



Lab1 ARM Assembly I

實驗一 ARM Assembly I

1. Lab objectives 實驗目的

- Familiar with basic ARMv7 assembly language.
- In this Lab , we will learn the topics below.
 - How to use logic and arithmetic instructions.
 - How to use conditional branches to finish the loop.
 - How to use registers and basic function parameter passing.
 - How to access memory and array.
- 熟悉基本 ARMv7 組合語言語法使用。
- 在這次實驗中需要同學了解
 - 算數與邏輯操作指令使用
 - 如何利用條件跳躍指令完成程式迴圈的操作
 - 暫存器 (Register) 使用與基本函式參數傳遞
 - 記憶體與陣列存取

2. Lab principle 實驗原理

Please refer to the course ARMASM lecture slide and STM32L4x6 Reference manual.

請參考上課 ARMASM 講義與 STM32L4x6 Reference manual。





0101 0101 1010 1010 0000 0000
1010 1010 0101 0101 0000 0000
1111 1111 1111 1111 0000 0000

3. Steps 實驗步驟

3.1. Hamming distance 漢明距離

The Hamming distance between two integers is the number of positions at which the corresponding bits are different. Take 0xAB and 0xCD as an instance. Since 0xAB = 0b1010 1011, 0xCD = 0b1100 1101, and they have 4 different bits. The hamming distance between 0xAB and 0xCD is 4.

兩個整數之間的漢明距離是相應位不同的位置數。以 0xAB 和 0xCD 為例。由於 0xAB = 0b1010 1011, 0xCD = 0b1100 1101, 它們具有 4 個不同的位元。0xAB 和 0xCD 之間的漢明距離為 4。

Requirement: Please modify the code provided below, calculate the Hamming distance between two constants, and store the result in the variable "result".

請修改下面提供的範例碼，計算兩個常數間的漢明距離，並將結果存放至 result 變數中。

Reference:

https://en.wikipedia.org/wiki/Hamming_distance#Algorithm_example

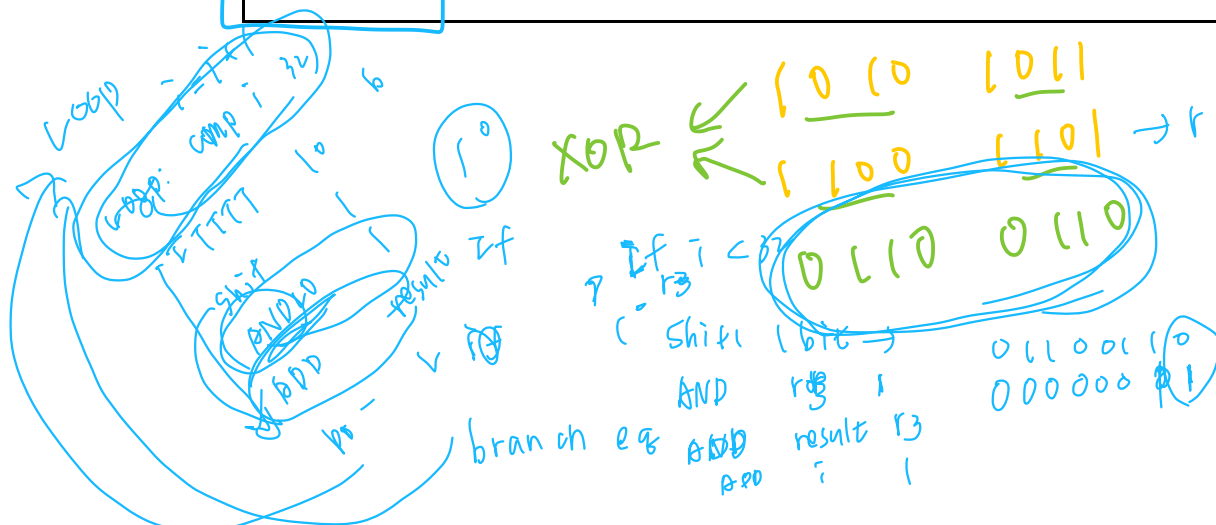
```
.data
    result: .byte 0
.text
.global main
.equ X, 0x55AA00
.equ Y, 0xAA5500

main:
    movs R0, #X //This line will cause an error. Why? Hint 2,
    movs R1, #Y
    ldr R2, =result
    bl hamm
L: b L

hamm:
    //TODO
    bx lr
```

