下圖一到圖四**

(x)= Variables Breakpoints Registers I/O Registers Modules		
Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0x2000042c (Hex)	
r1	0x42c (Hex)	
r2	0x2000046c (Hex)	
r3	0x0 (Hex)	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
No details to display for the current selection.		

圖一:初始狀態

(x)= Variables Breakpoints Registers I/O Registers Modules		
Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0x55 (Hex)	
r1	0x42c (Hex)	
r2	0x2000046c (Hex)	
r3	0x0 (Hex)	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
No details to display for the current selection.		

圖二: `movs r0, #AA`

(x)= Variables Breakpoints Registers I/O Registers Modules		
Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0x55 (Hex)	
r1	0x14 (Hex)	
r2	0x2000046c (Hex)	
r3	0x0 (Hex)	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
No details to display for the current selection.		


圖三: `movs r1, #20`

(x)= Variables Breakpoints Registers I/O Registers Modules		
Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0x55 (Hex)	
r1	0x14 (Hex)	
r2	0x69 (Hex)	
r3	0x0 (Hex)	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
No details to display for the current selection.		


圖四: `adds r2, r0, r1`，最後的執行結果

## 2.2. 變數宣告與記憶體觀察

X 的初始值為 100(0x64)

0x20000000 : 0x20000000 <Signed Integer>  + New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	100	1819043144	1867980911	560229490
0000000020000010	0	536871676	536871780	536871884

執行結束後 X 變為 185(0xB9)

0x20000000 : 0x20000000 <Signed Integer>  + New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	185	1819043144	1867980911	560229490
0000000020000010	0	536871676	536871780	536871884

Q1: 變數 X 與 str 的初始值是由誰在何處初始化的？

變數 X 與 str 的初始值是由 Assembler 在程式執行前存入 memory 裡面的 data segment 的







Q2: 若將 X 宣告改在 text section 對其程式執行結果會有何改變？

若將 X 宣告在 text section 如下圖所示，會造成 X 的位址從原本的 0x20000000 變成 0x80001f4，而 str 的位址則向前提前變為 0x20000000，原本 X 的位址，而因為 text section 只能 read 不能 write，所以 str r2, [r1]並不能成功修改 X 的值。

將 X 宣告改在 text section：

```
4.data
5    str: .asciz "Hello World!"
6.text
7    .global main
8    .equ AA, 0x55
9    X: .word 100
~
```

將 X 的位址存到 r1，由 register window 可以得知 X 的位址變為 0x80001f4:

Name	Value
General Registers	
 r0	0x2000043c (Hex)
 r1	0x80001f4 (Hex)
 r2	0x2000047c (Hex)
 r3	0x0 (Hex)
 r4	0
 r5	0

由 memory monitor 觀察 X，最後還是為 100:

Address	0 - 3	4 - 7	8 - B	C - F
00000000080001F0	0	100	1745373443	403841621
0000000008000200	1224892426	-402757622	134218228	536870912

由 memory monitor 觀察 str

Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	48656C6C	6F20576F	726C6421	00000000
0000000020000010	00000000	FC020020	64030020	CC030020

Q3: 程式執行完畢後 r2 內容與 str 字串在 memory 前 4 個 byte 呈現內容有何差異？

.data

X: .word 100

str: .asciz "Hello World!"

ldr r1, =X // r1 = the address of X

ldr r0, [r1] // load the value of X into r0

movs r2, #AA // set r2 = 0x55(0x55.equ AA, 0x55)

adds r2, r2, r0 // r2 = r2 + r0 = 0x55 + 0x64(100) = 0xB9



str r2, [r1] // store r2 into X

ldr r1, =str // r1 = the address of str

ldr r2, [r1] // load the value of str into r2

依照程式碼，r2 內容應與 str 字串在 memory 前 4 個 byte 相同，但如下圖所示，我們發現兩者的順序稍微不同在 register 裡 0x6c6c6548 而在 memory 裡為 48656c6c，兩者順序為相反，但其實兩者是相同的，只是在 memory 裡的顯示為一次顯示一個 byte 然後由小的位元顯示到大的位元，故在 memory 前面的為低位元，而後面的是高位元，我推測可能是因為 memory 式利用 stack 的方式存取所以會造成這樣的現象。

