

# Lab2 ARM Assembly I

0516076 施威綸

# 1. Lab objectives 實驗目的

Familiar with basic ARMv7 assembly language.

In this Lab, we will learn topics below.

- How to use conditional branch to finish the loop.
- How to use logic and arithmetic instructions.
- How to use registers and basic function parameter passing.
- How to access memory and array.

# 2. Steps 實驗步驟

## 2.1. Hamming distance

Calculate the Hamming distance of 2 half-word (2 bytes) numbers, and store the result into the variable "result"

```
1.data
 2
      result: .byte 0
 3
4.text
       .global main
       .equ X, 0x55AA
 7 .equ Y, 0xAA55
9 hamm:
      eor r0, r1
                          10: we xor 2 strings to figure out the difference
10
11
       lsr r1, r0, #1
                          12: load (0101 0101 0101 0101)2 into r2
       ldr r2, =0x5555
12
13
       and r0, r2
      and r1, r2
14
15
       add r0, r0, r1
16
                          18: load (0011 0011 0011 0011)2 into r2
17
       lsr r1, r0, #2
       ldr r2, =0x3333
18
       and r0, r2
19
20
       and r1, r2
21
22
       add r0, r0, r1
23
       lsr r1, r0, #4
       ldr r2, =0x0F0F
                          24: load (0000 1111 0000 1111)<sub>2</sub> into r2
24
       and r0, r2
25
26
       and r1, r2
27
```



```
28
       add r0, r0, r1
29
       lsr r1, r0, #8
       ldr r2, =0x00FF
30
                          30: load (0000 0000 1111 1111)2 into r2
      and r0, r2
31
       and r1, r2
32
33
34
       add r0, r0, r1
35
36
       bx lr
                          36: branch to caller
37
38 main:
39
       ldr r0, =X
40
       ldr r1, =Y
       bl hamm
41
42
       ldr r2, =result
43
       str r0, [r2]
44
45 L: b L
46
```

Hamming distance is basically using XOR to figure out the different "bits" of 2 numbers. The next step is to count the hamming weight, as well as the number of different bits with the result of XOR. By this algorithm, we easily came up with the number using AND & Right Shift to add counts in a tree pattern.

#### 2.2. Fibonacci serial

Declare a number N  $(1 \le N \le 100)$  and calculate the Fibonacci serial Fib(N). Store the result into register R4.

```
.syntax unified
       .cpu cortex-m4
 2
       .thumb
 3
 4.data
       arr: .space 400
 7 .text
       .global main
                           9: in this case, N = -3, which is invalid
 9
       .equ N, -3
10
11 fib:
12
       cmp r3, #100
13
       bgt exit
                           13: when N > 100, branch to exit
14
       cmp r3, #1
       blt exit
15
                           15: when N < 1, branch to exit
16
       add r0, r0, #4
17
18
       adds r4, r1, r2
                           18: r4 = fib(n), r1 = fib(n-2), r2 = fib(n-1)
19
                           20: IT block, return -2 when the result is
20
       itt vs
21
       movvs r4, #0
                               overflow
22
       subvs r4, r4, #2
23
```



```
24
      str r4, [r0]
25
                          26: move fib(n-1) to r1, fib(n) to r2, prepare
26
      mov r1, r2
                              for the next loop
      mov r2, r4
27
28
       sub r3, #1
29
30
       cmp r3, #0
       bne fib
31
32
       bx 1r
33
34
                          35: return -1 when N > 100 or N < 1
35 exit:
36
       mov r4, #0
       sub r4, r4, #1
37
38
       bx 1r
39
40
41 main:
42
      ldr r0, =arr
43
      mov r1, #1
44
      str r1, [r0]
45
      add r0, r0, #4
      mov r2, #1
46
47
      str r2, [r0]
48
      movs r3, #N
49
50
       sub r3, #2
51
      bl fib
52
53 L: b L
54
```

The function fib is actually a for loop which i = N. We added branches of returning -1 and returning -2 at the head of it.

The return value is in r4.