会不会用 OR、AND

hamm 要把結果存回 result 的位置





3.2. Fibonacci serial 斐波那契數列

The original problem that Fibonacci investigated (in the year 1202) was about how fast rabbits could breed in ideal circumstances. Let F_n denote the number of newborn rabbit(s) at the n^{th} month under the ideal condition. F_n can be modeled as the following formula.

斐波那契調查的最初問題（在1202年）是關於兔子在理想情況下的繁殖速度。令 F_n 表示理想條件下第 n 個月新生兔的數量。 F_n 可以按以下公式建模。

$$F_0 = 0, F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}, n > 2$$

Reference: https://it.wikipedia.org/wiki/Successione_di_Fibonacci

Requirement: Please modify the code provided below, and implement a subroutine "fib" which accepts a parameter N("r0") where $0 \leq N \leq 100$ and it will store the the n^{th} Fibonacci Number into "r4". The value of "r4" should be interpreted as a signed integer. If the result of F_n overflows, set "r4" to "-2". If the value of N exceeds the boundary, set "r4" to "-1".

請修改下面提供的程式，並實現子程序 "fib"，該程序接受一個參數 N(r0)，其中 $0 \leq N \leq 100$ ，並將第 n 個斐波那契數 F_n 存儲到 "r4" 中。其中，"r4" 的值應解讀為有號整數。如果 F_n 的結果溢位，則將 "r4" 設為 "-2"。如果 N 的值超出範圍，則將 "r4" 設置為 "-1"。

```
.text
.global main
.equ N, 20
```

```
fib:
```

```
//TODO
```

```
bx lr
```

```
main:
```

```
movs R0, #N
```

```
bl fib
```

```
L: b L
```

branch

goto 和 function call 的差別

function call 要記回來的位置 \Rightarrow 用 bl
而 goto 不記回來的位置





3.3. Bubble sort 氣泡排序

Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order.

冒泡排序是一種簡單的排序算法，透過反覆遍歷串列，比較相鄰元素並對排序錯誤的元素進行置換。


Requirement: Please modify the code provided below and implement a subroutine "do_sort" that accepts a parameter "arr", which points to a list containing 8 unsigned characters (each occupy 1 byte), and it will do the in-place sorting (in descending order). Note that it is required to use "R0" passing the argument and show the sorting process step by step in the memory browser (or memory viewer).

請修改下面提供的代碼，並實現一個子程序 "do_sort"，該程序接受一個參數 "arr" 其指向包含 8 個無號字符（每個字符佔 1 個位元組）的列表，並將進行 in-place 排序（降序）。請注意必須使用 "R0" 傳遞引數，並在內存瀏覽器（或內存查看器）中逐步顯示排序過程。

```
.data
arr1: .byte 0x19, 0x34, 0x14, 0x32, 0x52, 0x23, 0x61, 0x29
arr2: .byte 0x18, 0x17, 0x33, 0x16, 0xFA, 0x20, 0x55, 0xAC
.text
.global main
do_sort:
    //TODO
    bx lr
main:
    ldr r0, =arr1
    bl do_sort
    ldr r0, =arr2
    bl do_sort
L: b L
```

Hint: The memory access may require the instructions that support byte-alignment, such as STRB, LDRB.

提示：記憶體存取需使用 byte alignment 指令，例如：STRB, LDRB。

* 

```
for i = 0 ~ 7
    for j = 0 ~ 7-i; j++
        if a[j] < a[j+1]
            temp = a[j]
            a[j] = a[j+1]
            a[j+1] = temp
```

是由指令的前4 bit (NIVLIZ) 所對應的條件碼/用來做 condition execution
表示前面的指令若有更新這些 bit 的指令才執行現在的指令, 可以減少一些 bit 的複雜, 判斷的公式

4. Question 實驗課問題

Question 1 (for lab 3-1): What is the IT-block for ARM Assembly? What are the condition code suffixes of ARM instruction? What is the difference between instruction ADD and ADDS?

