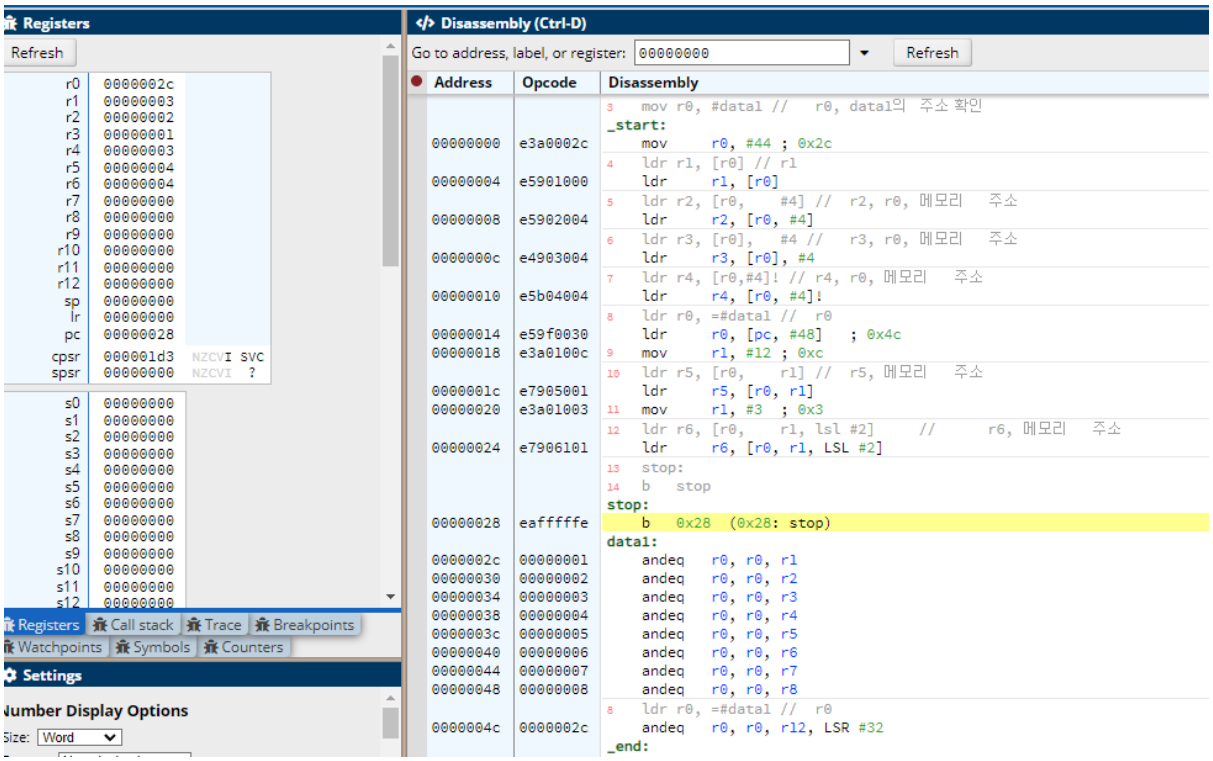
HW4

2017253019 안희영



.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)

