

HW4

2017253019 안희영

The screenshot shows a debugger interface with two main panels. The left panel, titled 'Registers', lists various CPU registers (r0-r15, cpsr, spsr) and their current values. The right panel, titled 'Disassembly (Ctrl-D)', shows a list of assembly instructions with their addresses, opcodes, and disassembled code. The instructions include moving data1 to r0, loading words from memory into r1-r6, and performing logical operations. A 'stop' instruction is highlighted in yellow.

.global _start

_start:

Mov r0, #data1 // r0에 data1의 위치를 넣습니다.

Ldr r1,[r0] // r1에 r0의 offset에 있는 word를 불러옵니다.(1 로드됨)

Ldr r2,[r0,#4] // r2에 r0의 offset에서 4바이트 떨어진 워드를 불러옵니다(r0 값 변하지 않음, 2로드됨)

Ldr r3,[r0],#4 // r3에 r0의 offset에서 불러온 뒤 r0의 값을 4만큼 이동합니다(읽어온 후 r0 값을 변동합니다, 1로드됨)

Ldr r4,[r0,#4]! // r0의 offset을 4만큼 이동합니다 그리고 r0의 위치를 읽어옵니다.(r4에 3로드됨)

Ldr r0,=#data1// r0에 data1의 위치를 넣습니다.

Mov r1,# r1에 0xc를 넣습니다

Ldr r5,[r0, r1] // r5에 r0의 위치에서 r1만큼(3개워드만큼)뒤의 데이터를 불러옵니다(r5 = 4)

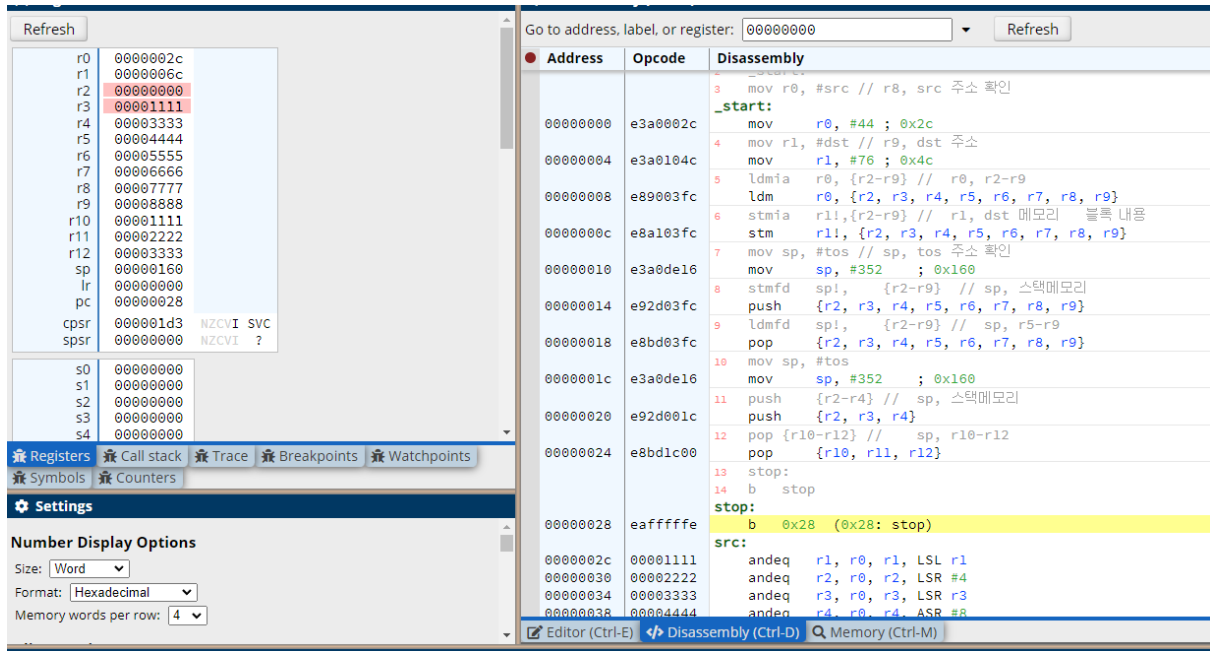
Mov r1,#3// r1에 0xc를 넣습니다

Ldr r6,[r0, r1,ls#2]// 1에 r0의 위치 + r1의 ls#2(곱하기 4) 만큼 떨어진곳의 데이터를 불러옵니다. (r6 = 4)

