p과제 4
마이크로프로세서응용
22100113 김성빈

Lab workbook Assembly 3

Branch Instruction(1)

- Jump (Branch)
  - Unconditional: B Label

```
main
  mov r0, (   1 ) ; limit
  mov r1, #1 ; index
   mov r2, #0 ; sum
loop
  cmp r0, r1 ; limit - index
  addGE r2, r2, r1 ; sum = sum + index
  addGE r1, r1, #1 ; index ++
  B loop
```

No	(1)	final r1	final r2
1	5	0x6	0xF
2	9	0xA	0x2D
3	10	0xB	0x37
4	20	0x15	0xD2

final의 의미는 더 이상 r2가 증가하지 않는
반복 횟수에 도달한 때

2

[Branch Instruction (1)]

Branch Instruction(2)

- Jump (Branch)
  - Conditional: B<cond> Label

```
; compare R0 and R1,
; and store R0 larger value, R1 smaller value
main
  mov r0, ( 1 ) ;
  mov r1, (  2 ) ;
  cmp r0, r1 ;
  movge r2, r0
  movge r3, r1
  Bge exit
  mov r2, r1
  mov r3, r0
exit
  mov r0, r2
  mov r1, r3
```

No	(1)	(2)	r2값	r3값
1	#10	#21	0x15	0xA
2	#100	#50	0x64	0x32
3	#-1	#10	0xA	0xFFFFFFF F
4	#-2	#-1	0xFFFF FFFF	0xFFFFFFF E

3

[Branch Instruction (2)]

Branch Instruction (3)

- Subroutine Call & Return
  - BL label
  - BL <cond> label

```
main
   MOV R1, #0x03
  MOV R2, R1, LSL #2
  BL func  ; call
  B end
func
  SUB R0, R1, R2
  MOV PC,LR   ; return

end
 ADD R0, R0, R1, LSL #3
```

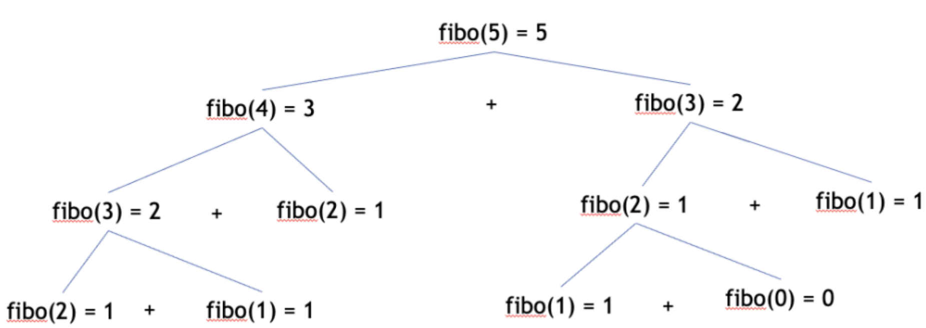
instruction	r14 (before)	r14 (after)	r15 (before)	r15 (after)
BL func	0x0	0xC	0x8	0x10
SUB R0,R1,R2	0xC	0xC	0x10	0x14
MOV PC,LR	0xC	0xC	0x14	0xC
B end	0xC	0xC	0xC	0x18

4

[Branch Instruction (3)]

Branch Instruction (4-1)

1. Complete the function call graph. For each call, show the return value (R3)



6

[Branch Instruction (4-1)]

Branch Instruction (4-2)

2. Find register values for each call by tracing , change First Line of code as MOV R0, #3

	r13(sp)	r14 (lr)	r15(pc)	r0	r1	r2	r3
fibo(3) call 직전 (main 내)	0xFF000000	0x0	0x4	0x3	0x0	0x0	0x0
fibo(2) call 직전 (fibo (3) 내)	0xFEFFFFF0	0x8	0x24	0x2	0x0	0x0	0x0
fibo(2) call 직후 (fibo(2) 내)	0xFEFFFFF0	0x28	0xC	0x2	0x0	0x0	0x0
1st fibo(1) call 직전 (fibo(2)내)	0xFEFFFFE0	0x28	0x24	0x1	0x0	0x0	0x0
1st fibo(1) return 직전 (fibo(1) 내)	0xFEFFFFE0	0x28	0x3C	0x1	0x0	0x0	0x1
1st fibo(1) return 직후 (fibo(2)내)	0xFEFFFFF0	0x28	0x28	0x1	0x0	0x0	0x1
1st fibo(0) call 직전 (fibo(2)내)	0xFEFFFFE0	0x28	0x30	0x0	0x0	0x1	0x1
1st fibo (0) call 직후 (fibo(0)내)	0xFEFFFFE0	0x34	0xC	0x0	0x0	0x1	0x1
1st fibi (0) return 직후 (fibo(2)내)	0xFEFFFFE0	0x34	0x34	0x0	0x0	0x1	0x0
2nd fibo(1) call 직전 (fibo(3) 내)	0xFEFFFFF0	0x28	0x30	0x1	0x0	0x1	0x1
2nd fibo (1) call 직후(fibo(1) 내)	0xFEFFFFF0	0x34	0xC	0x1	0x0	0x1	0x1
2nd fibo(1) return 직전 (fibo(1) 내)	0xFEFFFFF0	0x34	0x38	0x1	0x0	0x1	0x2
fibo(3) return 직전 (fibo(3) 내)	0xFF000000	0x8	0x3C	0x3	0x0	0x0	0x2

7

[Branch Instruction (4-2)]

Exercise-1

- 3개의 양의 정수가 R0, R1, R2에 저장되어 있다. 세개의 수들 중 짝수의 개수를 구해서 R3에 저장하는 subroutine Count_even 을 ARM assembly code를 작성하라.

