

# 어셈블리 프로그래밍 설계 및 실습

실험제목: Data\_transfer\_to\_or\_from\_Mem

실험일자: 2020 년 09 월 15 일 (화)

제출일자: 2020 년 09 월 21 일 (월)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

학 번: 2019202052

성 명: 김 호 성

## 1. 제목및목적

### A. 제목

1. problem 1
2. problem 2

### B. 목적

- 조건부 실행을 할 수 있다.
- 원하는 데이터를 메모리에 저장 및 불러오기가 가능하다.

## 2. 설계 (Design)

### A. Pseudo code

1.

MOV R[0]  $\leftarrow$  #1, R[1]  $\leftarrow$  #10, R[2]  $\leftarrow$  #15, R[3]  $\leftarrow$  #10

LDR R[4]  $\leftarrow$  TEMPADDR12

STRB R[4]  $\leftarrow$  R[0], R[4] = R[4] + 1

STRB R[4]  $\leftarrow$  R[1], R[4] = R[4] + 1

STRB R[4]  $\leftarrow$  R[2],

LDRB R[0]  $\leftarrow$  R[4, #-2]! ; R[0] = #1

CMP R[0], R[3]

MOVMI R[5]  $\leftarrow$  #2

MOVGT R[5]  $\leftarrow$  #1

MOVEQ R[5]  $\leftarrow$  #3

LDRB R[0]  $\leftarrow$  R[4, #1]!; R[0] = #10

MOVMI R[5]  $\leftarrow$  #2

MOVGT R[5]  $\leftarrow$  #1

MOVEQ R[5]  $\leftarrow$  #3

LDRB R[0]  $\leftarrow$  R[4, #1] ; R[0] = #15

MOVMI R[5]  $\leftarrow$  #2

MOVGT R[5]  $\leftarrow$  #1

MOVEQ R[5]  $\leftarrow$  #3

TEMPADDR12  $\leftarrow$  0x40000

MOV PC  $\leftarrow$  LR

END

2.

MOV R[0]  $\leftarrow$  #1, MOV R[1]  $\leftarrow$  #2, MOV R[2]  $\leftarrow$  #3, MOV R[3]  $\leftarrow$  #4

LDR R[4]  $\leftarrow$  TEMPADDR1

LDR R[6]  $\leftarrow$  TEMPADDR2

STRB R[4]  $\leftarrow$  R[3], R[4] = R[4] + 1

STRB R[4]  $\leftarrow$  R[2], R[4] = R[4] + 1

STRB R[4]  $\leftarrow$  R[1], R[4] = R[4] + 1

STRB R[4]  $\leftarrow$  R[0]

