

어셈블리프로그래밍 설계 및 실습

Lab #2 Data Transfer to or from MEM



Kwangwoon University
Embedded System Architecture Lab



학습 목표

- 기본 명령어 사용 예제를 통해 어셈블리어 프로그래밍을 이해한다.
- ARM 조건부 실행 코드 보는 방법을 이해하고 이를 어셈블리어 프로그래밍 능력을 습득한다.
- 원하는 데이터를 메모리로 저장 및 가져올 수 있도록 어셈블리어 프로그래밍 능력을 습득한다.

Contents

- ARM instruction set
 - Introduction
 - 기본 명령어 연습
 - 조건부 실행(conditional execution) 연습

- Single data transfer to/from memory
 - 메모리로 데이터 저장(store) 및 가져오기(load) 연습

Basic instruction set (1/4)

■ Register movement

Syntax: <instruction>{<cond>}{S} Rd, N

MOV	Move a 32-bit value into a register	$Rd = N$
MVN	move the NOT of the 32-bit value into a register	$Rd = \sim N$

- MOV R0, R2 ; R0 = R2
- MVN R0, R2 ; R0 = $\sim R2$

Basic instruction set (2/4)

■ Arithmetic

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N + \text{carry}$
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn - !(\text{carry flag})$
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N - !(\text{carry flag})$
SUB	subtract two 32-bit values	$Rd = Rn - N$

- ADC R0, R1, R2 ; R0 = R1 + R2 + C
 - ADD R0, R1, R2 ; R0 = R1 + R2
 - RSB R0, R1, R2 ; R0 = R2 - R1
 - RSC R0, R1, R2 ; R0 = R1 - R1 + C - 1
 - SBC R0, R1, R2 ; R0 = R1 - R2 + C - 1
 - SUB R0, R1, R2 ; R0 = R1 - R2
- ← CPSR의 C 비트 값

Basic instruction set (3/4)

■ Logical

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

AND	logical bitwise AND of two 32-bit values	$Rd = Rn \& N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn \mid N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \wedge N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$

- AND R0, R1, R2 ; R0 = R1 & R2
- ORR R0, R1, R2 ; R0 = R1 | R2
- EOR R0, R1, R2 ; R0 = R1 ^ R2
- BIC R0, R1, R2 ; R0 = R1 & (~R2)

Basic instruction set (4/4)

■ Comparison

- Only N, Z, C and V flags in CPSR are changed.
 - ▶ These instructions do not generate a result

Syntax: <instruction>{<cond>} Rn, N

CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \& N$

- CMN R1, R2 ; $R1 + R2 \rightarrow$ update flag
- CMP R1, R2 ; $R1 - R2 \rightarrow$ update flag
- TEQ R1, R2 ; $R1 \text{ XOR } R2 \rightarrow$ update flag
- TST R1, R2 ; $R1 \text{ AND } R2 \rightarrow$ update flag

Simple Example

■ Simple example

- Code size: 36 bytes, states: 5

```
AREA  ARMex, CODE, READONLY
ENTRY
```

start

```
MOV    r1, #10      ; Set up parameters
```

```
MOV    r2, #3
```

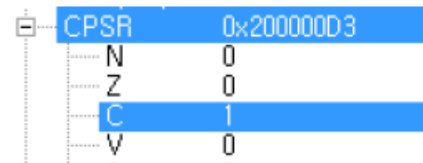
```
ADD    r0, r1, r2    ; r0 = r01+ r2
```

```
CMP    r0, r1
```

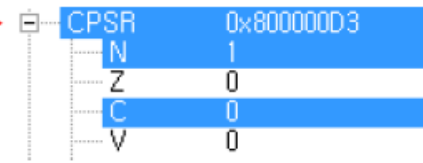
```
CMP    r1, r0
```

```
END
```

; Mark end of file



CPSR	0x200000D3
N	0
Z	0
C	1
V	0



CPSR	0x800000D3
N	1
Z	0
C	0
V	0

Conditional Execution (1/2)

■ Conditional field {Cond}

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not Equal
CS/HS	C set	Higher or same (unsigned \geq)
CC/LO	C clear	Lower (unsigned $<$)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned $>$)
LS	C clear or Z set	Lower or same (unsigned \leq)
GE	N and V the same	Signed \geq
LT	N and V different	Signed $<$
GT	Z clear, and N and V the same	Signed $>$
LE	Z set, or N and V different	Signed \leq
AL	Any	Always (This suffix is normally omitted.)
NV	Reserved	