Hint: 양의 정수는 LSb가 0이면 짝수로 판별된다.

```
main
    MOV R0,#0x1234
    MOV R1,#0x2345
    MOV R2, #0x7330
    BL COUNT_EVEN
    ADD R0, R0, R3
    ADD R1, R1, R3
    ADD R2, R2, R3
```

```
COUNT_EVEN
    BIC    R4, R4, #0xFFFFFFFF
    AND    R4, R0, #0x1
    CMP    R4, #0x0
    ADDEQ  R3, R3, #1

    BIC    R4, R4, #0xFFFFFFFF
    AND    R4, R1, #0x1
    CMP    R4, #0x0
    ADDEQ  R3, R3, #1

    BIC    R4, R4, #0xFFFFFFFF
    AND    R4, R2, #0x1
    CMP    R4, #0x0
    ADDEQ  R3, R3, #1

    MOV    PC, LR
```

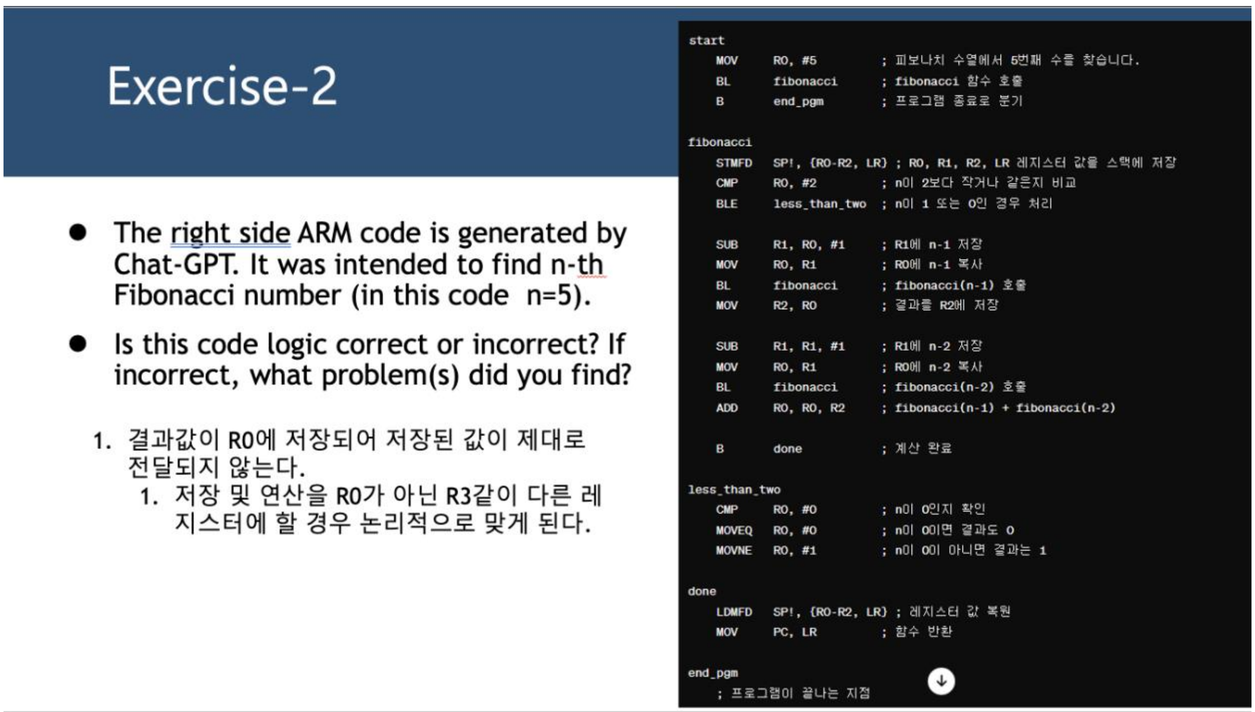
8

[Exercise-1]