STRB R[6]  $\leftarrow$  R[0], R[6] = R[6] + 1

STRB R[6]  $\leftarrow$  R[1], R[6] = R[6] + 1

STRB R[6]  $\leftarrow$  R[2], R[6] = R[6] + 1

STRB R[6]  $\leftarrow$  R[3], R[6] = R[6] + 1

TEMPADDR1  $\leftarrow$  0x40000

TEMPADDR2  $\leftarrow$  0x40200

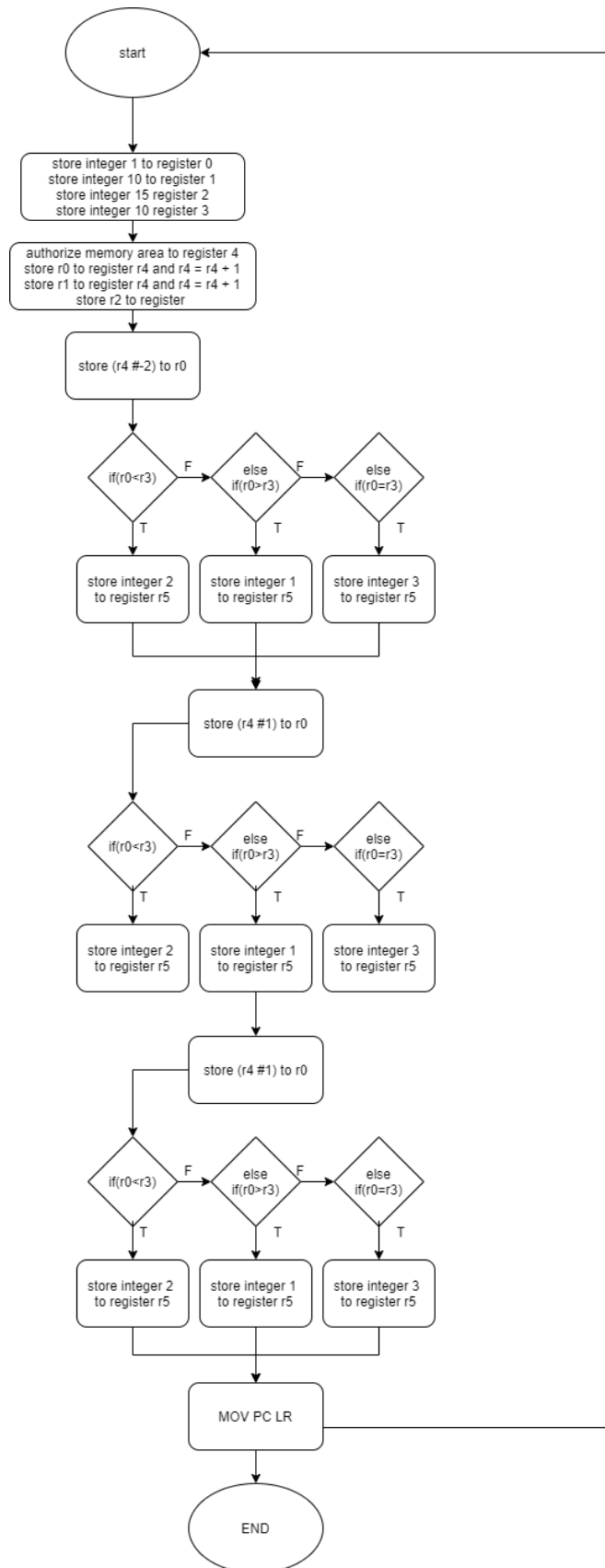
LDR R[5]  $\leftarrow$  R[6, #-3]!

LDR R[6]  $\leftarrow$  R[4, #-3]!

