

어셈블리프로그래밍설계및실습 보고서

과제 주차: 2주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 6, 7 목 5

학 번: 2019202021

성 명: 정성엽

제 출 일: 2022.10.03(월)

1. Problem Statement

어셈블리어의 기본 명령어를 이용하여 프로그래밍을 익히고 어셈블리어에서 N, Z, C, V flag를 이용한 조건부 실행을 익힌다. 레지스터에 저장되어 있는 데이터를 메모리로 저장하거나 메모리에 있는 데이터를 레지스터로 불러오는 방법을 실습하고 익힐 수 있도록 한다.

2. Design

A. 사용하는 Assembly 명령어

- i. MOV r0, r1 : r0 레지스터에 r1의 값 대입
- ii. LDR r0, [r1] : r0 레지스터에 r1의 메모리로부터 불러오기
- iii. STRB r0, [r1] : r1 메모리에 r0의 값을 저장한다.

B. 비교 연산

- i. flag에는 N, Z, C, V가 있으며 각 조건에 따라 조건부로 업데이트 된다.
Ex) CMP r0, r1 이면 r0-r1의 값을 플래그 업데이트한다.
- ii. N: r0-r1 연산 후 음수인 경우 1로 업데이트 한다. – Negative flag
- iii. C: r0-r1 연산 후 양수인 경우 1로 업데이트 한다. – Carry flag
- iv. Z: r0-r1 연산 후 0인 경우 1로 업데이트 한다. – Zero flag
- v. V: Overflow가 발생하면 1로 업데이트 한다. – Overflow flag

C. 조건부 실행

- i. 위 CMP 명령어를 이용해 비교한 이후에는 업데이트된 플래그를 통해 조건부 실행을 구현할 수 있다. 앞선 명령어에 conditional field를 추가하여 조건부 실행을 할 수 있다.
- ii. EQ: Z가 1로 업데이트 되었을 때, 즉 수가 같을 때 실행한다.
- iii. LT: N과 V가 다를 때, 즉 비교했을 때 우항이 더 큰 경우 실행된다.
- iv. GT: Z가 업데이트 되지 않고, N과 V가 같을 때, 즉 좌항이 더 큰 경우 실행된다.

위 3가지 conditional field 외에도 더 많이 있지만 이번 과제에서는 위 3개를 이용해 수를 비교하고 조건부 실행한다.

D. .ini 파일을 이용한 Memory map 지정

- i. .ini 파일을 통해 메모리의 READ, WRITE, EXECUTE 권한을 설정할 수 있다.
- ii. 메모리의 경우 LDR, STR 등의 명령어를 통해 읽고 쓸 수 있으며 기본적으로는

Byte 단위로 접근하여 읽을 수 있고 위 명령어 뒤에 B를 추가하여 LDRB, STRB를 이용해 사용한다면 1Byte 단위로 읽고 쓸 수 있다.

- iii. 메모리에 레지스터 값을 쓴다면 little-endian 방식 때문에 반대로 순서로 저장된다.

E. 과제 수행

i. Problem 1

- 무작위 수를 R0~R2에 저장하고 R12에 메모리 주소를 지정하여 메모리에 숫자 3개를 저장한다.
- 후에 LDRB를 이용하여 1바이트 단위로 읽어서 가져온 데이터의 값을 r4에 저장한다.
- CMP 명령어를 이용해 불러온 데이터와 0x0A의 값을 비교하여 클 경우 1, 작을 경우 2, 같을 경우 3을 r5에 저장한다.

ii. Problem 2

- R0, R1, R2, R3 레지스터에 각각 1, 2, 3, 4를 저장한다.
- R12에 메모리를 지정하고 후에 little-endian 방식 때문에 반대 순서로 불러옴을 주의하여 메모리에 각 레지스터 값을 저장한다.
- LDR 명령을 통해 메모리에 저장된 값을 R5, R6에 저장하여 주어진 조건으로 저장되었는지 확인한다.