Conditional Execution (2/2)

■ Example of usage

- {Cond}'s ex)
 - ▶ `ADD r0,r1,r2` ; `r0 = r1 + r2` (ADDAL)
 - ▶ `ADDEQ r0,r1,r2` ; If zero flag set then...
; ... `r0 = r1 + r2`
- {S}'s ex)
 - ▶ `ADDS r0,r1,r2` ; `r0 = r1 + r2`
; ... and set flags

Simple Example

■ Simple example

- Code size: 44 bytes, states: 8

```
AREA  ARMex, CODE, READONLY
ENTRY
```

start

```
MOV    r1, #10      ; Set up parameters
MOV    r2, #3
ADD    r0, r1, r2    ; r0 = r01+ r2
CMP    r0, r1
ADDCS  r0, r0, r2
CMP    r1, r0
ADDCS  r0, r1, r1
END                ; Mark end of file
```

Current

R0	0x00000010
R1	0x0000000A
R2	0x00000003

R0	0x00000010
R1	0x0000000A

Exercise (1/3)

■ 다음 동작을 하는 C 코드를 어셈블리어로 구현하라.

- 단, 반드시 조건부 실행 명령어를 사용하여라.

1.

```
if (r0 < r1)
    r2 = r0;
else if (r0 > r1)
    r2 = r1;
else
    r2 = r0 + r1;
```

2.

```
if (r0 > 0)
    r3 = r0;
if (r1 < 0)
    r3 += r1;
if (r2 <= 7)
    r3 += r2;
if (r3 == 0)
    r3 = r0;
```

Exercise (2/3)

■ Solution of exercise 1

- Code size: 28 bytes, states: 7

```
AREA    ARMex, CODE, READONLY
ENTRY
```

start

```
MOV     r0, #10      ; Set up parameters
MOV     r1, #10
CMP     r0, r1       ; compare( r0 - r1)
MOVMI   r2, r0       ; MOV r0 into r2, if r0 < r1.
MOVGT   r2, r1       ; MOV r1 into r2, if r0 > r1.
ADDEQ   r0, r0, r1    ; ADD r0 and r1 into r0, if r0 == r1
MOVEQ   r2, r0       ; MOV r1 into r2, if r0 == r1.

END                ; Mark end of file
```

Exercise (3/3)

■ Solution of exercise 2

- Code size: 52 bytes, states: 13

```
AREA    ARMex, CODE, READONLY
ENTRY
```

start

```
MOV     r0, #1      ; Set up parameters
MOV     r1, #-1
MOV     r2, #8

MOV     r4, #0
CMP     r0, r4      ; compare( r0 - 0)
MOVPL   r3, r0      ; MOV r0 into r3, if r0 > 0.

CMP     r1, r4      ; compare( r1 - 0)
ADDMI   r3, r3, r1   ; ADD r1 into r3, if r1 > 0.

MOV     r4, #7
CMP     r2, r4      ; compare( r2 - 7)
ADDLE   r3, r3, r2   ; ADD r2 into r3, if r2 <= 7.

TST     r3, #0      ; compare (r3 and 0)
MOVEQ   r3, r0      ; MOV r0 into r3, if r3 == 0

END                      ; Mark end of file
```



Load and Store (1/2)

- Load from MEM or store to MEM

Syntax: <LDR|STR>{<cond>}{B} Rd, addressing¹
LDR{<cond>}SB|H|SH Rd, addressing²
STR{<cond>}H Rd, addressing²

LDR	load word into a register	$Rd \leftarrow mem32[address]$
STR	save byte or word from a register	$Rd \rightarrow mem32[address]$
LDRB	load byte into a register	$Rd \leftarrow mem8[address]$
STRB	save byte from a register	$Rd \rightarrow mem8[address]$

Load and Store (2/2)

- Load from MEM or store to MEM (half-word)

LDRH	load halfword into a register	$Rd \leftarrow mem16[address]$
STRH	save halfword into a register	$Rd \rightarrow mem16[address]$
LDRSB	load signed byte into a register	$Rd \leftarrow SignExtend(mem8[address])$
LDRSH	load signed halfword into a register	$Rd \leftarrow SignExtend(mem16[address])$