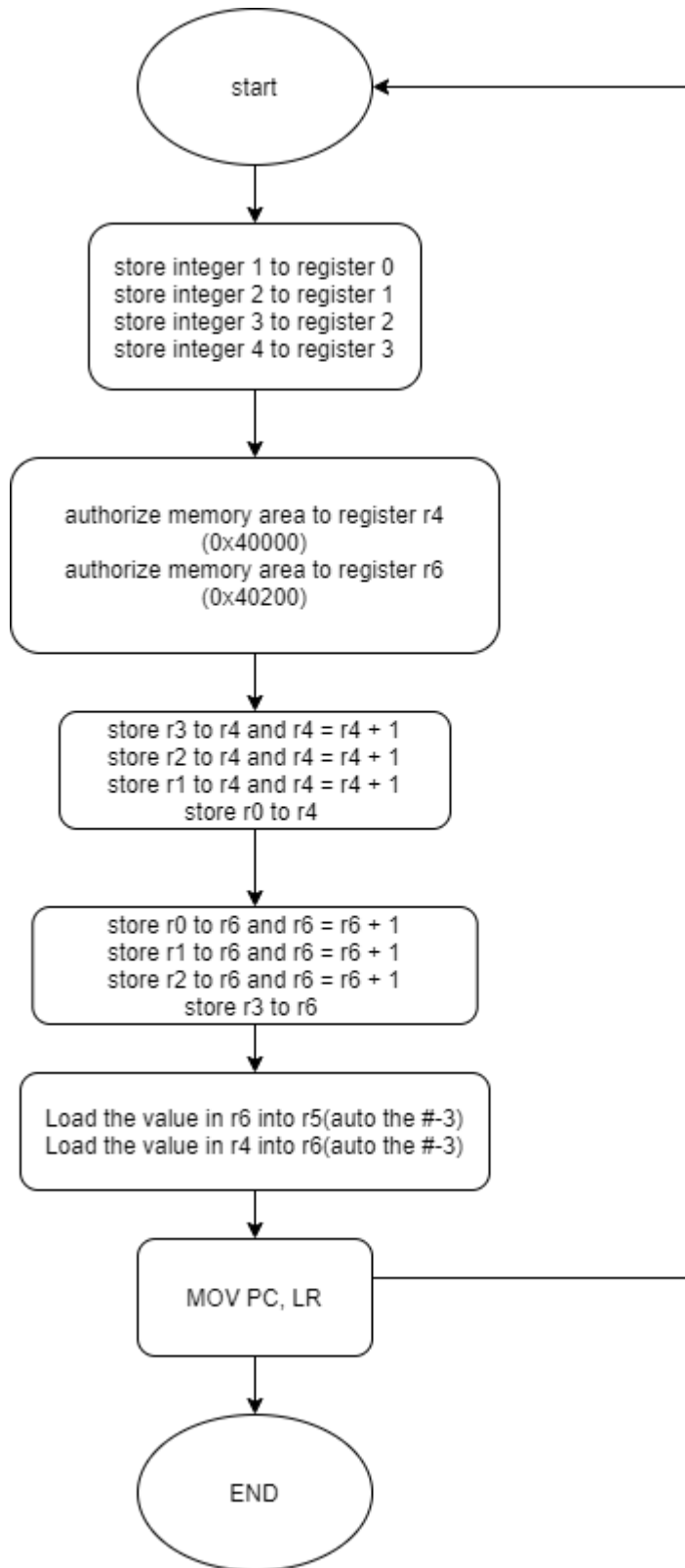
MOV PC  $\leftarrow$  LR

## B. Flow chart 작성

1.



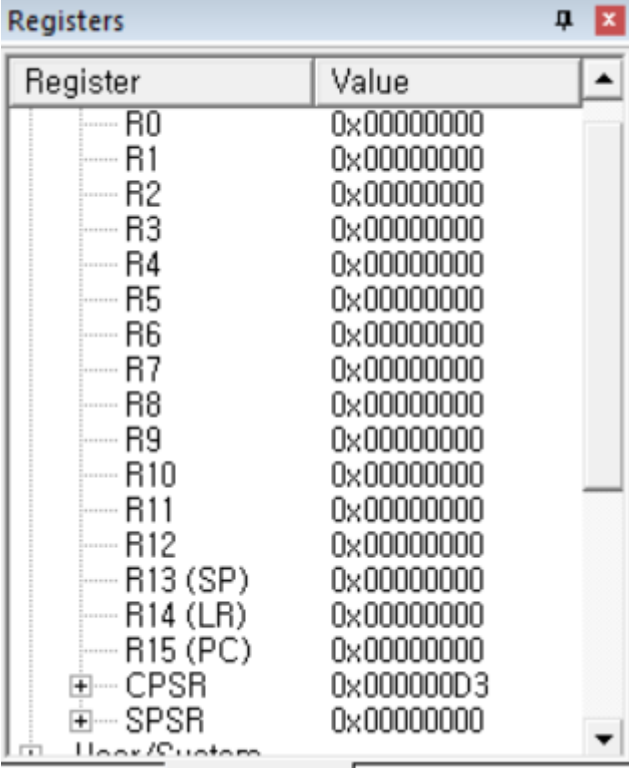
2.



### C. Result

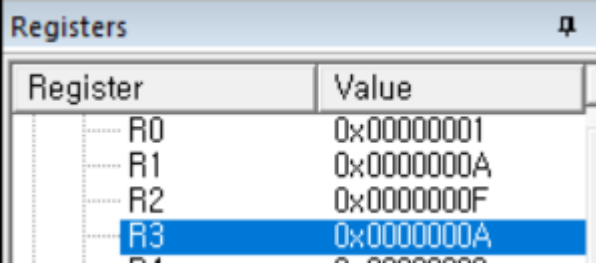
1.

초기 세팅



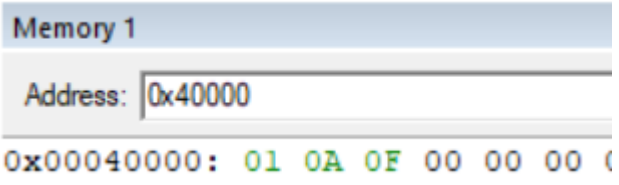
Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000

MOV R[0] ~ R[3]



Register	Value
R0	0x00000001
R1	0x0000000A
R2	0x0000000F
R3	0x0000000A

Store R[0] ~ R[3] to memory(0x40000)



Memory 1	
Address:	0x40000
0x00040000:	01 0A 0F 00 00 00 00 00

Compare r0 with r3 and store the value in r5 for the condition  
(r0 = 1)

