Lab2 ARM Assembly II
實驗二 ARM Assembly II

Group7 0616231 彭世丞

2-1. Postfix arithmetic

1. 實驗目的

這個實驗想要實現基本加減法的運算。
首先讀取一個字串，區分字串中的各個字元是數字(正負數)、還是運算符號
如果是數字，push進stack；如果是運算符號，就pop出兩個數，作運算後再把結果存入stack中。

2. 實驗過程

首先，我先在網路上了解甚麼是postfix arithmetic
http://163.28.10.78/content/senior/computer/ks_ks/smart/datastruct/compute/compute.htm
看完這個網站後，對於要如何實作才有較清楚的輪廓。

以下是我對於完成postfix arithmetic的想法:

```
for( int i=0; i<strlen; i++ ){
  grep one character x
  if( x == space )    continue;
  else if( x == '+' ){                                          //plus_sign
          pop two numbers y, z,          y = y + z
          push y back to stack   continue;       }
  else if( x == '-' ){
          if( next_word == space ){                             //minus_sign
                  pop two numbers y, z,          y = y – z
                  push y back to stack   continue        }
          else   set negative_sign to 1   }                     //this number is negitive
  atoi
  check if negative_sign is 1 or not
  push this number to stack
  edit i to the right spot
}
```

想好之後，我便開始撰寫code

過程中，遇到了許多問題：

- 我發現要從字串中取出各個字元非常困難，查了許多資料後終於找到如果使用ldrb這個指令可以實現
  參考資料：https://blog.csdn.net/u013477200/article/details/50723555

- 在atoi中，我一開始計算完這個number的位數就把"指向字串現在進行到哪裡的指標r1"移動到number位置的後面一位(空格)。這樣移動當然是對而且必要的，但是因爲在後面計算number數值大小的loop時我不斷地使用到r1來定位我在字串的位子，我卻錯以爲r1還在number的起始位置，因而讓result出現error。
  → 後來我又額外設一個register去存number的位數，就解決了這個問題。

- 連續pop出兩個數字 (EX: pop {r6, r5}) 時，要小心兩個數的順序問題。像是在遇到減法時，是r6-r5, 不是r5-r6。我一開始就犯了這個錯，藉由逐行debug才找到這個錯誤。

- 最後在測試時，發現"1 -1 +"這組測資會出現unaligned opcodes detected in executable segment的Error訊息。上網查了好久好久……
  → 發現在.asciz的後面加一行.align 2就成功了
  參考資料：https://openssl-dev.openssl.narkive.com/B2M7Xl7u/openssl-org-2190-bug-aes-armv4-s-981-error-unaligned-opcodes-detected-in-executable-segment

2-2. Calculating Greatest Common Divisor

1. 實驗目的

這個實驗想要實現計算出兩數的最大公因數(Use Stein's Algorithm)。
簡單來說就是要用組合語言來實現Stein's Algorithm。
過程中，要利用stack來存取每一次loop的結果，以及存取lr(因爲會branch很多次，如果不存的話之前的lr值會遺失)

2. 實驗過程

我結合作業內容提供的Stein's Algorithm，初步的想法如下：

```
GCD:
  pop two words r1, r0
  push lr
  if ( m == 0 )     branch to   return_n
  else if ( n == 0 )     branch to    return_m
  else if ( m % 2 == 0 && n % 2 == 0 )    branch to   both_even
  else if ( m % 2 == 0 )     branch to    only_m_isEven
  else if ( n % 2 == 0 )     branch to    only_n_isEven
  else     branch to   final_return

return_n:          set result,   pop lr,   add times of recursive
return_m:          set result,   pop lr,   add times of recursive
```

```
both_even:        add times that result should*2, m = m/2, n = n/2,   push m n,   go to GCD
only_m_isEven:   m = m/2,   push m n,   go to GCD
only_n_isEven:    n = n/2,   push m n,   go to GCD
final_return:      cmp r2 r3,   set number to abs(r2-r3) and min(m,n), push m n, go to GCD
```

過程中，遇到的問題：

- 我參考作業說明的範例(如下)來了解push、pop、和從stack中如何ldr數值的方法，就很快地上了軌道。

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

- 我一開始忘記要記錄下，如果兩個數都是偶數並都除以2後，原本的result要乘以2的次數。
  → 設了變數來存放這個次數後，就解決了

- 原本我還以爲，bne和beq也會有修改lr值，害我寫的code變得好複雜，後來我上網查，發現這兩個指令就只有單純的branch而已，就變好寫很多了

- 這個小題我覺得最難的部分，就在於要決定在何時push lr，又要在何時pop lr。(因爲當你pop lr時，如果對寫的code很不清楚的話，會不知道它到底會跳去哪裡)
  → 我用紙和筆，確實畫下stack並填值跑流程，來確認自己寫的code每一個push和pop的lr都是正確的，沒有多餘或錯誤的lr存入。

- 關於max_size，其實我有想到如果不完全按照作業範例給的GCD Stein's Algotithm版本的簡化版，會讓max_size比我原本寫的值小，但我後來還是決定就照作業範例來實作。

<u>實驗心得與結語</u>

這次lab2的兩小題讓我學會了怎麼靈活的運用stack來push和pop資料，對於sp的移動方式也有更深一層的認識。

我發現，一開始的下筆都是最難的。但一旦開始進入打code的環節，就會自動的對各個難題抽絲剝繭，就算遇到無法處理的問題，至少還是有個方向，從那個方向一直google、google、再google，終究會找到解決辦法的。所以，希望我以後不要只是在想要怎麼寫、拖延進度，趕快下筆寫才是正解。