Stop:

B stop

data1: .word 1,2,3,4,5,6,7,8



.global _start

_start:

Mov r0, #src // r0에 src의 위치를 읽어옴

Mov r1, #dst // r1에 dst의 위치를 읽어옴

Ldmia r0, {r2-r9} // r0의 위치에서 시작하여 r2-r9 만큼의 블록을 읽어옴

Stmia r1!, {r2-r9} // r1의 위치에서 r2-r9 dst 위치에 저장. R1값 변경됨

Mov sp, #tos // sp에 tos의 주소값

Stmfd sp!, {r2-r9} // sp의 스택 메모리에 r2-r9값 저장

Ldmfd sp!, {r2-r9} // sp의 스택 메모리에 r2-r9값 읽어옴

Mov sp, #tos //에 tos의 위치 불러옴

Push {r2-r4} // //tos를 시작으로 r2-r4값을 스택에 넣음

Pop {r10-r12} // //스택에 저장된 값을 r10-r12에 꺼냄

stop:

b stop

src: .word 0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666, 0x7777, 0x8888

dst: .space 20

stack: .space 256

tos: //top of stack

Address	Opcode	Disassembly
fffffdd0	aaaaaaaa	bge 0xfeaaaa80
fffffdd4	aaaaaaaa	bge 0xfeaaaa84
fffffdd8	aaaaaaaa	bge 0xfeaaaa88
fffffddc	aaaaaaaa	bge 0xfeaaaa8c
fffffde0	aaaaaaaa	bge 0xfeaaaa90
fffffde4	aaaaaaaa	bge 0xfeaaaa94
fffffde8	aaaaaaaa	bge 0xfeaaaa98
fffffdec	aaaaaaaa	bge 0xfeaaaa9c
fffffde0	aaaaaaaa	bge 0xfeaaaaa0
fffffde4	aaaaaaaa	bge 0xfeaaaaa4
fffffde8	aaaaaaaa	bge 0xfeaaaaa8
fffffdec	aaaaaaaa	bge 0xfeaaaaac
1	.global _start	
2	_start: // (a) (b) (c) (d)	
3	ldr r0, =#0x80 // 0x80 0x70000000 0x10 0x10	
4	ldr r1, =#0x10 // 0x10 0x90000000 0x80 0x10	
5	cmp r0, r1 // N, V, Z ?	
6	movgt r2, #1 ; 0x1	
7	movlt r2, #2 ; 0x2	
8	moveq r2, #3 // r2 ?	
9	moveq r2, #3 ; 0x3	
10	stop:	
11	b 0x18 (0x18: stop)	
12	andeq r0, r0, r0	
13	_end:	
14	bge 0xfeaaaaad0	
15	bge 0xfeaaaaad4	

- (a) C 플래그 1, r0가 크다.
- (b) N,V플래그 1. N 플래그가 1이기 때문에 cmp의 뺄셈 결과가 음수 = r1이 더 크다.
- (c) N플래그 1. 큰 수가 아니었기 때문에 뺄셈 결과(2보수후 덧셈)에서 캐리가 발생하지 않음. V 플래그 0
- (d) z플래그 1. cmp의 뺄셈 결과가 0이 나왔기 때문에 세트됨. = > r1과 r0의 값이 같았다는 의미

Address	Opcode	Disassembly
fffffddc	aaaaaaaa	bge 0xfeaaaa8c
fffffde0	aaaaaaaa	bge 0xfeaaaa90
fffffde4	aaaaaaaa	bge 0xfeaaaa94
fffffde8	aaaaaaaa	bge 0xfeaaaa98
fffffdec	aaaaaaaa	bge 0xfeaaaa9c
fffffde0	aaaaaaaa	bge 0xfeaaaaa0
fffffde4	aaaaaaaa	bge 0xfeaaaaa4
fffffde8	aaaaaaaa	bge 0xfeaaaaa8
fffffdec	aaaaaaaa	bge 0xfeaaaaac
1	.global _start	
2	_start:	
3	mov r0, #54 ; 0x36	
4	mov r1, #24 ; 0x18	
5	gcd: cmp r0, r1	
6	subgt r0, r0, r1	
7	sublt r1, r1, r0	
8	bne gcd	
9	bne 0x8 (0x8: gcd)	
10	stop:	
11	b 0x18 (0x18: stop)	
12	andeq r0, r0, r0	
13	_end:	

r0 - r1일 때 둘의 값을 비교한다.