Simple Example

■ Example 1

- Code size: 20 bytes, states: 16
- LDR 또는 STR명령어를 사용할 경우 ini파일을 통해 사용하고자 하는 메모리 영역에 read 또는 write권한을 부여해야 함!

```
AREA  ARMex, CODE, READONLY
ENTRY
```

start

```
MOV r0, #8
```

```
LDR r1, TEMPADDR1
```

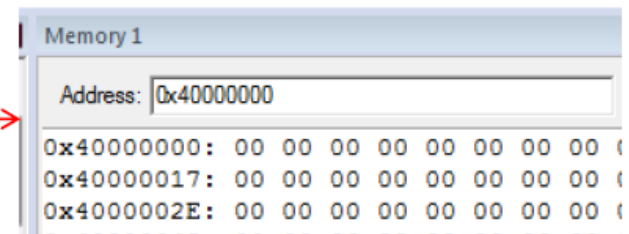
```
STRB r0, [r1]
```

```
TEMPADDR1  & &40000000
```

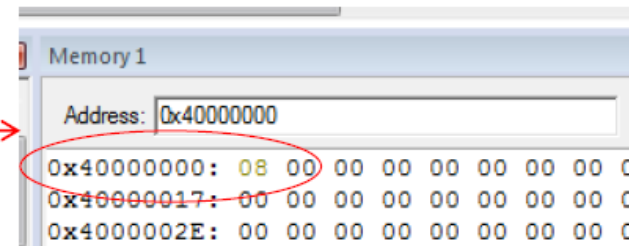
```
MOV pc, lr
```

```
END
```

; Mark end of file



Memory 1	
Address: 0x40000000	
0x40000000:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000017:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000002E:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

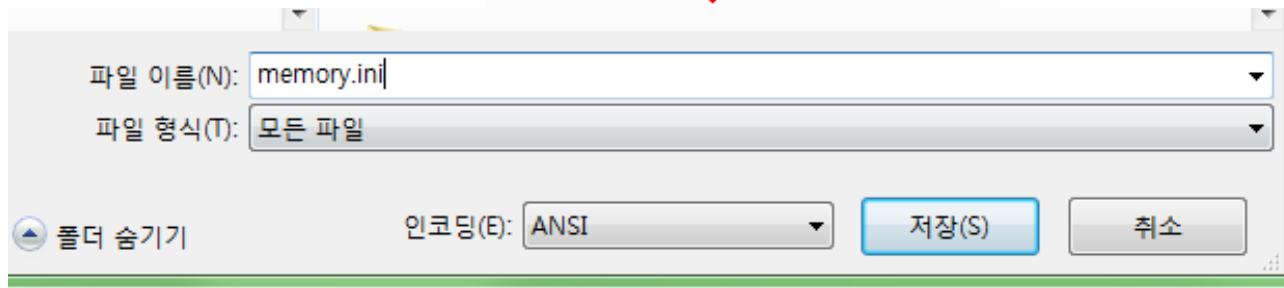
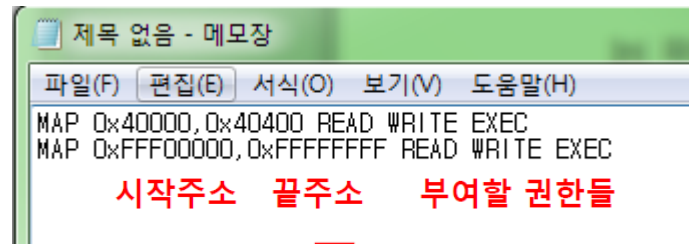


Memory 1	
Address: 0x40000000	
0x40000000:	08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000017:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000002E:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Register INI File

■ Make ini file

- Debug모드 진입할 때 읽게 되는 파일로, 임의의 메모리 영역에 대해 read, write, exec 권한 부여
- 접근하는 영역에 read권한이 없을 경우 메모리로부터 load불가, write 권한이 없을 경우 메모리로 store불가



Addressing mode (1/2)

■ Immediate

- ex) LDR R0, [R1, #4] ; mem [R1+4]

■ Register

- ex) LDR R0, [R1, R2] ; mem [R1+R2]