Registers	
Register	Value
R0	0x00000001
R1	0x0000000A
R2	0x0000000F
R3	0x0000000A
R4	0x00040000
R5	0x00000002

Compare r0 with r3 and store the value in r5 for the condition  
(r0 = 10)

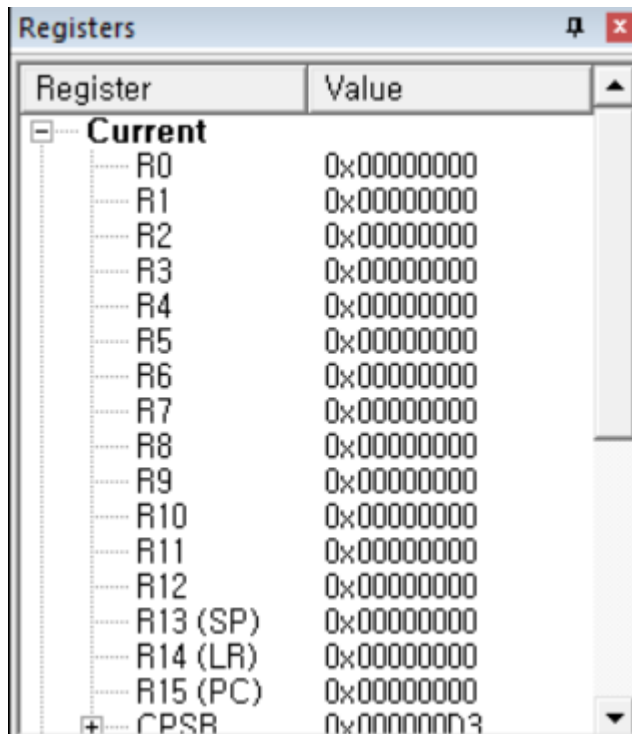
Registers	
Register	Value
R0	0x0000000A
R1	0x0000000A
R2	0x0000000F
R3	0x0000000A
R4	0x00040001
R5	0x00000003

Compare r0 with r3 and store the value in r5 for the condition  
(r0 = 15)

Register	Value
R0	0x0000000F
R1	0x0000000A
R2	0x0000000F
R3	0x0000000A
R4	0x00040001
R5	0x00000001

2.

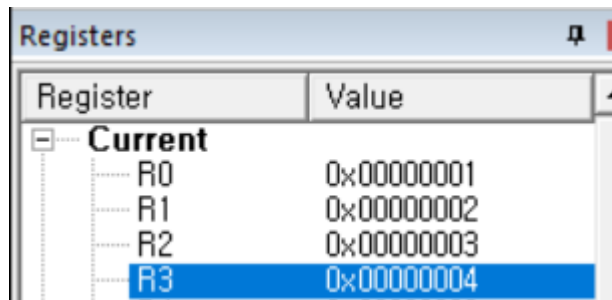
초기 세팅



A screenshot of a 'Registers' window from a debugger. The window has a title bar with a maximize button and a close button. It contains a table with two columns: 'Register' and 'Value'. The table is titled 'Current' with a minus sign icon. It lists registers R0 through R15 (PC) and CPSR. All registers have a value of 0x00000000. The CPSR register has a value of 0x00000003. A plus sign icon is visible at the bottom left of the table.

Register	Value
<b>Current</b>	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x00000003

MOV R[0] ~ R[3]



A screenshot of the 'Registers' window after the MOV instruction. The values for R0, R1, R2, and R3 are now 0x00000001, 0x00000002, 0x00000003, and 0x00000004 respectively. R3 is highlighted in blue. The other registers remain at 0x00000000.

Register	Value
<b>Current</b>	
R0	0x00000001
R1	0x00000002
R2	0x00000003
R3	0x00000004
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x00000000

authorize memory area to use (r4 = 0x40000, r6 = 0x40200)

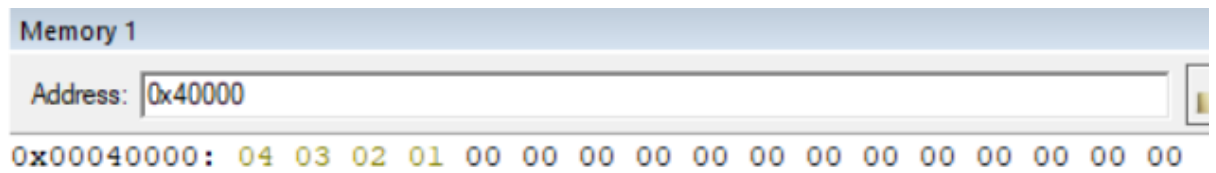


A screenshot of the 'Registers' window showing the memory authorization step. R4 is 0x00040000, R5 is 0x00000000, and R6 is 0x00040200. R6 is highlighted in blue.