3. Conclusion

A. Problem 1

i. 코드 구현

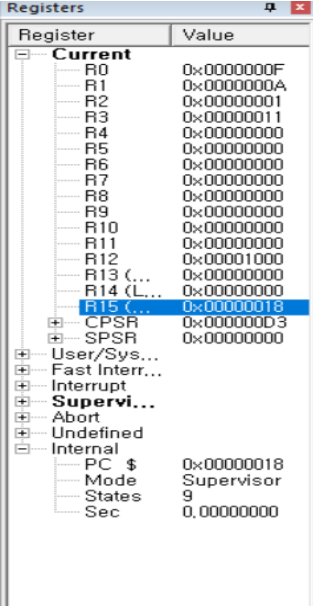
```
Problem1.s
1  AREA ARMex, CODE, READONLY
2  ENTRY
3  start
4      ;set random value
5      MOV r0, #15
6      MOV r1, #10
7      MOV r2, #1
8      ;set memory address and set initial value
9      LDR r12,TEMPADDR1
10     MOV r3, #17
11     STRB r3,[r12,#0]
12     ;store R0~R2 register value to each byte
13     STRB r0,[r12,#0]
14     STRB r1,[r12,#1]
15     STRB r2,[r12,#2]
16     ;load from memory into register r4
17     LDRB r4,[r12],#1
18     ;compare r4 and decimal 10
19     CMP r4,#10
20     MOVEQ R5, #3;if equal, r5=3
21     MOVLTI R5, #2;if "<", r5=2
22     MOVGTI R5, #1;if ">", r5=1
23     ;load from memory into register r4
24     LDRB r4,[r12],#1
25     ;compare r4 and decimal 10
26     CMP r4,#10
27     MOVEQ R5, #3;if equal, r5=3
28     MOVLTI R5, #2;if "<", r5=2
29     MOVGTI R5, #1;if ">", r5=1
30     ;load from memory into register r4
31     LDRB r4,[r12]
32     ;compare r4 and decimal 10
33     CMP r4,#10
34     MOVEQ R5, #3;if equal, r5=3
35     MOVLTI R5, #2;if "<", r5=2
36     MOVGTI R5, #1;if ">", r5=1
37     ;set TEMPADDR1's address
38     TEMPADDR1 & 400001000
39
40     END
```

ii. 코드 사이즈

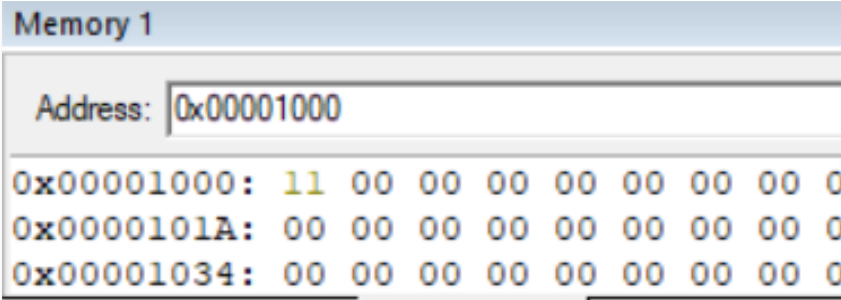
```
Build Output
assembling Probleml.s...
linking...
Program Size: Code=100 RO-data=0 RW-data=0 ZI-data=0
".\Objects\probleml.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

코드 사이즈는 100임을 확인할 수 있다.

iii. 무작위 숫자 3개 레지스터에 저장 및 메모리 지정 후 최초로 메모리 저장되는 값 임의 저장

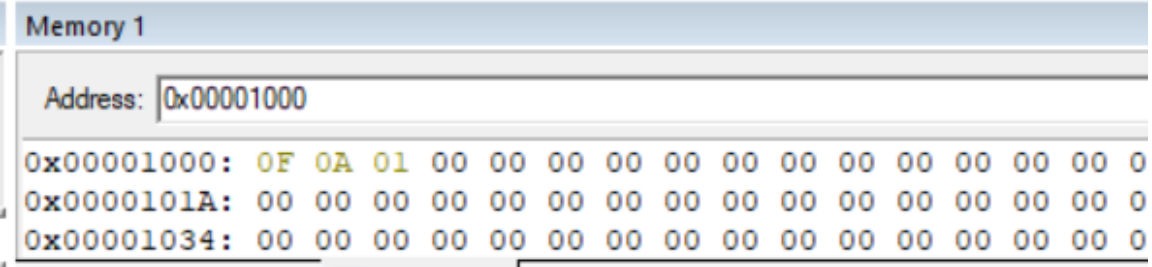


Registers window showing R15 (Current) with value 0x00000018. Other registers R0-R14 are 0x00000000. CPSR is 0x000000D3, SPSR is 0x00000000. PC \$ is 0x00000018, Mode is Supervisor, States is 9, Sec is 0.00000000.



Memory 1 window showing address 0x00001000 with value 11. Other addresses 0x0000101A and 0x00001034 show 00.

iv. 임의의 숫자 3개 메모리에 저장



Memory 1 window showing address 0x00001000 with value 0F 0A 01. Other addresses 0x0000101A and 0x00001034 show 00.

v. 첫번째 메모리에 저장된 숫자 불러온 후 #10과 비교

Registers	
Register	Value
Current	
R0	0x0000000F
R1	0x0000000A
R2	0x00000001
R3	0x00000011
R4	0x0000000F
R5	0x00000001
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00001001
R13 (...)	0x00000000
R14 (L...	0x00000000
R15 (...)	0x00000038
+ CPSR	0x200000D3
+ SPSR	0x00000000
+ User/Sys...	
+ Fast Interr...	
+ Interrupt	
+ Supervi...	
+ Abort	
+ Undefined	
- Internal	
PC \$	0x00000038
Mode	Supervisor
States	22
Sec	0.00000000

R0에 저장되었던 수(메모리에서 불러와 R4에 저장된 수)와 #10을 비교하였고 R0가 #16이므로 더 크다. 그러므로 R5에는 정상적으로 1의 값이 저장되었다.

vi. 두 번째 메모리에 저장된 숫자 불러온 후 #10과 비교

Current	
R0	0x0000000F
R1	0x0000000A
R2	0x00000001
R3	0x00000011
R4	0x0000000A
R5	0x00000003
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00001002
R13 (...)	0x00000000
R14 (L...	0x00000000
R15 (...)	0x0000004C
+ CPSR	0x600000D3
+ SPSR	0x00000000

R1에 저장되었던 수(메모리에서 불러와 R4에 저장된 수)와 #10을 비교하였고 R1이 #10이므로 같다. 그러므로 R5에는 정상적으로 3의 값이 저장되었다.

vii. 세 번째 메모리에 저장된 숫자 불러온 후 #10과 비교

Register	Value
Current	
R0	0x0000000F
R1	0x0000000A
R2	0x00000001
R3	0x00000011
R4	0x00000001
R5	0x00000002
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00001002
R13 (...)	0x00000000
R14 (L...	0x00000000
R15 (...)	0x00040404
CPSR	0x800000D3
SPSR	0x00000000
User/Sys...	

R2에 저장되었던 수(메모리에서 불러와 R4에 저장된 수)와 #10을 비교하였고 R1이 #1이므로 더 작다. 그러므로 R5에는 정상적으로 2의 값이 저장되었다.

B. Problem 2

i. 코드 구현

