Lab3 ARM Assembly II
實驗三 ARM Assembly II

0516059劉嘉豪,
0516322朱蝶

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

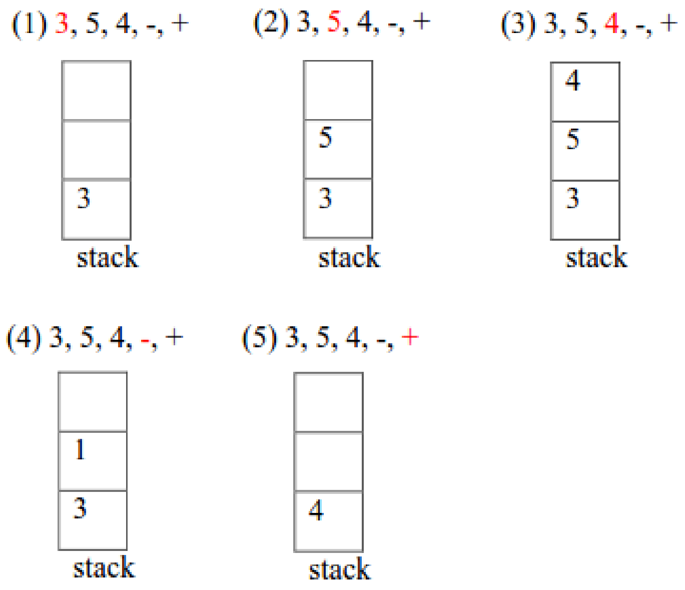
Reference by the course materials. 請參考上課 Assembly 部分講義。

3. Steps 實驗步驟

1.1. Postfix arithmetic (40%)

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 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，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: .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.Problem definition

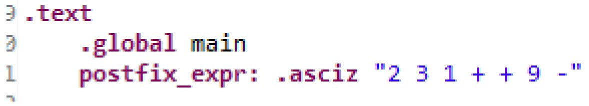
Following is algorithm for evaluation postfix expressions. 1) Create a stack to store operands (or values). 2) Scan the given expression and do following for every scanned element. …..a) If the element is a number, push it into the stack …..b) If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack 3) When the expression is ended, the number in the stack is the final answer

2.Pseudo Code

```
While(expr[i]!='\0'){

    if(is_space){
        i++;
    }

    Else if(is_integer){
        compute_value;
    }
    Else if(is_neg){
        Check if(is_space(expr[i+1])){
            Is_minus;
        }else{
            Is_neg_number;
        }
    }
    Else if(is_plus){
        Is_plus;
    }

}

Is_minus/is_plus(){
    Pop two num and do the operation;
    Push result back
}
```

3. test case and result

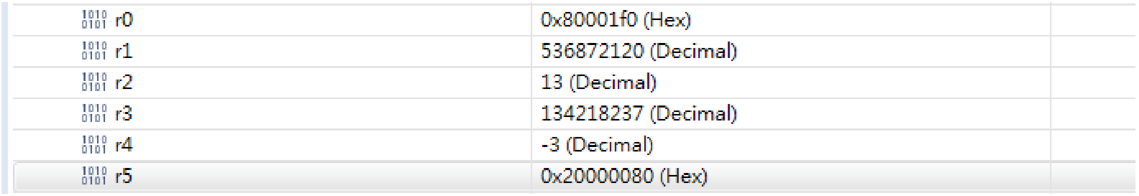
case 1:



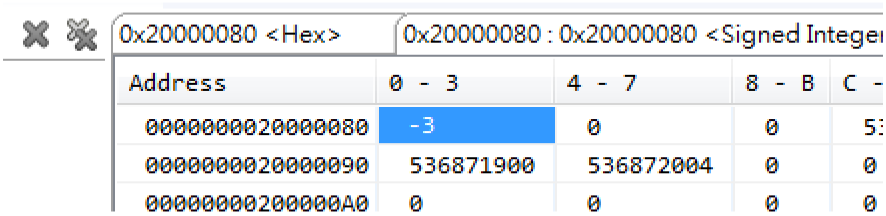
```
.text
	.global main
	postfix_expr: .asciz "2 3 1 + + 9 -"
```

Case 1 result:

R4為答案，R5為存放答案的位址

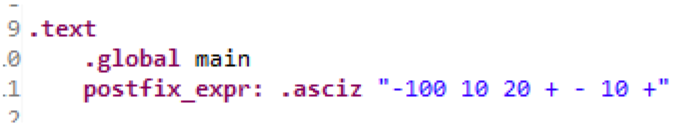


r0	0x80001f0 (Hex)
r1	536872120 (Decimal)
r2	13 (Decimal)
r3	134218237 (Decimal)
r4	-3 (Decimal)
r5	0x20000080 (Hex)