R0에 r1을 빼서 음수가 나오면 r1에 r0를 빼서 r1을 줄이고 양수가 나오면 r0에 r1을 빼서 r0를 줄인다.

둘이 같아질때까지 gcd의 위치로 루프를 진행한다.

.end

	Address	Opcode	Disassembly
r0 0000000f			3 _start:
r1 00000000			4 LDR R0, N
r2 00000000			
r3 00000000			5 _start:
r4 00000000	00000000	e59f0020	ldr r0, [pc, #32] (0x28: N)
r5 00000000			
r6 00000000			6 BL FINDSUM
r7 00000000	00000004	eb000000	bl 0xc (0xc: FINDSUM)
r8 00000000			7 END: B END
r9 00000000			
r10 00000000	00000008	ea000000	END:
r11 00000000			b 0x8 (0x8: END)
r12 00000000			7 FINDSUM:
sp 00000000			8 PUSH {R4, LR} // save state (R4, LR)
lr 00000020			
pc 00000008	0000000c	e92d4010	FINDSUM:
cpsr 400001d3 NZCVI SVC			9 push {r4, lr}
spsr 00000000 NZCVI ?	00000010	e1b04000	10 MOVN R4, R0 // save N in R4, and check for 0
			movn r4, r0
	00000014	da000002	11 BLE RETURN // if N == 0, just return N
s0 00000000			ble 0x24 (0x24: RETURN)
s1 00000000			
s2 00000000			12 RECURSE:
s3 00000000			SUB R0, R4, #1 // set R0 = N-1
s4 00000000			
s5 00000000	00000018	e2440001	RECURSE:
s6 00000000			sub r0, r4, #1 ; 0x1
s7 00000000			
s8 00000000	0000001c	ebfffffa	13 BL FINDSUM
s9 00000000			bl 0xc (0xc: FINDSUM)
s10 00000000	00000020	e0840000	14 ADD R0, R4, R0 // R0 = N + FINDSUM(N-1)
s11 00000000			add r0, r4, r0
s12 00000000			15 RETURN:
			16 POP {R4, PC} // the return value is in R0
	00000024	e8bd0010	RETURN:
			pop {r4, pc}

.text

.global _start

_start:

LDR R0,N //R0에 N에 저장된 값을 불러옴

BL FINDSUM //FINDSUM을 호출, lr에 pc값이 저장됨

END: B END //END로 이동

FINDSUM: //FINDSUM 시작지점

PUSH {R4, LR} // R4에 저장된 값과 lr값을 스택에 저장,

//여러 번의 서브루틴 호출에 대응 가능). lr에 저장된 시점으로 돌아온다.

MOVS R4, R0 //r4에 r0의 값을 이동한다. R0의 값에 따라 플래그 세트

BLE RETURN //플래그에 따라 return으로 이동. 이 경우 r0가 0일 때 z플래그 세트후

RECURSE: // recurse 시작점

SUB R0, R4,#1 //r4값을 1 줄이고 r0로 저장한다.

BL FINDSUM //FINDSUM으로 이동. lr값에 pc를 저장. 이후 이 시점의 pc값을 스택에서 읽어와 뒤의 add를 실행한다.

ADD R0, R4,R0 //r0에 r4(이전 서브루틴에서 저장된 r0에 1씩을 뺀값)과 r0(현재 r4들의 합)을 더해 저장

RETURN: //RETURN 시작점

POP {R4, PC} //r4와 pc값을 다시 돌려놓는다. pc값에 따라서 돌아갈 위치에서 다시 재진행된다.

N: .word 5

.end