```

1  AREA ARMex, CODE, READONLY
2  ENTRY
3  start
4      ;Set Register value
5      MOV r0, #1
6      MOV r1, #2
7      MOV r2, #3
8      MOV r3, #4
9      ;Load Memory address
10     LDR r12, TEMPADDR1
11     ;store register values for memory sequence by little-endian
12     STRB r0, [r12, #0]
13     STRB r1, [r12, #1]
14     STRB r2, [r12, #2]
15     STRB r3, [r12, #3]
16     ;load from memory to register r5
17     LDR r5, [r12]
18     ;store register values for memory sequence by little-endian
19     STRB r0, [r12, #3]
20     STRB r1, [r12, #2]
21     STRB r2, [r12, #1]
22     STRB r3, [r12, #0]
23     ;load from memory to register r6
24     LDR r6, [r12]
25
26     ;set memory address
27     TEMPADDR1 & 40000000
28
29     END

```

ii. 코드 사이즈

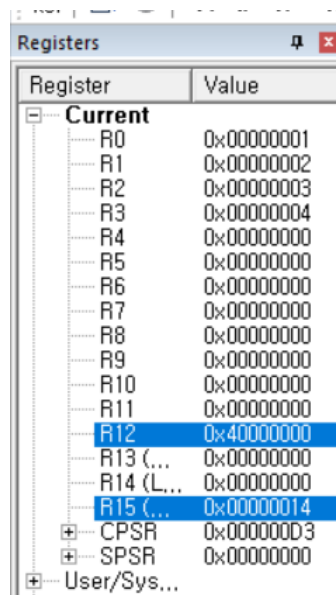
```