#### 2.3. Bubble sort

Implement the Bubble sort algorithm for the 8-byte data array with each element in 8 bits by assembly.

```
1    .syntax unified
2    .cpu cortex-m4
3    .thumb
4    .data
5     arr1:    .byte 0x19, 0x34, 0x14, 0x32, 0x52, 0x23, 0x61, 0x29
6     arr2:    .byte 0x18, 0x17, 0x33, 0x16, 0xFA, 0x20, 0x55, 0xAC
7
8    .text
9     .global main
10
```



```
11 do sort:
                            12: move the size of array into r1
 12
        mov r1, #8
 13
 14
        b outer_loop
 15
 16 outer_loop:
 17
        sub r1, r1, #1
 18
        cmp r1, #0
 19
        ittt ne
                            19: IT block, run inner loop if r1 != 0
 20
        movne r6, r0
                            20: r6 temporarily store the address of the
 21
        movne r2, r1
                                array
 22
 23
        bne inner_loop
 24
                            25: directly return to caller when outer loop is
 25
        bx 1r
                               done
26
 27 inner_loop:
 28
 29
        cmp r2, #0
 30
        beq outer_loop
 31
 32
        ldrb r3, [r6]
 33
        ldrb r4, [r6, #1]
 34
        cmp r3, r4
                            35: IT block, switch contents when arr[n] >
        ittt gt
 35
                               arr[n+1]
 36
        movgt r5, r3
 37
        movgt r3, r4
        movgt r4, r5
 38
 39
        strb r3, [r6]
        add r6, r6, #1
 40
                            41: store final r3, r4 to memory
        strb r4, [r6]
 41
 42
 43
       sub r2, r2, #1
 44
 45
        b inner loop
 46
 47 main:
 48
        ldr r0, =arr1
 49
        bl do sort
 50
        ldr r0, =arr2
        bl do_sort
 51
 52
 53 L: b L
```

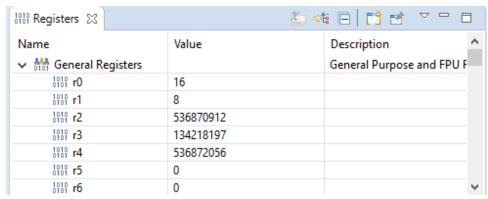
Bubble sort is an  $O(n^2)$  algorithm with doubly-nested loop.

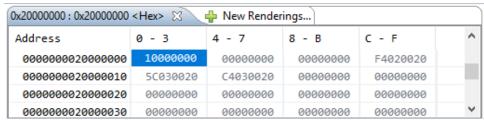
## 3. Results and analysis 實驗結果與分析

### 3.1. Hamming distance

The result was stored from r0 to *result* in memory (0x20000000).







This code will cause assembler error because MOV immediate data range from 0 to  $\pm 255$  (8 bits).

So we modified to this:

When the immediate value is too big, such as address, to be moved into a register, we can use the LDR pseudo instruction.

## 3.2. Fibonacci serial

In the case of N = 5, the result was stored at r4 = 5.

Name ✓ ∰ General Registers	Value	Description General Purpose and FPU F
1010 rO	536870928	
1010 r1	3	
1010 r2	5	
1010 r3	0	
1010 r4	5	

N = -3, invalid number

Nam	ne 👫 General Registers	Value	Description General Purpose and FPU F	^
	1010 rO	536870916		
	1010 r1	1		
	1010 r2	1		
	1010 r3	-5		
	1010 r4	-1		

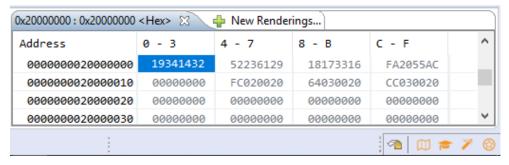
N = 47, overflow



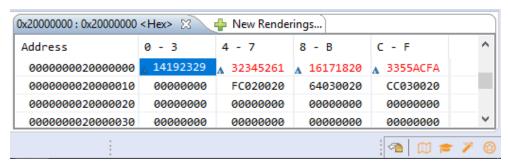
Name	Value	Description	ı
→ ₩ General Registers		General Purpose and FPU F	
1010 rO	536871096		ı
1010 r1	1836311903		ı
1010 r2	-2		ı
1010 r3	0		ı
1010 <b>r4</b>	-2		ı

### 3.3. Bubble sort

#### Before:



#### After:



The ARM structure store value in little-endian way, so we use the <Hex>rendering way

# 4. Conclusions and ideas 心得討論與應用聯想

In last semester, I have once written MIPS for the CO assignment. It literally took me a week to finish 3 codes. With the experience, I have more basic concept of assembly language now. Although there are subtle differences between MIPS and ARM, especially xPSR, the assignment was still extremely helpful.