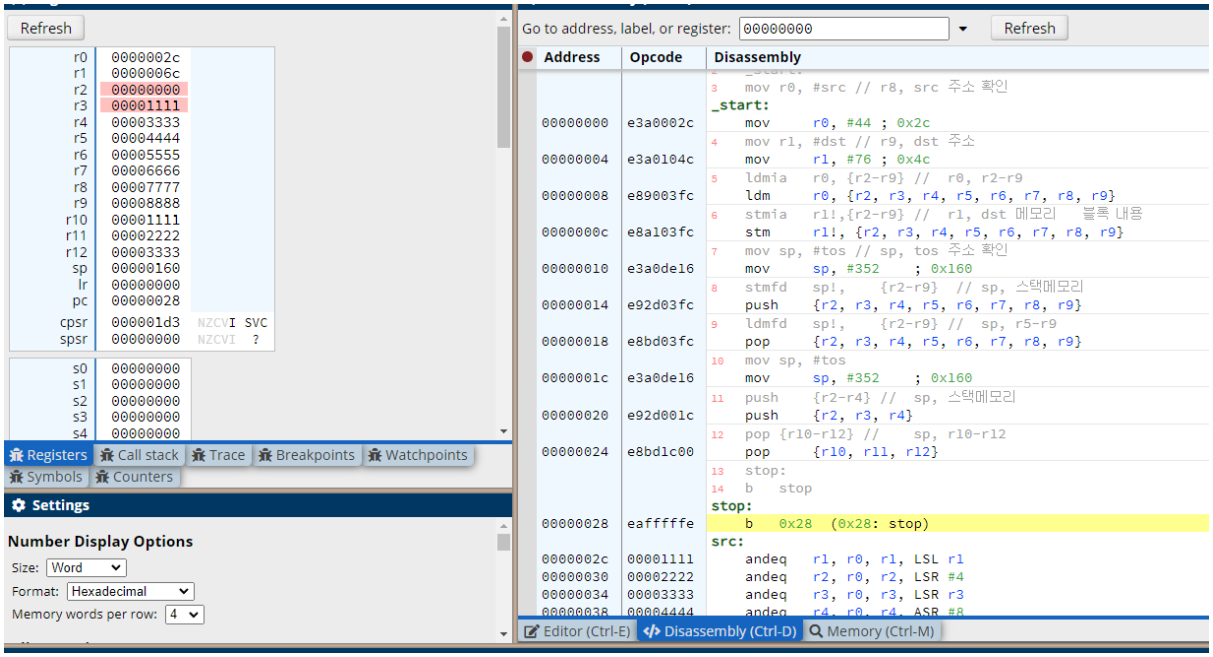
Mov r1,#3// r1에 0xc를 넣습니다

Ldr r6,[r0, r1,lsl#2]// 1에 r0의 위치 + r1의 lsl#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:

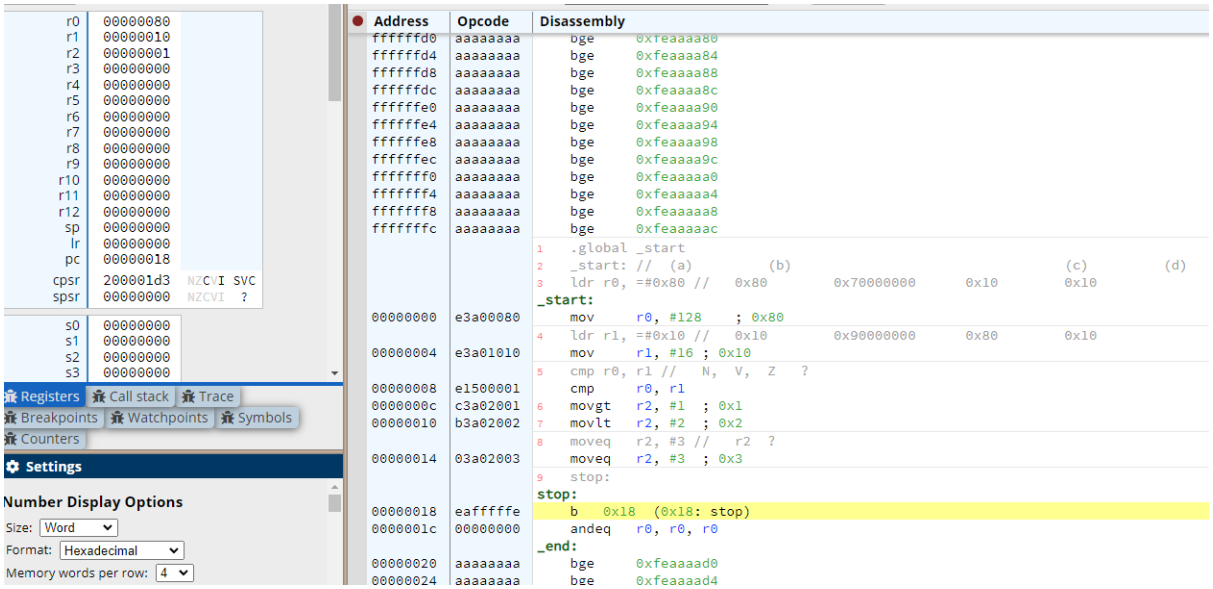
b stop

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

dst: .space 20

stack: .space 256

tos: //top of stack

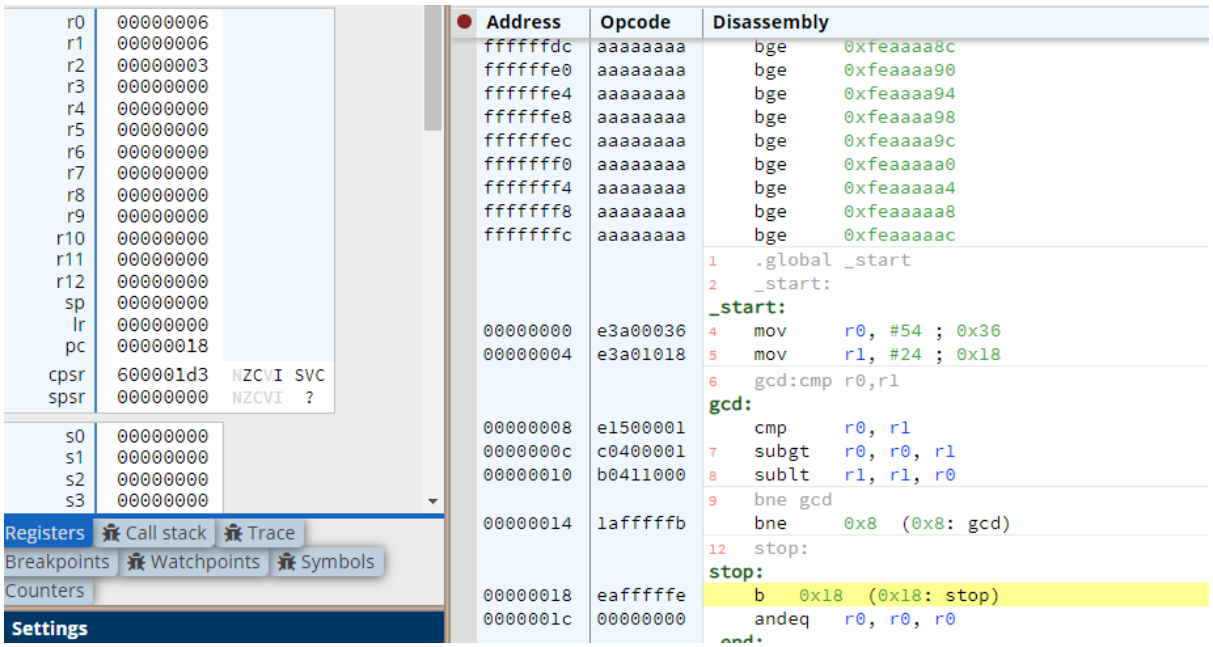


(a) C 플래그 1 , r0가 크다.

(b) N,V플래그 1. N 플래그가 1이기 때문에 cmp의 뺄샘 결과가 음수 = r1이 더 크다.

(c) N플래그 1. 큰 수가 아니었기 때문에 뺄샘 결과(2보수후 덧샘)에서 캐리가 발생하지 않음. V 플래그 0

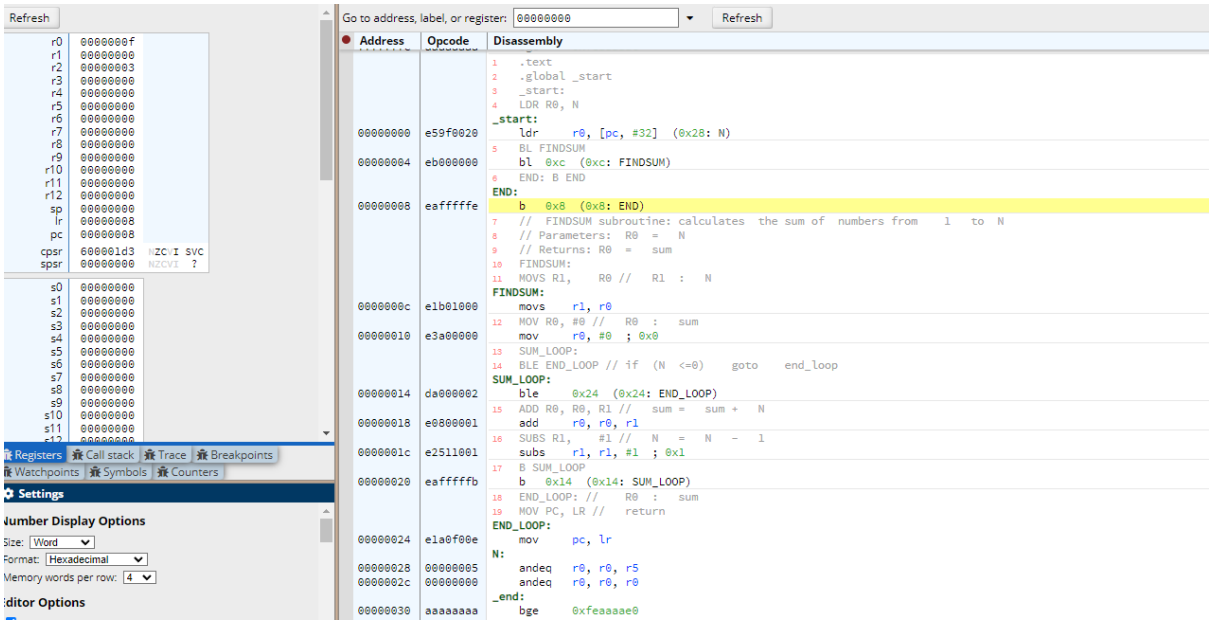
(d) z플래그 1. Cmp의 뺄샘 결과가 0이 나왔기 때문에 세트됨. = > r1과 r0의 값이 같았다는 의미