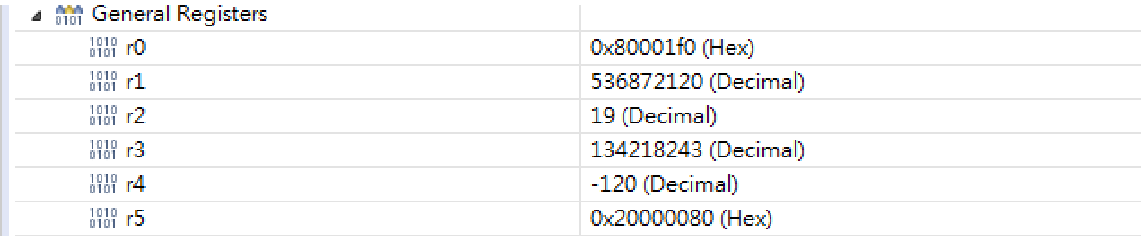
Address	0 - 3	4 - 7	8 - B	C -
0000000020000080	-3	0	0	5:
0000000020000090	536871900	536872004	0	0
00000000200000A0	0	0	0	0

Case 2:

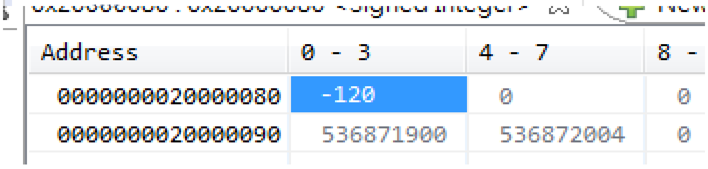


```
.text
	.global main
	postfix_expr: .asciz "-100 10 20 + - 10 +"
```

Case2 result:



General Registers	
r0	0x80001f0 (Hex)
r1	536872120 (Decimal)
r2	19 (Decimal)
r3	134218243 (Decimal)
r4	-120 (Decimal)
r5	0x20000080 (Hex)



Address	0 - 3	4 - 7	8 -
0000000020000080	-120	0	0
0000000020000090	536871900	536872004	0

1.2. 求最大公因數並計算最多用了多少 **stack size (60%)**

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

1. Problem definition:

The steps to find GCD using Stein's Algorithm gcd(a, b).

(1) If Both a and b are 0s, gcd is zero gcd(0, 0)=0.

(2) gcd(a, 0) = gcd(0, b) = 0, because every number divides 0.

(3) If a and b are both even, gcd(a, b) = 2*gcd(a/2, b/2) because 2 is a common divisor. Multiplication with 2 can be done with bitwise shift operator.

(4) If a is even and b is odd, gcd(a, b) = gcd(a/2, b). Similarly, if a is odd and b is even, then gcd(a, b) = gcd(a, b/2). It is because 2 is not a common divisor.

(5) If both a and b are odd, then gcd(a, b) = gcd(|a-b|/2, b). Note that difference of two odd numbers is even.

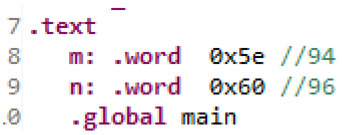
(6) Repeat steps 3–5 until a = b, or until a = 0. In either case, the GCD is power(2, k) * b, where power(2, k) is 2 raise to the power of k and k is the number of common factors of 2 found in step 2.

2. Pseudo Code

```
Main(){
    Push m,n into stack;
    Gcd();
    Store result and max_size into memory;
}

Gcd(){
    Pop{m,n}
    If(m==n)        return m;
    If(m==0||n==0)        return the not zero number
    If(m%2==0){
        If(n%2==1)
            gcd(m>>1,n)
        else  //both even
            gcd(m>>1,n>>1)<<1
    }else{
        If(n%2==0)
            gcd(m,n>>1)
        else  //both odd
            gcd(|m-n|>>1,min(m,n))
    }
}
```

