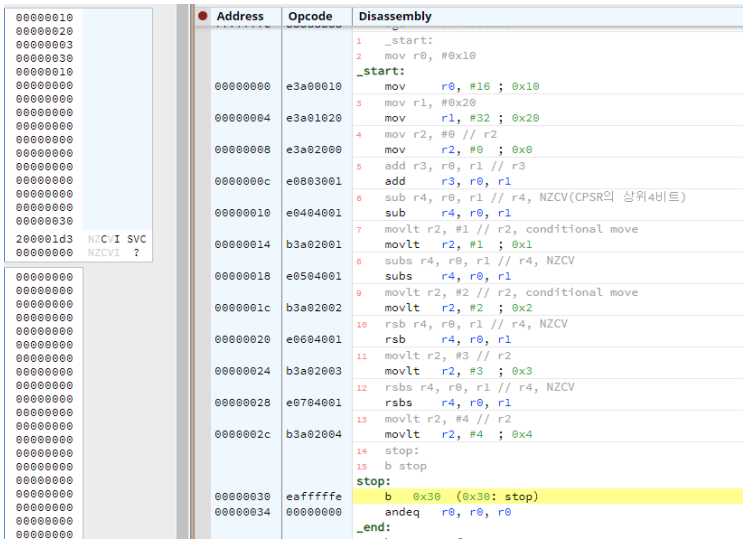
HW3

2017253019 안희영

2.



Address	Opcode	Disassembly
		1 _start:
		2 mov r0, #0x10
		_start:
00000000	e3a00010	mov r0, #16 ; 0x10
		3 mov r1, #0x20
00000004	e3a01020	mov r1, #32 ; 0x20
		4 mov r2, #0 // r2
00000008	e3a02000	mov r2, #0 ; 0x0
		5 add r3, r0, r1 // r3
0000000c	e0803001	add r3, r0, r1
		6 sub r4, r0, r1 // r4, NZCV(CPSR의 상위4비트)
00000010	e0404001	sub r4, r0, r1
		7 movlt r2, #1 // r2, conditional move
00000014	b3a02001	movlt r2, #1 ; 0x1
		8 subs r4, r0, r1 // r4, NZCV
00000018	e0504001	subs r4, r0, r1
		9 movlt r2, #2 // r2, conditional move
0000001c	b3a02002	movlt r2, #2 ; 0x2
		10 rsb r4, r0, r1 // r4, NZCV
00000020	e0604001	rsb r4, r0, r1
		11 movlt r2, #3 // r2
00000024	b3a02003	movlt r2, #3 ; 0x3
		12 rsbs r4, r0, r1 // r4, NZCV
00000028	e0704001	rsbs r4, r0, r1
		13 movlt r2, #4 // r2
0000002c	b3a02004	movlt r2, #4 ; 0x4
		14 stop:
		15 b stop
		stop:
00000030	eafffffe	b 0x30 (0x30: stop)
00000034	00000000	andeq r0, r0, r0
		_end:

mov r0, #0x10 // r0에 0x10 이동 r0=10

mov r1, #0x20// r1에 0x20 이동 r1=0x20

mov r2, #0 // r2에 0 이동 r2=0

add r3, r0, r1 // r3에 r0와 r1의 값을 더해서 넣는다. R3 = 0x30

sub r4, r0, r1 // r0와 r1의 뺄샘연산을 r4에 넣는다. -0x10

movlt r2, #1 // N플래그가 1이면 r2에 1 이동(N은 0인상태)

