Lab2 ARM Assembly II
實驗二 ARM Assembly II

1. Lab objectives 實驗目的

Familiar with the ARMv7 assembly language programming.

熟悉基本ARMv7組合語言語法使用。

2. Lab principle實驗原理

Reference by the course materials.

請參考上課Assembly部分講義。

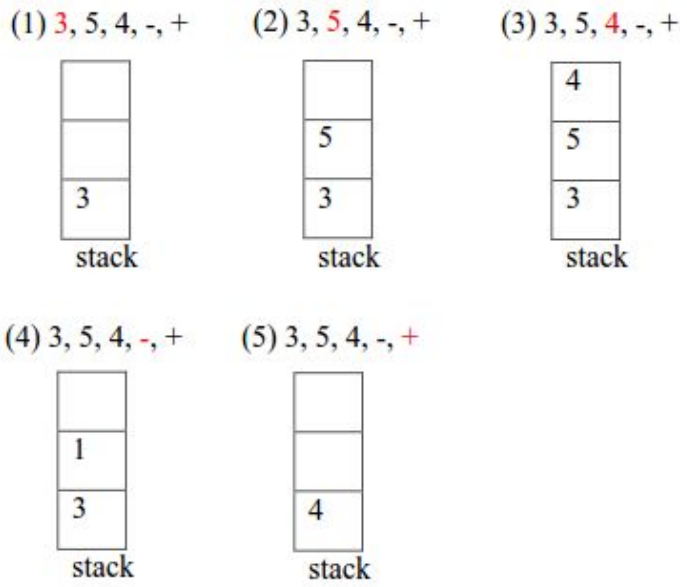
3. Steps 實驗步驟

1.1. Postfix arithmetic (50%)

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

操作 stack 來完成 postfix 的加減法運算

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



1.1.2. 實作要求

Please Complete the program below. You must allocate space in data section and set that space as stack. Must use PUSH, POP operations to calculate the result of the

postfix expression, and store it into variable “expr_result”.

完成以下的程式碼。你必須在 data section 分配空間，並將那空間設置為 stack。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。

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

.data
   user_stack_bottom: .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
```

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 asci value 0. YOU CAN ASSUME THAT THE EXPRESSION IS LEGAL.

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，運算只有加減，字串以 ascii value 0 作為結尾；可以假設此運算式必可求出解。

Prototype of atoi:

Input : start address of the string (using register)

Output : integer value (using register)

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

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

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: We will change the value of postfix_expr in demo.

Note: 助教會在 demo 時修改 postfix_expr 數值

1.2. 求最大公因數並計算最多用了多少 stack size (50%)

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 (Stein's Algorithm)。

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 (Stein's Algorithm).

GCD Stein's Algorithm (C version) :

```
int GCD(int a, int b)
{
   if(a == 0) return b;
   if(b == 0) return a;
   if(a % 2 == 0 && b % 2 == 0) return 2 * GCD(a >> 1, b >> 1);
   else if(a % 2 == 0)  return GCD(a >> 1, b);
   else if(b % 2 == 0) return GCD(a, b >> 1);
   else return GCD(abs(a - b), min(a, b));
}
```

計算在 recursion 過程中，記錄最多用了多少 stack size (以 byte 為單位)，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" (use byte as unit).

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

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

```
main:
  // r0 = m, r1 = n
  BL GCD
  // get return val and store into result
```

Prototype of GCD:

Input : A, B (using register)

Output : GCD value (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.