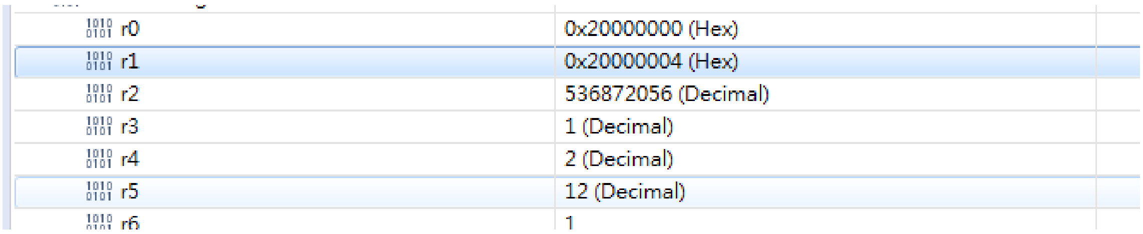
Three. Test case and result

Case 1



```
.text
    m: .word  0x5e //94
    n: .word  0x60 //96
    .global main
```

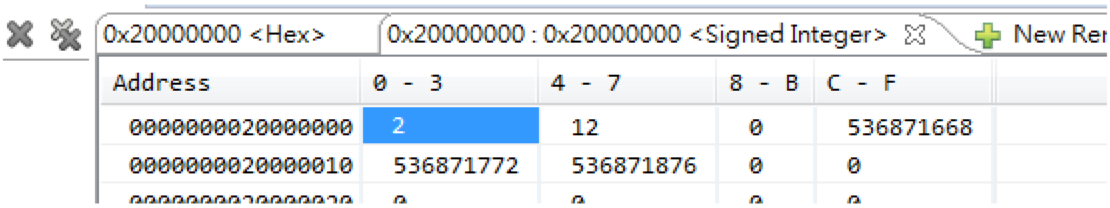
Result:



r0	0x20000000 (Hex)
r1	0x20000004 (Hex)
r2	536872056 (Decimal)
r3	1 (Decimal)
r4	2 (Decimal)
r5	12 (Decimal)
r6	1

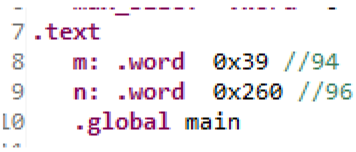
R4(R0)為最大公因數，R5(R1)為stack max size

Memory的存放狀況



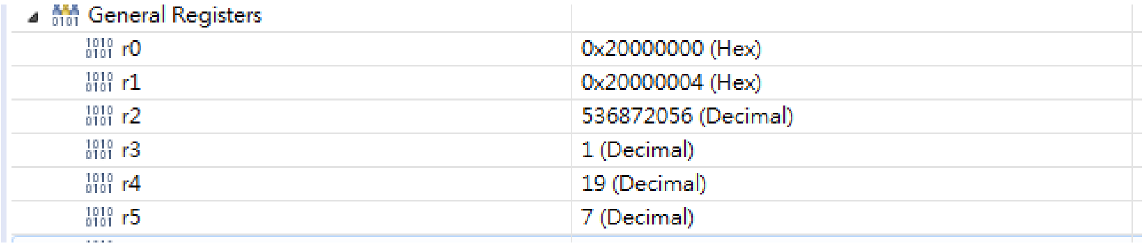
Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	2	12	0	536871668
0000000020000010	536871772	536871876	0	0
0000000020000020	0	0	0	0

Case 2

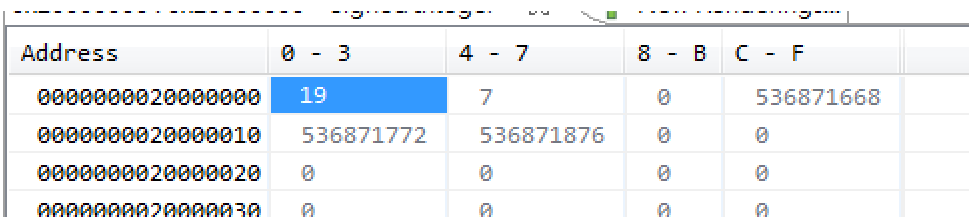


```
.text
    m: .word  0x39 //94
    n: .word  0x260 //96
    .global main
```

Result:



r0	0x20000000 (Hex)
r1	0x20000004 (Hex)
r2	536872056 (Decimal)
r3	1 (Decimal)
r4	19 (Decimal)
r5	7 (Decimal)



Address	0 - 3	4 - 7	8 - B	C - F
0000000020000000	19	7	0	536871668
0000000020000010	536871772	536871876	0	0
0000000020000020	0	0	0	0
0000000020000030	0	0	0	0

Conclusion:

第一題在實作時常常忘記把位址裡面的值存放到暫存器裡面，導致出來的結果相去甚遠，一步一步檢查之後才把所有該存放的值存放好。

第二題，bx lr hard -> easy。

總之，這次做比較久