

[숙제 6] lab2 실습 내용

제출할 자료:

(1) 실습 A 관련(LAB2 P10) Lab2 p4를 참고하여 example1() 함수 코드를 수정한 뒤 재빌드 하세요. 빨간색 박스 코드에 해당되는 어셈블리 코드, step by step으로 수행할 때 메모리 변화를 캡처하고 설명과 함께 보고서에 첨부하세요.

1. example1()을 assembly 코드로 변환하면 아래와 같이 표현된다.

(Assembly Code)

0x10558subtr3, r11, #16example1+28

0x1055cstrtr3, [r11, #-12]example1+32

0x10560ldtr3, [r11, #-12]example1+36

0x10564movtr2, #16example1+40

0x10568strtr2, [r3]example1+44

0x1056cmovtr3, #256t; 0x100example1+48

0x10570strtr3, [r11, #-16]example1+52

0x10574noprtr; (mov r0, r0)example1+56

0x10578ldtr3, [pc, #28]t; 0x1059c <example1+96>example1+60

0x1057cldtr2, [r3]example1+64

0x10580ldtr3, [r11, #-8]example1+68

0x10584eorstr2, r3, r2example1+72

0x10588movtr3, #0example1+76

0x1058cbeqt0x10594 <example1+88>example1+80

0x10590blt0x3073c <__stack_chk_fail_local>example1+84

0x10594subtsp, r11, #4example1+88

0x10598popt{r11, pc}example1+92

0x1059candeqtr1, r6, r12, lsl pcexample1+96

The screenshot displays a debugger interface with two main panels. The left panel shows the assembly code for the function example1(), with line numbers 24 through 48. The right panel shows the current state of the CPU registers, including r0 through r12, sp, lr, pc, cpsr, and d0.

Assembly Code (Left Panel):

```
24  ptr = &a;
25
26  *ptr = 9;
27
28  a = 3;
29
30  //
31  int b;
32  int *ptr1;
33
34  ptr1 = &b;
35  *ptr1 = 16;
36
37  b = 256;
38
39  }
40
```

Register Values (Right Panel):

Register	Value	Comment
r0	0x1	1
r1	0xfffffa4	4294898084
r2	0xfffffa4	4294898092
r3	0xfffff04c	4294897740
r4	0x11018	69656
r5	0x10158	65880
r6	0xb408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b84	563972
r11	0xfffff05c	4294897756
r12	0xfffff078	4294897784
sp	0xfffff048	4294897736
lr	0x10530	66864
pc	0x10560	66912
cpsr		
d0		VFP double-

2. *ptr1 = 16을 수행 후 r2에는 16(0x10) 이 저장되고 r3가 가리키는 주소값 0x0ffffef04c에 a의 값인 0x00000010이, 다음주소 0xffffef050에는 a의 주소값인 0xffffef04c이 저장된다.

The screenshot shows a debugger interface with assembly code on the left and memory/register windows on the right.

Assembly Code:

```

24 ptr = 5a;
25
26
27 *ptr = 9;
28
29 a = 3;
30
31 int b;
32 int *ptr1;
33
34 ptr1 = &b;
35
36 *ptr1 = 16;
37
38 b = 256;
39
40
41 void example2()
42 {
43     char mem[40] = {0, };
44     char *a;
45     int *b;
46     long long *c;
47     int i;
48
49     a = (char*)mem;

```

Memory Window:

address	hex	char
0xffffef04c	10 00 00 00
0xffffef050	4c f0 fe ff	L...
0xffffef054	20 9b 08 00

Registers Window:

name	value (hex)	value (decimal)	description
r0	0x1	1	
r1	0xffffef1a4	429488084	
r2	0x10	16	
r3	0xffffef04c	4294897740	
r4	0x11018	69656	
r5	0x10158	65880	
r6	0x80408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xffffef05c	4294897756	register 11 (64-bit)
r12	0xffffef078	4294897784	register 12 (64-bit)

3. b=256을 실행한 후에 0xffffef04c의 값이 0x00000100(=256)으로 바뀐 것을 확인할 수 있다.

The screenshot shows the same debugger interface after executing the instruction `b = 256;`.

Memory Window:

address	hex	char
0xffffef04c	00 01 00 00
0xffffef050	4c f0 fe ff	L...
0xffffef054	20 9b 08 00

Registers Window:

name	value (hex)	value (decimal)	description
r0	0x1	1	
r1	0xffffef1a4	429488084	
r2	0x100	256	
r3	0x100	256	
r4	0x11018	69656	
r5	0x10158	65880	
r6	0x80408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xffffef05c	4294897756	register 11 (64-bit)
r12	0xffffef078	4294897784	register 12 (64-bit)

(2) 실습 B 관련(LAB2 P20) int형 배열을 선언 및 초기화하고 그 배열의 시작주소를 char형 포인터로 강제 캐스팅한 뒤, 그 포인터변수를 이용하여 배열의 내용이 아래와 같이 되도록 C 코드를 작성해 보세요. 주석을 포함하는 코드와 결과 화면을 캡처하여 보고서에 첨부하세요.

