Lab3 ARM Assembly II

0510532 楊上萱

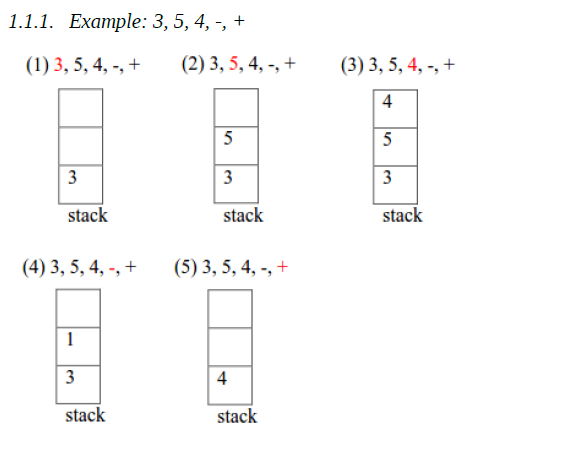
# Lab objectives 實驗目的

熟悉基本ARMv7組合語言語法使用。

# Steps 實驗步驟

* 1. Postfix arithmetic

操作stack 來完成postfix的加減法運算



C++實作如下：

|  |
| --- |
| #include <iostream>  #include <stack>  #include <cstdio>  #include <cstdlib>  #include <cstring>  **using** **namespace** std**;**  int main**(){**  stack**<**int**>** s**;**  char input**[**100000**];**  char **\***p**;**  int value **=** 0**;**  int top1**,** top2**;**  int answer **=** 0**;**  cin**.**getline**(**input**,** 100000**);**  p **=** strtok**(**input**,** " "**);**  **while(**p **!=** **NULL){**  **if(!**strcmp**(**p**,** "+"**)** **||** **!**strcmp**(**p**,** "-"**)** **||** **!**strcmp**(**p**,** "\*"**)){**  top1 **=** s**.**top**();**  s**.**pop**();**  top2 **=** s**.**top**();**  s**.**pop**();**  **if(!**strcmp**(**p**,** "+"**)){**answer **=** top1**+**top2**;** s**.**push**(**answer**);}**  **else** **if(!**strcmp**(**p**,** "-"**)){**answer **=** top2**-**top1**;** s**.**push**(**answer**);}**  **else** **if(!**strcmp**(**p**,** "\*"**)){**answer **=** top1**\***top2**;** s**.**push**(**answer**);}**  **}**  **else{**  value **=** atoi**(**p**);**  s**.**push**(**value**);**  **}**  p **=** strtok**(NULL,** " "**);**  **}**  printf**(**"%d\n"**,** s**.**top**());**  **return** 0**;**  **}** |

ARM Assembly實作原理：

1. 移動pointer來load要處理的string (using ldrb)
2. 因為有正負數，所以用一個register來記住當前的sign(r4)
3. 用ascii比較每一個character：
   * 數字→做atoi並把數字push進stack中
   * ’+’→pop出stack最上面的兩個數做加法再push回去
   * ’-‘→遇到減號有兩種情況，需要往後看一個character，若是空格，則代表現在要做減法運算，pop出stack最上面的兩個數做減法再push回去；若不是空格，則代表這個負號表示負數，把sign register更新為-1，並繼續做atoi
   * 空格→pointer往後移動
   * Ascii 0→程式結束
   * Stack中剩下的數即為答案
4. atoi的實作如下：
   * initial: atoi\_result=0
   * 第一次進來：num-=48; result+=num; update pointer;
   * 接著loop：num -= 48; result+=num; result\*=10; update pointer;
   * loop直到碰到space(ascii 32)，把result乘上sign並push進stack
   1. 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。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。

C++實作：

|  |
| --- |
| 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**);**  //if(a&1==0 && b&1==0)  **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**));**  **}** |

ARM實作原理：

1. 基本上原理就是照著C++的判斷式來決定回傳值，每次呼叫GCD時順便把parameters(m和n)、return address(lr)push進stack，等到遞迴回來時便可以直接從stack中pop出來用

# Results and analysis 實驗結果與分析

* 1. Postfix Arithmetic

push(-100)



push(10)



push(20)



pop(20), pop(10), 10+20=30



Pop(30), pop(-100), -100-30=-130



push(10)