■ Scaled register

- ex) LDR R0, [R1, R2, LSL #2]

Addressing mode (2/2)

■ Pre-index addressing

- ex) LDR R0, [R1, #4] ; R0 = mem [R1+4], R1 unchanged

■ Auto-index addressing

- ex) LDR R0, [R1, #4]! ; R0 = mem [R1+4], R1 = R1+4

■ Post-index addressing

- ex) LDR R0, [R1], #4 ; R0 = mem [R1], R1 = R1+4

Index method	Data	Base address register	Example
Preindex with writeback	$mem[base + offset]$	$base + offset$	LDR r0, [r1, #4]!
Preindex	$mem[base + offset]$	not updated	LDR r0, [r1, #4]
Postindex	$mem[base]$	$base + offset$	LDR r0, [r1], #4

Simple Example

■ Auto-index

- Code size: 40 bytes, states: 20

```
AREA    ARMex, CODE, READONLY
ENTRY
```

start

```
LDR    r0, TEMPADDR1
LDR    r1, TEMPADDR2
MOV    r2, #156
MOV    r3, #201
```

```
STR    r2, [r0]
STR    r3, [r1]
```

```
LDR    r1, [r0, #4]!
```

```
TEMPADDR1    & &00001000
TEMPADDR2    & &00001004
```

```
MOV    pc, lr
END
```

; Mark end of file

Memory 1	
Address:	0x00001000
0x00001000: 9C 00 00 00 C9 00 00 00 00	

Current	
R0	0x00001000
R1	0x00001004
R2	0x0000009C

Memory 1	
Address:	0x00001000
0x00001000: 9C 00 00 00 C9 00 00 00 00	

Current	
R0	0x00001004
R1	0x000000C9
R2	0x0000009C

Exercise (1/2)

- 다음 레지스터에 각각 다음 값이 들어있다고 했을 때, 각 레지스터의 값을 메모리에 차례대로 저장하라.
 - 단 0x00001000번지부터 시작하여 한 숫자당 2바이트 단위로 저장한다.
 - R0: 0x10, R1: 0x0A, R2: 0x08, R3: 0x22

Exercise (2/2)

■ Solution of exercise 3

- Code size: 44 bytes, states: 16

```
AREA    ARMex, CODE, READONLY
ENTRY
```

start

```
MOV r0, #16
MOV r1, #10
MOV r2, #08
MOV r3, #34
```

```
LDR r4, TEMPADDR1
STRB r0, [r4], #2
STRB r1, [r4], #2
STRB r2, [r4], #2
STRB r3, [r4]
```

```
TEMPADDR1  & &00001000
```

```
MOV pc, lr
END
```

; Mark end of file

Memory 1															
Address: 0x00001000															
0x00001000: 10 00 0A 00 08 00 22 00 00 00															

Problem

1. 메모리에 저장된 숫자 3개를 1 바이트 단위로 읽어서 해당 가져온 데이터의 값에 따라 R5의 값을 바꾸는 프로그램을 작성하라.
 - 0x0A보다 클 경우 1을, 0x0A보다 작을 경우 2를, 0x0A와 같을 경우 3을 저장한다.
 - ▶ 단, 반드시 조건부 실행 명령어를 사용 해야 한다.
 - ▶ 읽어올 메모리의 번지는 구현자가 임의로 설정
 - ▶ 또한 최초에 메모리에 저장되는 값은 구현자가 임의로 입력 후 시작한다.
 - ex) 메모리 번지 0x00001000에 0x11를 저장하고 시작
2. 각 레지스터에 다음과 같은 값들이 저장돼 있다고 했을 때, 현재 little-endian 방식임을 감안하여 R5와 R6에 다음을 저장하라.
 - R0=1, R1=2, R3=3, R4=4
 - R5 = 0x04030201
 - R6 = 0x01020304
 - ▶ (힌트) 메모리를 사용하시오



Homework

■ 제출기한

● Soft copy

- ▶ 2019. 9. 20(금) 밤 16시 29분 59초까지 u-campus에 제출
- ▶ 압축 파일은 각 과제 파일을 모아서 제출(ini파일 반드시 포함)
- ▶ 압축파일 형식
 - Assignment_(번호)_(학번).zip
 - ▶ ex) Assignment_1_2012722069.zip
- ▶ 하드카피 제출 X