Build Output
linking...
Program Size: Code=64 RO-data=0 RW-data=0 ZI-data=0
".\Objects\Problem2.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
<

```

코드 사이즈가 64임을 확인할 수 있다.

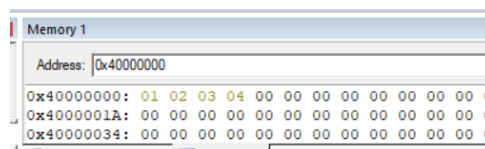
- iii. R0~R3 레지스터에 주어진 값 저장 및 메모리 주소 설정



Register	Value
Current	
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	0x40000000
R13 (...)	0x00000000
R14 (L...	0x00000000
R15 (...)	0x00000014
CPSR	0x00000003
SPSR	0x00000000
User/Sys...	

R0~R3 레지스터에 각각 1~4가 저장됨을 확인할 수 있고 R12 레지스터에 0x40000000 메모리 주소가 지정되어 있음을 확인할 수 있다.

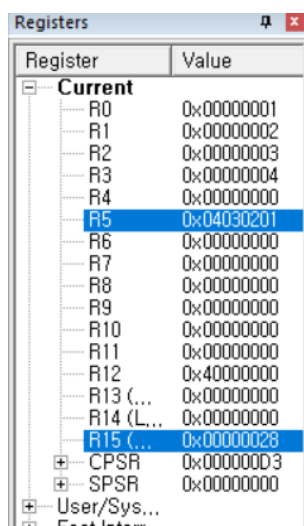
- iv. 메모리에 원하는 순서로 바이트 단위로 저장



Memory 1	
Address: 0x40000000	
0x40000000:	01 02 03 04 00 00 00 00 00 00 00 00 00 00 00 00
0x40000010:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000020:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

1, 2, 3, 4 순으로 메모리에 저장하였다.

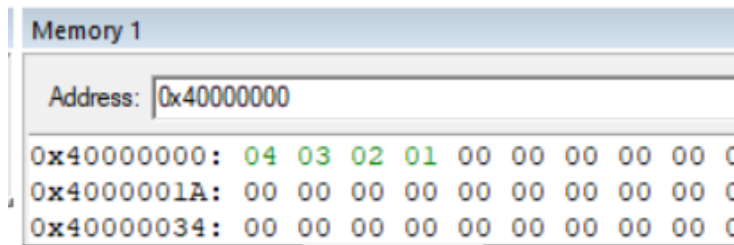
- v. 위에 저장된 값을 불러오기



Register	Value
Current	
R0	0x00000001
R1	0x00000002
R2	0x00000003
R3	0x00000004
R4	0x00000000
R5	0x04030201
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x40000000
R13 (...)	0x00000000
R14 (L...	0x00000000
R15 (...)	0x00000028
CPSR	0x00000003
SPSR	0x00000000
User/Sys...	
Fast Interr	

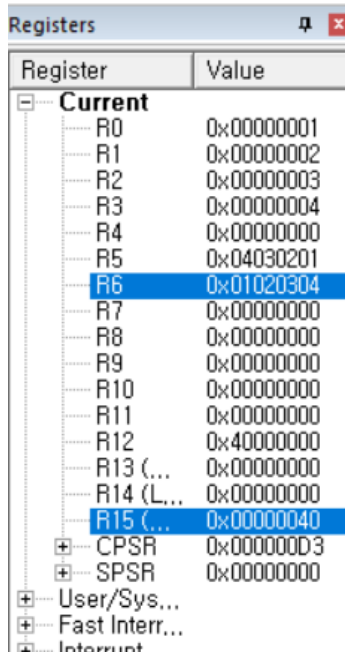
메모리에 01, 02, 03, 04 순으로 저장되어 있지만 little-endian임을 고려하면 역순인 04, 03, 02, 01로 저장됨을 볼 수 있다. 즉 r5에 0x04030201이 저장되었다.

- vi. 메모리에 원하는 순서로 바이트 단위로 저장



Memory 1	
Address: 0x40000000	
0x40000000:	04 03 02 01 00 00 00 00 00 00 00 00 00 00 00 00
0x4000001A:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000034:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- vii. 위에 저장된 값 불러오기



Register	Value
Current	
R0	0x00000001
R1	0x00000002
R2	0x00000003
R3	0x00000004
R4	0x00000000
R5	0x04030201
R6	0x01020304
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x40000000
R13 (...)	0x00000000
R14 (L...)	0x00000000
R15 (...)	0x00000040
+ CPSR	0x00000003
+ SPSR	0x00000000
+ User/Sys...	
+ Fast Interr...	
+ Interrupt	

메모리에는 04, 03, 02, 01 순으로 저장되어 있지만 little-endian임을 고려하면 역순인 01, 02, 03, 04로 저장됨을 볼 수 있다 즉 r6에 0x01020304가 저장되었다.

4. Consideration

이번에 N, Z, C, V flag들을 확인하고 conditional execution 또한 확인했다. 이를 이용해 비교한 값이 같을 때, 클 때, 작을 때 3가지를 비교하고 비교한 결과에 따라 다른 값이 저장되도록 하였다. Flag들을 통해 조건부 연산을 실행할 때 각 플래그의 조합에 따라 Signed와 unsigned가 달라짐을 알 수 있었다. 또한 디버그에서 메모리를 할당 후 ini 파일 없이 진행하면 읽고 쓰는 권한을 부여할 수 없지만 ini 파일을 통해 미리 권한을 부여하여 메모리에 저장하거나 불러올 수 있게 함을 확인하였다. 또한 불러올 때는 little

endian 방식이기 때문에 역순으로 저장됨을 확인하였다.

5. Reference

이형근 교수님/data_transfer_to_or_from_Mem.pdf/광운대학교 컴퓨터정보공학부/2022