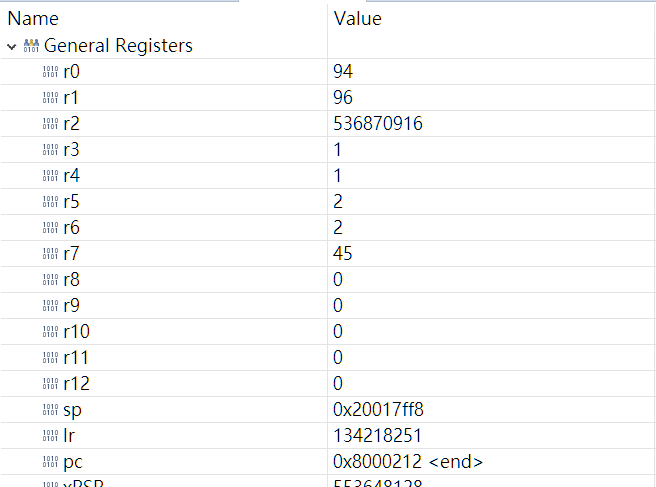
pop(10), pop(-130), -130+10=-120



Program end



* 1. 求最大公因數並計算最多用了多少stack size



1. gcd(94, 96)

= 2\*gcd(47, 48) = 2\*gcd(47, 24)=2\*gcd(47, 12)=2\*gcd(47, 6)=2\*gcd(47, 3)

=2\*gcd(44, 3)=2\*gcd(22, 3)=2\*gcd(11, 3)=2\*gcd(8, 3)=2\*gcd(4, 3)=2\*gcd(2, 3)

=2\*gcd(1, 3)=2\*gcd(2, 1)=2\*gcd(1, 1)=2\*gcd(0, 1)

=2\*1 = 2

1. 遞迴深度總共15層，每次push進去三個數，因此max\_size為3\*15=45
2. **Conclusions and ideas 心得討論與應用聯想**

這次作業花了蠻多時間和心思，不過也學到了許多。實作postfix arithmetic的同時除了更熟悉stack的操作，還學會了如何byte-wise處理字串，實作Stein gcd可以更清楚地了解原本在軟體層面實作遞迴時很難看出來的stack變化。

1-1程式碼

|  |
| --- |
| **.**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 +"  // r0: postfix\_eppr  // r1 -> r0  // r2: present char  // r3: next char, r3=10  // r4: sign  // r5: atoi result  // r6: pop -> r6(first)  // r7: pop -> r7(second)  main**:**  LDR R0**,** **=**postfix\_expr  movs r1**,** r0 // r1 = "-100 10 20 + - 10 +"  ldr sp**,** **=**user\_stack  adds sp**,** sp**,** 128  strtok**:**  movs r4**,** 1 // sign = 1  movs r5**,** #0 // atoi\_result = 0  ldrb r2**,** **[**r1**]** // load byte  cmp r2**,** #48 // is num?  bge atoi\_1 // is num: atoi  cmp r2**,** #45 // '-'  beq isMinus  cmp r2**,** #43 // '+'  beq add\_op  cmp r2**,** #32 // 'space'  beq space  cmp r2**,** #0 // 'end'  beq program\_end  program\_end**:**  pop **{**r0**}**  ldr R1**,** **=**expr\_result  str r0**,** **[**r1**]**  B program\_end  atoi\_1**:**  subs r2**,** r2**,** 48 // ascii to num  adds r5**,** r5**,** r2 // result += num  adds r1**,** r1**,** 1 // update pointer(r1)  ldrb r2**,** **[**r1**]**  b atoi  atoi**:**  //TODO: implement a “convert string to integer” function  // result = 0; result += num; result \*= 10;  cmp r2**,** #32 // is space?  beq push\_stack  subs r2**,** r2**,** 48 // ascii to num  adds r5**,** r5**,** r2 // result += num  movs r3**,** #10  muls r5**,** r5**,** r3 // result \*= 10  adds r1**,** r1**,** 1 // update pointer(r1)  ldrb r2**,** **[**r1**]**  b atoi  push\_stack**:**  muls r5**,** r5**,** r4 // result \*= sign  push **{**r5**}**  adds r1**,** r1**,** 1 // update pointer(r1)  b strtok  isMinus**:**  ldrb r3**,** **[**r1**,**1**]**  cmp r3**,** #32 // is space?  beq minus\_op  movs r4**,** **-**1 // is not space: sign = -1  adds r1**,** r1**,** 1 // r1++  ldrb r2**,** **[**r1**]** // load byte  b atoi\_1  space**:** // update pointer(r1)  adds r1**,** r1**,** 1 // r1++  b strtok  add\_op**:**  pop **{**r6**}**  pop **{**r7**}**  adds r7**,** r7**,** r6 // top2+top1  push **{**r7**}**  adds r1**,** 1  b strtok  minus\_op**:**  pop **{**r6**}**  pop **{**r7**}**  subs r7**,** r7**,** r6 // top2-top1  push **{**r7**}**  adds r1**,** 1  b strtok |