The screenshot shows a debugger window with the following components:

- Source Window:** Displays the C code for `main.c`. The code defines an array `array` of 16 integers, casts it to a `char*` pointer `p`, and fills it with the character 'a' in a loop.
- Disassembly Window:** Shows the assembly instructions for the C code, including stack frame setup, variable initialization, and the loop logic.
- Memory Window:** Displays the memory contents of the array. It shows addresses from `0xfffff114` to `0xfffff12c`, each containing the hex value `0a` (which represents the ASCII character 'a').
- Registers Window:** Shows the state of CPU registers, including `r0` through `r15`.

The C code in the source window is as follows:

```

65 }
66 void exampleB(){
67     int array[16] = {0, }; // int형 배열 초기화
68     char *p; // 포인터 변수 선언
69     p = (char*)array; // char 형변환
70     for(int i = 0; i < 16; i += 3) // 3바이트 건너뛰기
71     {
72         *(p + i) = 'a'; // a 를 저장
73     }
74 }

```

(3) 실습 C 관련(LAB2 P26) main 함수에 example3() 함수를 추가하고 재빌드하세요. 프로그램을 실행하면 p26 아래 사진처럼 메모리가 출력된 것을 볼 수 있습니다. 왜 이러한 결과가 나왔는지 indexing 관점에서 설명해 보세요. 13/16/22/26번줄 str 명령어 수행결과에 주목하세요). 메모리 출력 결과와 함께 보고서에 첨부하세요.

L12에서 mov명령어로 r3에 1이 저장되고 str r3, [sp, #4] 명령어로 0xfffff044 에 1이 저장된다.(pre-indexing)

(0xfffff044 : 01)

L13을 수행할 당시 sp가 0xfffff040을 가리키고 있고 L15에서 mov 명령어로 r4에 저장된 2가 str r4, [sp], #4 명령어로 0xfffff044에 저장이 되고 sp도 +4 한다. (post-indexing)

(0xfffff044 : 02)

L18에서 mov 명령어로 r4에 0이 저장되고 str r4, [sp] 명령어로 0xfffff048에 r4값이 저장된다.

(0xfffff048 : 00)

L21에서 mov 명령어로 r5에 3이 저장되고 str r5, [sp, #8]! 명령어로 r5에 저장된 3이 0xfffff050에 저장된다. (auto-indexing)

(0xfffff050 : 03)

L26에서 str 명령어를 통해 r3값인 4를 0xfffff04c에 저장한다. (pre-indexing)

$(sp + (r0, LSL \#2)) = (0xfffff050 + 0xfffffff0) = (0xfffff050 - (4)) = 0xfffff04c$

(0xfffff04c : 04)

The screenshot displays a debugger interface with two main panels. The left panel shows the assembly code for a function named `example3`. The code includes instructions for adjusting the stack pointer (`sub sp, sp, #12`), storing values into memory at specific offsets from the stack pointer (`str r3, [sp, #4]`, `str r4, [sp, #4]`, `str r5, [sp, #8]!`), and performing arithmetic operations (`add sp, sp, #10`). The right panel shows a memory dump with addresses ranging from `0xfffff044` to `0xfffff050`. The values stored at these addresses are `01`, `02`, `00`, `04`, and `03` respectively, which correspond to the data stored by the instructions in the assembly code.

address	hex	char
0xfffff044	01	01
0xfffff048	02	02
0xfffff04c	00	00
0xfffff050	04	04
0xfffff054	03	03

(4) 실습 D 관련(LAB2 P33) main 함수에 example4() 함수를 추가하고 재빌드합니다. example4() 함수를 call 하자마자 스택에 저장하는 리턴 주소가 몇 번지 주소에 저장되는지 그 메모리 주소값, 그 주소에 저장된 리턴값을 디버거를 이용하여 확인하세요. 확인하는 과정을 캡처하여 설명과 함께 보고서에 첨부하세요. (main 함수에서 example4() line에 breakpoint를 걸어서 disassembly 화면을 보면 [address1] bl [address2] 코드가 보일 겁니다. address1 값이 bl 명령어가 들어 있는 메모리 주소입니다. bl 명령어 수행 후 return 해야할 주소는 [address1] + 4 입니다. bl 명령어를 수행하면 lr에 그 return 주소가 저장되고 함수 처음에 lr을 스택에 push할 겁니다. sp가 가리키는 메모리 근처에서 찾아보세요.)