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.



_start:

LDR R0,N //r0에 N값(5)를 읽어옴.

BL FINDSUM //FINDSUM을 호출함.(FINDSUM이 끝났을 때 돌아올 pc를 lr에 저장해둠)

END: B END //END로 이동(끝)

FINDSUM: //FINDSUM 시작점

MOVS R1, R0 // r1에 r0(N = 5)를 저장

MOV R0, #0 //r0를 비운다.

SUM_LOOP: //루프 시작지점.

BLE END_LOOP //N이 0이면 ENDLOOP로 이동(이 코드에선 z플레그가 1이면 실행됨)

ADD R0, R0,R1 //r0에 r0값과 r1값을 더함

SUBS R1, #1 //r1을 1줄임, 이때 플래그에 영향 끼침 0일 때 (Z 플래그)

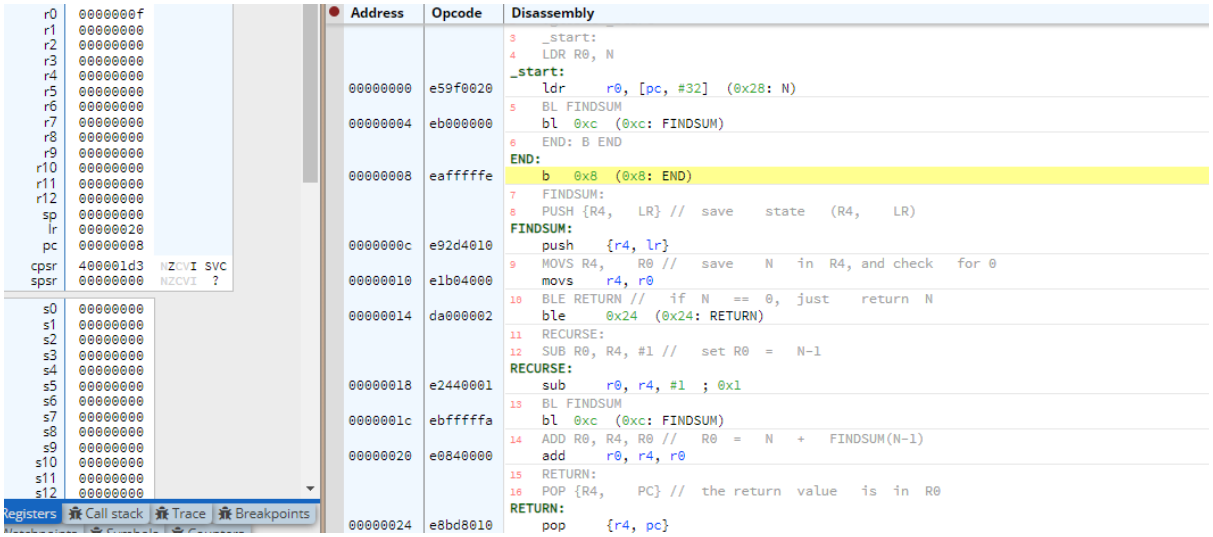
B SUM_LOOP // SUM_LOOP로 이동

END_LOOP: //리턴 시작점

MOV PC, LR //lr에 저장된 서브루틴 종료후 돌아갈 pc로 pc를 세트

N: .word 5

.end



.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. 이경우 r0가 0일 때 z플레그 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