1-2程式碼

|  |
| --- |
| **.**syntax unified  **.**cpu cortex**-**m4  **.**thumb  **.**data  result**:** **.**word 0  max\_size**:** **.**word 0  **.**text  **.**global main  m**:** **.**word 0x5E  n**:** **.**word 0x60  // r0: m  // r1: n  // r2: tmp  // r5: return value  // r7: max\_size  main**:**  ldr r0**,** **=**m  ldr r0**,** **[**r0**]**  ldr r1**,** **=**n  ldr r1**,** **[**r1**]**  push **{**r0**}**  push **{**r1**}**  movs r7**,** #0 // max\_size=0  bl GCD  ldr r2**,** **=**result  str r5**,** **[**r2**]** // return\_value  ldr r2**,** **=**max\_size  str r7**,** **[**r2**]** // // max\_size  end**:**  B end  GCD**:**  //TODO: Implement your GCD function  cmp r0**,** #0 // m==0?  beq m\_zero  cmp r1**,** #0 // n==0?  beq n\_zero  ands r2**,**r0**,**1 // m&1==0?  ands r3**,**r1**,**1 // n&1==0?  orrs r4**,**r2**,**r3 // !(m&1) && !(n&1)? ->!(m&1 | n&1)  cmp r4**,** #0 // !(m&1 | n&1)?  beq mn\_even  cmp r2**,** #0 // m&1==0?  beq m\_even  cmp r3**,** #0 // n&1==0?  beq n\_even  b **else**  m\_zero**:** // return n  movs r5**,** r1  bx lr  n\_zero**:** // return m  movs r5**,** r0  bx lr  mn\_even**:** // return 2\*gcd(m>>1, n>>1)  push **{**r0**}**  push **{**r1**}**  push **{**lr**}**  ADDS r7**,** #3 // max\_size += 3  lsr r0**,** 1 // m>>1  lsr r1**,** 1 // n>>1  bl GCD  movs r6**,** #2  muls r5**,** r5**,** r6 // return 2\*gcd(m>>1, n>>1)  pop **{**lr**}**  pop **{**r1**}**  pop **{**r0**}**  bx lr  m\_even**:** // return gcd(m>>1, n)  push **{**r0**}**  push **{**r1**}**  push **{**lr**}**  adds r7**,** #3 // max\_size += 3  lsr r0**,** 1 // m>>1  bl GCD  pop **{**lr**}**  pop **{**r1**}**  pop **{**r0**}**  bx lr  n\_even**:** // return gcd(m, n>>1)  push **{**r0**}**  push **{**r1**}**  push **{**lr**}**  adds r7**,** #3 // max\_size += 3  lsr r1**,** 1 // n>>1  bl GCD  pop **{**lr**}**  pop **{**r1**}**  pop **{**r0**}**  bx lr  **else:** // return gcd(abs(m-n), min(m,n))  push **{**r0**}**  push **{**r1**}**  push **{**lr**}**  adds r7**,** #3  cmp r1**,** r0 // n>=m? // return gcd(n-m, m)  bge swap  subs r0**,** r0**,** r1 // return gcd(m-n, n)  b else\_return  swap**:** // return gcd(n-m, m)  subs r6**,** r1**,** r0 // tmp = n-m  movs r1**,** r0 // n = m  movs r0**,** r6 // m = n-m  else\_return**:**  bl GCD  pop **{**lr**}**  pop **{**r1**}**  pop **{**r0**}**  bx lr |