R4	0x00040000
R5	0x00000000
R6	0x00040200



Store R[3] ~ R[0] to memory(0x40000)



Store R[0] ~ R[3] to memory(0x40200)

0x000401FE: 00 00 01 02 03 04

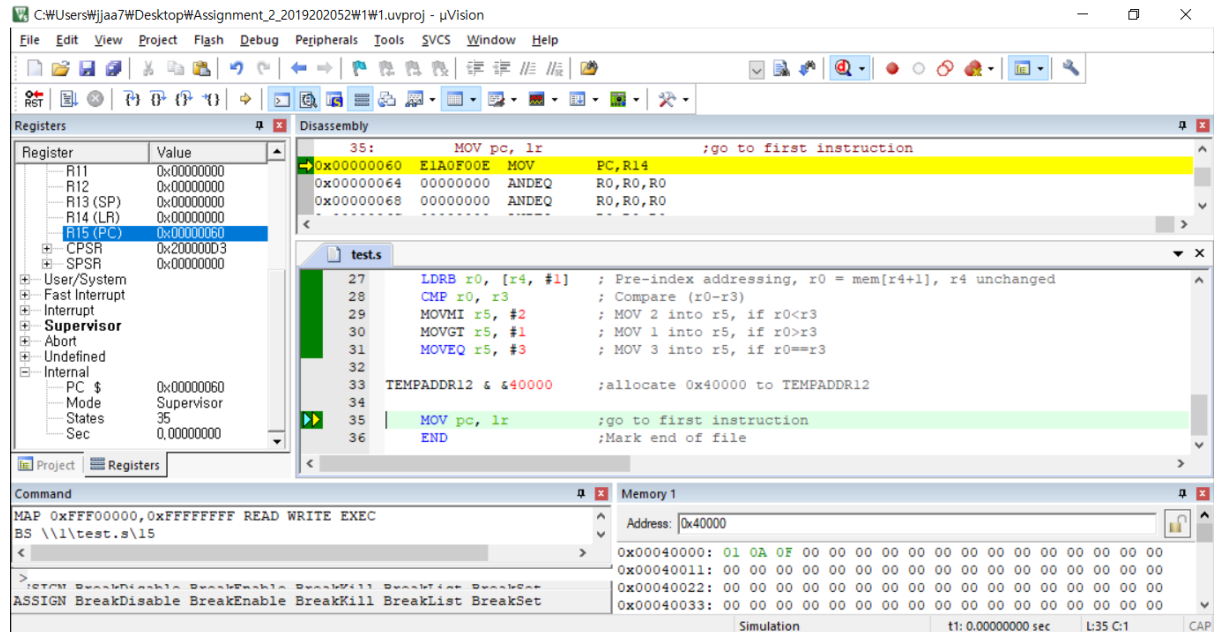
Load R[6, #-3]! to R5(auto) and Load R[4, #-3]! to R[6]

R4	0x00040000
R5	0x04030201
R6	0x01020304
R7	0x00000000

## D. Performance

1.