(x)= Variables Breakpoints Registers I/O Registers Modules	
Name	Value
General Registers	
r0	0x64 (Hex)
r1	0x20000004 (Hex)
r2	0x6c6c6548 (Hex)
r3	0x0 (Hex)

0x20000000 : 0x20000000 <Hex>   New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	B9000000	48656C6C	6F20576F	726C6421
0000000020000010	00000000	00000000	00000000	04030020
0000000020000020	6C030020	D4030020	00000000	00000000
0000000020000030	00000000	00000000	00000000	00000000
0000000020000040	00000000	00000000	00000000	00000000

Q4: 變數 str “Hello World!” 有無其他種宣告方式？若有請說明其中一種。

有，除了使用 `.asciz` 外，還可以使用 `.ascii` 宣告如下，兩者的最大差別為利用 `.asciz` 宣告每一個 string 都會跟隨一個 zero byte，執行結果會相同

```
str: .ascii "Hello World!"
```

### 2.3. 簡易算數與基本記憶體指令操作

計算的組合語言如下：

```
.syntax unified
.cpu cortex-m4
.thumb

.data
X: .word 5 //declare X as 5
Y: .word 10 // declare Y as 10
Z: .word 0 // declare Z as 0

.text
.global main

main:
ldr r2, =X // load the address of X into r2
ldr r3, =Y // load the address of Y into r3
ldr r0, [r2] // load the value of X into r0
ldr r1, [r3] // load the value of X into r0
movs r4, #10 // set r4 to 10
muls r0, r0, r4 // r0 = r0 * r4 = 5 * 10 = 50
adds r0, r0, r1 // r0 = r0 + r1 = 50 + 10 = 60
subs r5, r1, r0 // r5 = r1 - r0 = 10 - 60 = -50
str r0, [r2] // store the value of r0 into X
ldr r6, =Z // load the address of Z into r6
str r5, [r6] // store the value of r5 into Z
```

L: B L

X 的 memory address 為 0x20000000(r2)



Y 的 memory address 為 0x20000004(r3)

Z 的 memory address 為 0x20000008(r6)



下圖為執行結果的 register window

<small>32bit</small> <small>32bit</small> r0	0x3c (Hex)
<small>32bit</small> <small>32bit</small> r1	0xa (Hex)
<small>32bit</small> <small>32bit</small> r2	0x20000000 (Hex)
<small>32bit</small> <small>32bit</small> r3	0x20000004 (Hex)
<small>32bit</small> <small>32bit</small> r4	10
<small>32bit</small> <small>32bit</small> r5	-50
<small>32bit</small> <small>32bit</small> r6	0x20000008 (Hex)



Memory 初始的樣子：

0x20000000 : 0x20000000 <Signed Integer>   New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	5	10	0	0
0000000020000010	0	536871676	536871780	536871884

將計算過後的 X 值存入  $X = 10 * X + Y = 10 * 5 + 10 = 60$

0x20000000 : 0x20000000 <Signed Integer>   New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	60	10	0	0
0000000020000010	0	536871676	536871780	536871884

將計算過後的 Z 值存入  $Z = Y - X = 10 - 60 = -50$

0x20000000 : 0x20000000 <Signed Integer>   New Renderings...				
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	60	10	-50	0
0000000020000010	0	536871676	536871780	536871884

#### 4.心得討論與應用聯想

這次的實驗為第一次的實驗，主要在讓我們認識這一塊板子，讓我覺得相當特別的是這塊板子還有 debug 的功能，這樣會讓之後的實作變相對來說輕鬆，但在實驗過程中，因為還是有許多背後的原理，所以很多的過程和來龍去脈還沒有相當的清楚，可能之後上完課會有更多的收穫。而在看 user manual 也發現感覺還有蠻多可以開發的東西，相信之後我們把各個外部介面的零件裝上去後，我們可以做出很有趣的東西，第一次的實驗相當的好，非常容易了解。