



## 實驗三 ARM Assembly II

### 1. 實驗目的

熟悉基本 ARMv7 組合語言語法使用。

### 2. 實驗原理

請參考上課 Assembly 部分講義。

### 3. 實驗步驟

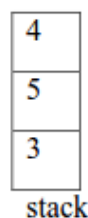
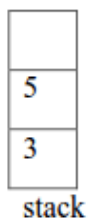
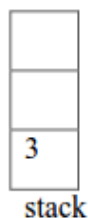
#### 3.1. Postfix arithmetic

操作 stack 來完成 postfix 的加減法運算

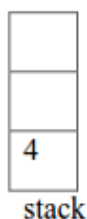
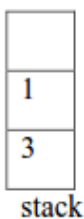
Using stack to evaluate postfix expression which only includes addition and subtraction operations.

3.1.1. Example: 3, 5, 4, -, +

(1) 3, 5, 4, -, +    (2) 3, 5, 4, -, +    (3) 3, 5, 4, -, +



(4) 3, 5, 4, -, +    (5) 3, 5, 4, -, +



#### 3.1.2. 實作要求

完成以下的程式碼，必須要利用 PUSH, POP 操作 stack 來完成 postfix expression 的運算，並將結果存進 expr\_result 這個變數裡。

Please Complete the program below. You must use PUSH, POP operations to calculate the result of the postfix expression, and store it into variable “expr\_result”.



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

.data
    user_stack .zero 128
    expr_result .word 0

.text
.global main
postfix_expr .asciz    "-100 10 20 + - 10 +"

main:
    LDR R0, =postfix_expr

//TODO: Setup stack pointer to end of user_stack and calculate the
expression using PUSH, POP operators, and store the result into
expr_result

program_end:
    B     program_end

atoi:
    //TODO: implement a "convert string to integer" function
    BX LR
```

**postfix\_expr 格式：**postfix\_expr 是一串 postfix 運算式的字串，每個數字/運算子之間會用 1 個空白來區隔；input 的數字是 10 進位整數，數字正負數皆支援，字串以 ascii value 0 作為結尾；**可以假設此運算式必可求出解。**

Format of postfix\_expr: "postfix\_expr" is a postfix expression. In the expression, every operand/operator is separated with a space. The operands could be signed decimal numbers, and the operators could be "+" or "-". The string of the postfix expression is ended with a ascii value 0. **YOU CAN ASSUME THAT THE EXPRESSION IS LEGAL.**

### Prototype of atoi:

Input : start address of the string (using register)

Output : integer value (using register)

**Hint:** 可以利用 MSR 來修改 MSP(Main Stack Pointer)的值

Hint: You can use MSR to modify the value of MSP(Main Stack Pointer)

### Reference:

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489f/CIHFIDAJ.html>

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0497a/CHDBIBGJ.html>



**Note:**助教會在 demo 時修改 postfix\_expr 數值

Note: We will change the value of postfix\_expr in demo.



### 3.2. 求最大公因數並計算最多用了多少 stack size

在程式碼中宣告 2 個變數 m 與 n，並撰寫 Stein 版本的最大公因數，將結果存入變數 result 裡，請使用 recursion 的寫法，並使用 stack 傳遞 function 的 parameters，禁止單純用 register 來傳。

Declare two variables “m, n”. Using Stein’s algorithm to find the GCD(Greatest Common Divisor) of them, and storing the result into variable “result”. Please use recursion to implement the algorithm and use stack to pass the parameters of the function. Don't pass the parameters with registers directly.

計算在 recursion 過程中，記錄最多用了多少 stack size，並將它存進 max\_size 這個變數中。

Calculate the maximum stack size used in the recursion process, and store the result into variable “max\_size”.

```
.data
    result: .word 0
    max_size: .word 0
.text
    m: .word 0x5E
    n: .word 0x60

GCD:
    //TODO: Implement your GCD function
    BX LR
```

Prototype of GCD:

Input : A,B (using stack)

Output : GCD value (using register), max stack size (using register)

**Hint:** stack 的操作

Hint: manipulations of stack

```
MOVS    R0, #1;
MOVS    R1, #2
PUSH    {R0, R1}
LDR     R2, [sp]      // R2 = 1
LDR     R3, [sp, #4]  // R3 = 2
POP     {R0, R1}
```

**Note :** 助教會在 demo 時修改 m, n 數值

Note: We will change the value of m, n in demo.

**Reference:**

GCD Algorithm (Euclid & Stein) :

<http://www.cnblogs.com/drizzlecrj/archive/2007/09/14/892340.html>