subs r4, r0, r1 // r0 에 r1을 뺀 결과를 플래그에 반영. c플래그 0(결과가 음수기 때문) N=1 c=0

movlt r2, #2 // N플래그가 1이면 r2에 1 이동(N은 1인상태) r2=2

rsb r4, r0, r1 // r1에서 r0를 뺀 값을 r4로 이동 r4=0x10

movlt r2, #3 // // N플래그가 1이면 r2에 3 이동(N은 1인상태) r2=3

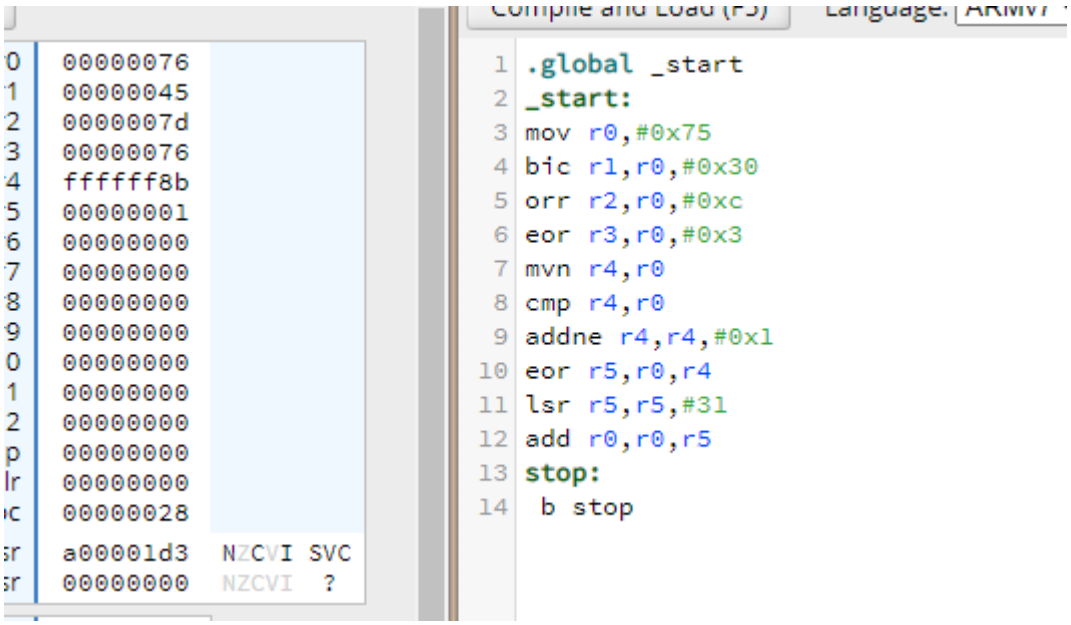
rsbs r4, r0, r1 // r1에서 r0를 뺀 값을 r4로 이동, 플래그에 영향을 줌. c플래그 1(결과가 양수기 때문) n=0, c=1, r4=10

movlt r2, #4 // r2 N플래그가 1이면 r2에 4 이동(N은 0인상태)

stop:

b stop //stop으로 계속 이동.(멈춤)

3.



Mov r0,#0x75 75를 r0로

Bic r1,r0,#0x30 0x30에 해당하는 비트를 0으로 변환, r1으로 이동

Orr r2,r0,#0xc r0를 0xC와 or 연산한 결과를 r2로 이동

Eor r3,r0,#0x3 0x3과 r0의 값을 xor연산하여 r3로 이동

Mvn r4,r0 r0의 비트들을 반전한 결과를 r4로 이동

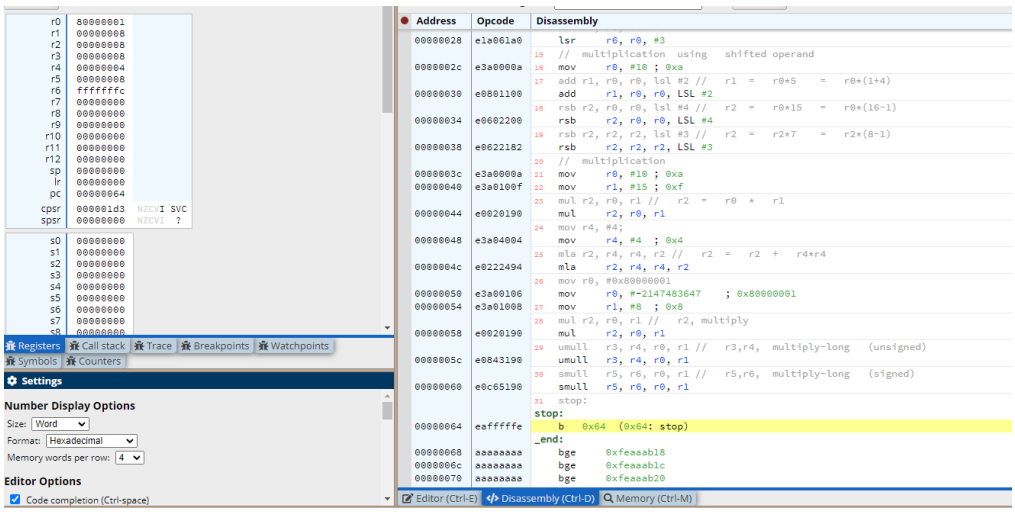
Cmp r4, r0 r4와 r0가 같은 숫자인지 채크 (N플래그 1로 세트)

Addne r4,r4,#0x1 N플래그가 1이면 r4의 값과 1을 더해서 r4로 이동

Eor r5,r0,r4 r4와 r0를 xor연산한 결과를 r5로 이동

Lsr r5,r5,#31 31비트를 논리쉬프트하여 r4와 r0의 부호가 같으면 0, 틀리면 1

Add r0,r0,r5 r5와 r0의 값을 더해서 r0로 이동한다.