Performance = Code size \* Code size \* states =



State = 35

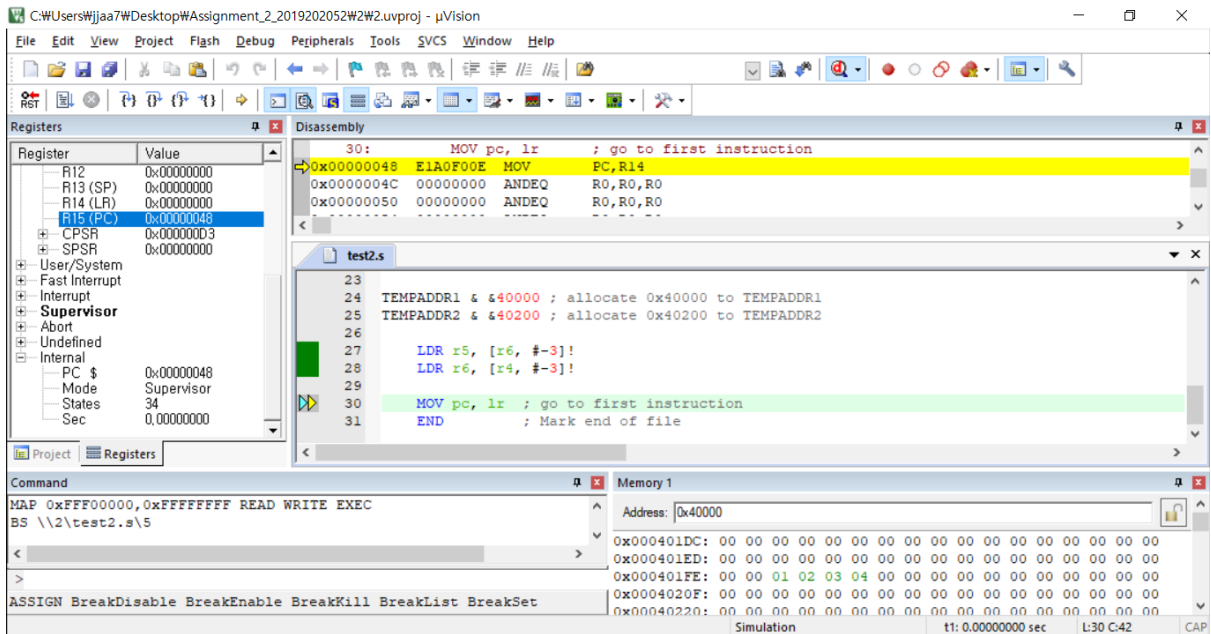
=====  
Total RO Size (Code + RO Data) 100 ( 0.10kB)  
Total RW Size (RW Data + ZI Data) 0 ( 0.00kB)  
Total ROM Size (Code + RO Data + RW Data) 100 ( 0.10kB)  
=====

Code size = 100

Performance = 350000

2.

Performance = Code size \* Code size \* states



=

State = 34

=====

Total RO Size (Code + RO Data)	76 ( 0.07kB)
Total RW Size (RW Data + ZI Data)	0 ( 0.00kB)
Total ROM Size (Code + RO Data + RW Data)	76 ( 0.07kB)

=====

Code size = 76

Performance = 196,384

### 3. 고찰 및 결론

#### A. 고찰

이번 실습을 통해서 두 개의 값을 비교하여 조건문을 연산하는 것과, 메모리로 데이터 저장 (store) 및 가져오기(load)에 대해 배울 수 있었다. 첫 번째 문제는 메모리에 저장된 숫자 3개를 STRB를 활용하여 지정된 memory에 값을 저장하였다. Memory에 저장된 값을 차례로 LDRB 명령어를 통해 값을 읽어 명령어 CMP를 이용하여 0x0A와 값을 비교하여 flag를 update 할 수 있도록 하였다. Update된 flag 값에 따라 MOV가 작동할 수 있도록 conditional field인 GT, EQ, LT를 이용하였다. 두 번째 문제는 명령어 MOV를 이용하여 각 register에 값을 저장한 후, r4 register에 메모리 주소 값을 저장하였다. 명령어 STRB를 활용하여 memory에 각각 r0, r1, r2, r3에 저장된 값을 저장할 수 있게 하였다. LDR를 통하여 r4 - 4 주소 값에 저장되어 있는 값을 register r5로 저장할 수 있게 하였다. 그리고 r0, r1, r2, r3의 값을 다른 memory에 거

꾸로 store하여 거꾸로 load하여 r6에 값을 저장하였다.

## B. 결론

- LDR: Load의 명령어로 첫 번째 인자는 register, 두 번째 인자는 주소로 쓰이게 된다.

- STR: Store의 명령어로 첫 번째 인자는 register, 두 번째 인자는 주소로 쓰이게 된다.

LDR과 STR 명령어는 ini파일을 통해 사용하고자 하는 메모리 영역에 read 또는 write 권한을 부여해야한다.

- MOV pc, lr  
코드의 진행을 처음으로 돌리는 역할이다.

## 4. 참고문헌

이준환교수님/어셈블리설계및실습/광운대학교(컴퓨터정보공학부)/2020