ARM 組合語言中的 IT-block 指的是甚麼? ARM 指令的條件碼後綴是什麼? 指令 ADD 與 ADDS 之間有何差別?

→ 有沒有 S 的差別是, 是否更新 register 最前面的 4 個 condition bit (NIVLIZ), 沒有 S 就不更新, 這跟這樣設計 (有 S 要更新 resource)

Question 2 (for lab 3-2): How to detect overflow by software? That is, using an algorithm or logical operation to detect overflow.

如何通過軟體檢測溢位? 即使用演算法或是邏輯判斷的方式偵測溢位。

→ 1 的符號都是正的, 溢位的話原本的 sign bit 那也會變 1, 變負, 和 0 比, 由於溢位

Question 3 (for lab 3-2): Does ARMv7-M provide any hardware support on overflow detection?

ARMv7-M 是否在溢出檢測方面提供任何硬體支持?

→ register 前的 NIVLIZ 4 bit
negative zero carry overflow

5. Reference & Hint 參考資料與提示

Hint 1 (for lab 3-1): Given an 8-bit integer X. How do we use logic operations such as AND, OR, XOR, and SHIFT to perform bitwise masking, setting and negation?

For example, we can set bit-4 to 1, set bit-7 to (not bit-7), and mask all the bits of X except bit-4 ~ bit-7 to "0".

給定一個 8 位元的整數 X。我們如何通過 AND, OR, XOR, SHIFT 等邏輯運算來進行按位掩碼, 設置和取反?

例如, 我們可以將 bit-4 設置為 1, 將 bit-7 設置為 "not bit-7", 然後將 X 除第 4~7 位元外的所有位元清除為 "0".

X	set bit-4	negative bit-7	mask
0b 0xxx xxxx	0b 0xx1 xxxx	0b 1xx1 xxxx	0b 1xx1 0000
0b 1xxx xxxx	0b 1xx1 xxxx	0b 0xx1 xxxx	0b 0xx1 0000

(x: don't care.)

和 IT-block 有關

Hint 2 (for lab 3-1): When we try to use the "mov(s)" instruction to move a large constant into a register. During assembly, the assembler may raise errors. why? How to move a large constant into a register?

當我們嘗試使用 "mov(s)" 指令將大的常數值移入暫存器中時, 在組譯期間, 組譯器可能會引發錯誤。為什麼? 如何將大的常數值移入暫存器?

operand 2 的位元是 32 位

① 0~255 的數, 或是這些數 shift left 在 32-bit 以內的數

② 0x 00XY00XY 形式的常數 } XY

③ 0x XY00XY00 形式的常數 } 都是

④ 0x XYXYXYXY 形式的常數 } 16 位數字

operand 2 的數值表示方法不是直接放什麼就是什麼數值
→ 只有 12 bit

它用 rotate 之類的對應方法, 可以表示更大的數, 但有些數字表示不出來

在 mov 這些表示不出來的數字就會有警告

用 load 指令已處理完畢



Hint 3 (for lab 3-3): What happened to arr1 and arr2 when we substitute ".byte" to ".word"?

當我們將 ".byte" 替換成 ".word", arr1 跟 arr2 有什麼變化?

Hint 4 (for lab 3-3): Do we need to back up any registers before applying "do_sort" on "arr2"?

在對 "arr2" 進行 "do_sort" 之前, 我們是否需要備份任何暫存器?

問問題:

- ① =, #, data result: .byte 0
- ② error: target not available
- ③ 變址數列 N 寄存器(?) \Rightarrow mov 可能不行
- ④ 如果 mov 變址, 範圍? (見 hint 2)
- ⑤ debug 一次全 run 完
- ⑥ lab 交什麼?
- ⑦ 我的 debug 找不到 path
- ⑧ hint 3