STEP 1

The screenshot shows a debugger window with the following components:

- Disassembly View:** Shows the assembly code for `example4()`. Line 15 is highlighted: `bl 0x10774`. The comment indicates this is a branch to `example4`.
- Memory View:** Shows a memory dump starting at address `0x1052c`. The dump shows several lines of memory, with the first line being `0x1052c: 00 00 00 e3`.
- Registers View:** Shows the state of registers. The `lr` register (link register) is highlighted, showing its value as `0x10774`.

memory

address: 0xfffff05c, size: 4

address	hex	char
0xfffff05c	30 05 01 00	0...
0xfffff060	00 00 00 00
0xfffff064	40 0b 01 00
0xfffff068	00 00 00 00
0xfffff06c	01 00 00 00
0xfffff070	a4 f1 fe ff
0xfffff074	24 05 01 00
0xfffff078	95 f8 bc 45	...E

L15에 [address1] bl [address2] 형태로 저장된 코드를 찾을 수가 있다. address1=0x1052c, address2=0x10774 이고 0x1052c + 4 = 0x10530이 example4() 수행 후 스택으로 푸시되어 0xfffff05c에 저장된다.

(5) 실습 D 관련(LAB2 P33) example4() 함수의 i값을 계속 증가시켜보면서 i값이 얼마일 때 위의 스택에 저장된 리턴값이 깨지는 지를 확인합니다. 확인하는 과정을 캡처하여 설명과 함께 보고서에 첨부하세요. (스택의 최하위 주소를 가리키는 sp를 찾아 메모리를 주시하세요. 위에서 설명한 return 주소 lr이 메모리에서 i값으로 바뀔 때가 깨지는 상황입니다. 메모리가 깨지기 전 세 번의 루프를 포함하는 중간 결과를 캡처하세요.)

STEP 2

memory	0xffff040	0xffff05f	4
address	hex	char	
more			
0xffff040	01 00 00 00	
0xffff044	00 00 00 00	
0xffff048	00 00 00 00	
0xffff04c	00 00 00 00	
0xffff050	00 00 00 00	
0xffff054	20 9b 08 00	
0xffff058	64 f0 fe ff	d...	
0xffff05c	30 05 01 00	0...	

(1) i=0;

(2) i=1;

memory	0xffff040	0xffff05f	4
address	hex	char	
more			
0xffff040	01 00 00 00	
0xffff044	02 00 00 00	
0xffff048	03 00 00 00	
0xffff04c	00 00 00 00	
0xffff050	00 00 00 00	
0xffff054	20 9b 08 00	
0xffff058	64 f0 fe ff	d...	
0xffff05c	30 05 01 00	0...	

(3) i=2;

(4) i=3;

memory	0xffff040	0xffff05f	4
address	hex	char	
more			
0xffff040	01 00 00 00	
0xffff044	02 00 00 00	
0xffff048	03 00 00 00	
0xffff04c	04 00 00 00	
0xffff050	05 00 00 00	
0xffff054	20 9b 08 00	
0xffff058	64 f0 fe ff	d...	
0xffff05c	30 05 01 00	0...	

(5) i=4;

(6) i=5;

i=5 일 때, BUFFERSIZE가 5로 설정되어 0xffff040부터 0xffff050까지 할당되어 있기 때문에 할당된 크기를 벗어나 buffer overflow가 발생한다.(0xffff054에 저장된 값이 덮어짐)

(6) 실습 D 관련(LAB2 P33) example4() 함수 코드에서 while문 내 i=10를 5로 수정, main 함수에 example4() 함수를 추가하고 재빌드하세요. 정상 수행되는지 확인하세요. 정상적으로 수행된 결과를 캡처하여 보고서에 첨부하세요. (disassembly section을 보면 중간에 sp의 값을 저장해서 스택에 접근할 때 사용하는 다른 레지스터가 있습니다. 그 레지스터가 가리키는 메모리 주소 근처에서 찾아보세요.)

STEP 3 i값 수정 전 Overflow가 발생

0xfffff040	01 00 00 00
0xfffff044	02 00 00 00
0xfffff048	03 00 00 00
0xfffff04c	04 00 00 00
0xfffff050	05 00 00 00
0xfffff054	06 00 00 00

array = {123456}을 입력 했을 때 6이 LR값을 침범하여 값이 저장되었다

STEP 4 배열 메모리 저장 화면

0xfffff040	01 00 00 00
0xfffff044	02 00 00 00
0xfffff048	03 00 00 00
0xfffff04c	04 00 00 00
0xfffff050	05 00 00 00
0xfffff054	20 9b 00 00

버퍼의 크기가 5이고 getchar() 입력도 버퍼의 크기만큼 받음. LR값을 침범하지 않았다

STEP 5 결과화면

```

Sungwon@ubunt
Sungwon@ubuntu:~/lab2$ qemu-arm -g 8080 ./lab2
Input buffer value: 123456789
Print array: 1 2 3 4 5
Sungwon@ubuntu:~/lab2$

```

배열의 크기만큼만 저장이 되어 정상적으로 1 2 3 4 5가 출력되는 것을 확인 가능.