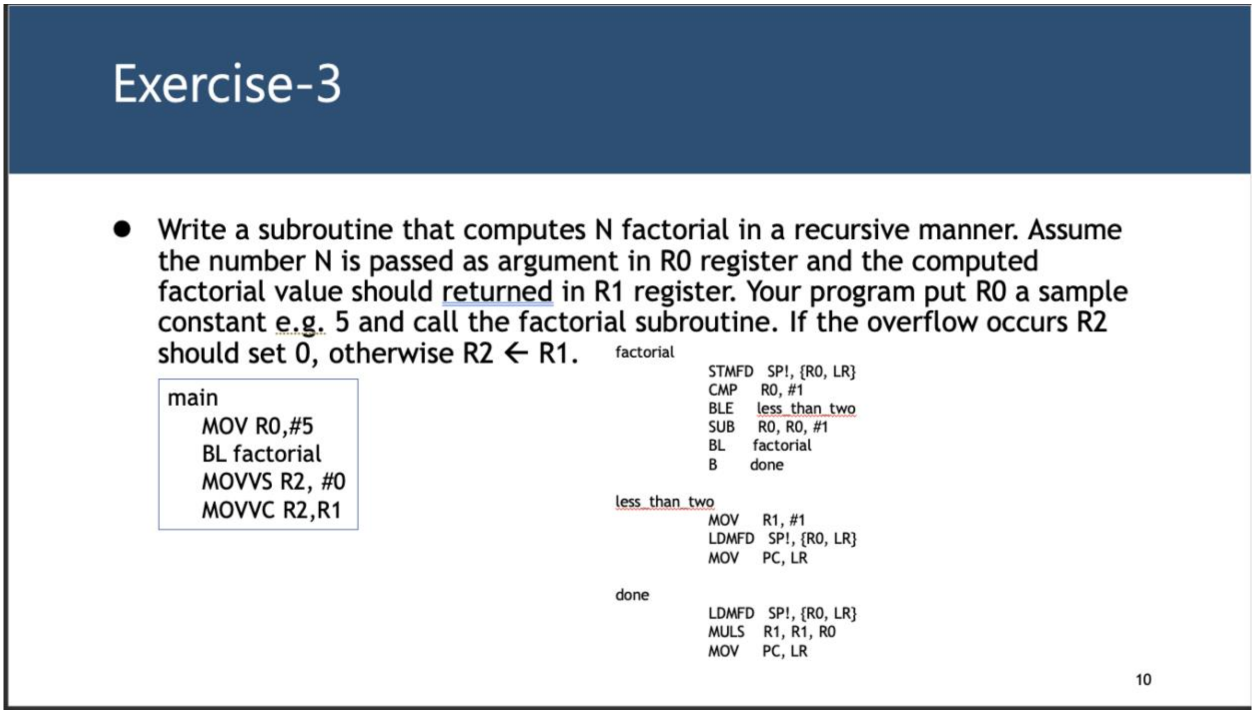
```
COUNT_EVEN
        BIC     R4, R4, #0xFFFFFFFF
        AND     R4, R0, #0x1
        CMP     R4, #0x0
```

```
        ADDEQ   R3, R3, #1
        BIC     R4, R4, #0xFFFFFFFF
        AND     R4, R1, #0x1
        CMP     R4, #0x0
        ADDEQ   R3, R3, #1
        BIC     R4, R4, #0xFFFFFFFF
        AND     R4, R2, #0x1
        CMP     R4, #0x0
        ADDEQ   R3, R3, #1
        MOV     PC, LR
```

[Exercise-1-code]



[Exercise-2]



[Exercise-3]

```
factorial
        STMFD   SP!, {R0, LR}
        CMP     R0, #1
        BLE     less_than_two
        SUB     R0, R0, #1
        BL      factorial
        B       done
less_than_two
        MOV     R1, #1
        LDMFD   SP!, {R0, LR}
        MOV     PC, LR

done
        LDMFD   SP!, {R0, LR}
        MULS    R1, R1, R0
        MOV     PC, LR
```

[Exercise-3-code]

```
.section .data


fibo_cache:
  .word -1, -1, -1, -1, -1, -1, -1, -1


.section .text
```

```
.global _start

_start:

    MOV R0, #6
    MOV R6, #0x4
    LDR R1, =fibo_cache

    BL fibo
    B end

fibo:
    STMFD SP!, {R0, LR}
    MUL R7, R0, R6 // R7 = R0 * 0x4
    LDR R2, [R1, R7] // R2 = R1[R0]

    CMP R2, #-1 // check if uninitialized
    BNE done // if initialized, return R1[R0]

    CMP R0, #1 // base case, n <= 1
    MUL R7, R0, R6 // R7 = R0 * 0x4
    STRLE R0, [R1, R7] // R1[R0] = R0
    BLE done

    MOV R8, R0 // store n for mem[n] = fibo(n-1) + fibo(n-2)
    STMFD SP!, {R8}

    SUB R0, R0, #1 // fibo(n-1)
    BL fibo
    MOV R4, R3

    SUB R0, R0, #1 // fibo(n-2)
    BL fibo

    ADD R5, R4, R3 // fibo(n-1) + fibo(n-2)
    LDMFD SP!, {R8}
    MUL R7, R8, R6 // R7 = R8 * 0x4
```

```
    STR R5, [R1, R7] // mem[n] = fibo(n-1) + fibo(n-2)

    B done

done: // return mem[n], R3 = mem[n]
    LDMFD SP!, {R0, LR}
    MUL R7, R0, R6 // R7 = R0 * 0x4
    LDR R3, [R1, R7] // return (R3 = R1[R0]), result stored in R3
    MOV PC, LR

end:
```

[Exercise-4-BONUS-code]