4. 

Mov r0, #0x10 // r0 에 10 이동

mov r1, #4 // r1에 4 이동

mov r2, r0,lsl #3 // r2에 r0값을 3비트 쉬프트하여(곱하기 8) r0에 이동

lsl r2, r0, #3 // r0를 3비트 쉬프트한 값을 r2에 저장.(값이 같음)

mov r3, r0,lsr r1 // r3에 r0를 r1값만큼 쉬프트하여(나누기 16) r3에 저장

lsr r3, r0, r1 // r0를 r1값만큼 쉬프트한 값을 r3에 저장.(값이 같음)

mov r0, #-8 // r0에 -8 저장

mov r4, r0, asl #2 // r4에 r0를 부호를 유지하며 2비트 쉬프트

mov r5, r0, asr #3 // r5에 r0를 부호를 유지하며 3비트 쉬프트(나누기 8)

asr r5,r0,#3 // r5에 r0를 부호를 유지하며 3비트 쉬프트(동일)

mov r6, r0, lsr #3// r6에 r0를 3비트 쉬프트한(나누기) 값을 저장1

Mov r0, #10 //r0에 10을 저장

add r1, r0, r0, lsl #2 //r1에 r0와 r0를 2비트 쉬프트(곱하기 4)한 값을 더하여 저장

rsb r2, r0, r0, lsl #4 // r2에 r0를 4비트 쉬프트(16배)한 값에 r0를 뺀 값을 저장

rsb r2, r2, r2, lsl #3 // r2에 r0를 4비트 쉬프트(8배)한 값에 r0를 뺀 값을 저장

Mov r0, #10 // r0에 10을 저장

mov r1, #15 //r1에 15를 저장

mul r2, r0, r1 // r2에 r0과 r1의 곱을 저장

mov r4, #4 // r4에 4 저장

mla r2, r4, r4, r2 // r2에 r4와 r4의 곱 에 r2의 값을 더한 값을 저장

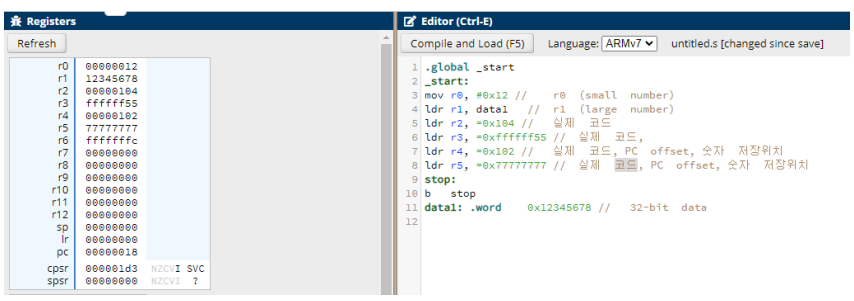
mov r0, #0x80000001 // 0x80000001를 r0에 저장

mov r1, #8 // r1에 8 저장

mul r2, r0, r1 // r2에 r0과 r1의 곱 저장 (32비트 이상의 결과가 잘려서 나타남)

umull r3, r4, r0, r1 // r1와 r0의 부호없는 곱셈 결과를 r3,r4에 저장(64비트)

smull r5, r6, r0, r1 // r1과 r0의 부호있는 곱을 r5, r6에 (64비트)



```
mov     r0,     #0x12 //   r0          r0에 12 저장
ldr     r1,     data1    // LDR r1 ,[PC,#offset]      r1에 data1의 pc 주소에서 데이터를 읽어옴
ldr     r2,     =0x104 //   mov r2, #0x104
ldr     r3,     =0xffffff55 //r3에 0xffffff55를 읽어옴
ldr     r4,     =0x102 //  mov r4, #0x102와 같다.
ldr     r5,     =0x77777777 //      r5에 0x77777777 를 읽어옴
stop:
b       stop   //stop 으로 이동
data1: .word        0x12345678 //        32-bit      data
```