[숙제 6] lab2 실습 내용
제출할 자료:
(1) 실습 A 관련(LAB2 P10)
2020-2-CA-assembly Lab2 p4를 참고하여 example1() 함수 코드를 수정한 뒤 재빌드하세요. 빨간색 박스 코드에 해당되는 어셈블리 코드, step by step으로 수행할 때 메모리 변화를 캡처하고 설명과 함께 보고서에 첨부하세요.

```
int b;
int *ptr1;

ptr1 = &b;                                  0x105cc subtr3, r11, #20                example1+56
                                            0x105d0 strtr3, [r11, #-12]             example1+60
*ptr1 = 16;                               > 0x105d4 ldrtr3, [r11, #-12]             example1+64
                                            0x105d8 movtr2, #16                     example1+68
                                            0x105dc strtr2, [r3]                    example1+72

b = 256;                                    0x105e0 movtr3, #256t; 0x100            example1+76
                                            0x105e4 strtr3, [r11, #-20]t; 0xffffffec example1+80

                                            0x105e8 nopttt; (mov r0, r0)            example1+84
                                            0x105ec ldrtr3, [pc, #28]t; 0x10610 <example1+124>example1+88
                                            0x105f0 ldrtr2, [r3]                    example1+92
                                            0x105f4 ldrtr3, [r11, #-8]              example1+96
                                            0x105f8 eorstr2, r3, r2                 example1+100
                                            0x105fc movtr3, #0                      example1+104
```

memory

0xfffef078	0xfffef097	4

address	hex
more	
0xfffef078	98 0f 01 00
0xfffef07c	74 f0 fe ff
0xfffef080	78 f0 fe ff
0xfffef084	20 9b 08 00
0xfffef088	94 f0 fe ff
0xfffef08c	88 05 01 00
0xfffef090	00 00 00 00
0xfffef094	c0 0a 01 00
more	

<그림1>

memory

0xfffef078	0xfffef097	4

address	hex
more	
0xfffef078	10 00 00 00
0xfffef07c	74 f0 fe ff
0xfffef080	78 f0 fe ff
0xfffef084	20 9b 08 00
0xfffef088	94 f0 fe ff
0xfffef08c	88 05 01 00
0xfffef090	00 00 00 00
0xfffef094	c0 0a 01 00
more	

<그림2>

memory

0xfffef078	0xfffef097	4

address	hex
more	
0xfffef078	00 01 00 00
0xfffef07c	74 f0 fe ff
0xfffef080	78 f0 fe ff
0xfffef084	20 9b 08 00
0xfffef088	94 f0 fe ff
0xfffef08c	88 05 01 00
0xfffef090	00 00 00 00
0xfffef094	c0 0a 01 00
more	

<그림3>

ptr1과 b가 가리키고 있는 메모리 주소는 0xfffef078이다.
그림 1에서는 아직 값이 들어가지 않은 상황으로 0x980f0100의 값이 저장되어있다.
그림 2는 *ptr1 = 16이 실행된 결과로 ptr1이 가리키는 메모리에 0x10000000 의 값이 저장되었다. little endian으로 0x00000010 = 16(10진수)이다.
그림 3은 b=256이 실행된 결과로 b가 가리키는 메모리에 0x00010000 의 값이 저장되었다. little endian으로 0x00000100 = 256(10진수)이다.
두 그림 모두 b가 지역 변수이기에 스택을 사용하고 있다. 그리고 두 그림 모두 같은 동작을 수행한다. 차이점은 그림 2의 동작을 할 때는 해당 메모리의 값을 레지스터에 저장하고, 레지스터를 통해 메모리를 참조한다. 그와 조금 다르게 그림 3의 동작을 할 때는 레지스터에 따로 주소를 저장하지 않는다.

(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 C코드를 작성해 보세요. 주석을 포함하는 코드와 결과 화면을 캡처하여 보고서에 첨부하세요.

열기(O) main.c ~/programming/CS/lab2 저장(S)

```
46  int i;
47
48  a = (char*)mem;
49  for (i = 0; i < 3; i++)
50  {
51    *(a + i) = 10;
52  }
53
54  b = (int*)(mem + 4);
55  for (i = 0; i < 3; i++)
56  {
57    *(b + i) = 1;;
58  }
59
60  c = (long long*)(mem + 16);
61  for (i = 0; i < 3; i++)
62  {
63    *(c + i) = 12;
64  }*/
65
66  int mem[40] = {0, }; //int형 배열 40만큼 선언, 초기화
67
68  for (int i = 0; i < 16; i++)
69    *((char*)mem + i) = (i%3 == 0) ? 10 : 0;
70    //배열 시작 주소 char형 포인터로 강제 형 변환
71    //조건 참이면 10, 거짓이면 0
72 }
73
74
75 void example4()
76 {
77  int buf[5] = {0, };
78  int i = 0;
79
80  printf("\nInput buffer value: ");
81  while (i < 10)
```

C 탭 너비: 2 74행, 1열 삽입

<수정된 C코드와 주석>

memory

0xffffeefe4	0xffffef003	8

address	hex
more	
0xffffeefe4	0a 00 00 0a 00 00 0a 00
0xffffeefec	00 0a 00 00 0a 00 00 0a
0xffffeeff4	00 00 00 00 00 00 00 00
0xffffeeffc	00 00 00 00 00 00 00 00
more	

<결과>

(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.

memory

0xffffef074	0xffffef093	4

address	hex
more	
0xffffef074	02 00 00 00
0xffffef078	00 00 00 00
0xffffef07c	04 00 00 00
0xffffef080	03 00 00 00
0xffffef084	7c 05 01 00
0xffffef088	30 0f 01 00
0xffffef08c	58 01 01 00
0xffffef090	00 00 00 00
more	

원래 0xfff2074의 자리에 값 10이 저장되어 있어야한다. 13번 줄에서 값1을 pre-indexing 방식으로 저장하고, 16번 줄에서 값 2를 post-indexing 방식으로 저장한다. post-indexing으로 해서 4를 더한 자리에 저장하는 것이 아닌, 원래 sp자리에 저장하게 되어 값이 덮어쓰기 되었다.

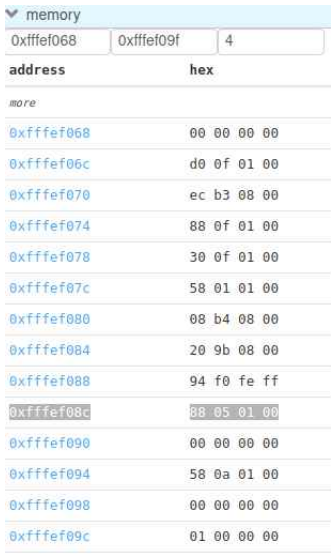
16번 줄에서 post-indexing으로 변경된 메모리 주소로 19번 줄에서 값 0을 저장하게 되어 0xffffe078자리에 0이 저장되었다.

22번 줄에서 auto-indexing으로 먼저 sp+80이 계산되어 0xffffef080의 자리에 값 30이 저장되었다. 그리고 sp값이 sp+8로 변경되었다.

26번 줄에서 pre-indexing으로 [sp, r0, lsl # 2]로 결정되는 메모리의 주소는 sp - r0 * 2 * 2 = sp - 4 이고 그 자리는 0xffffef07c이다. 이 자리에 값 4가 저장되게 된다.

(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] <example4> 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.)

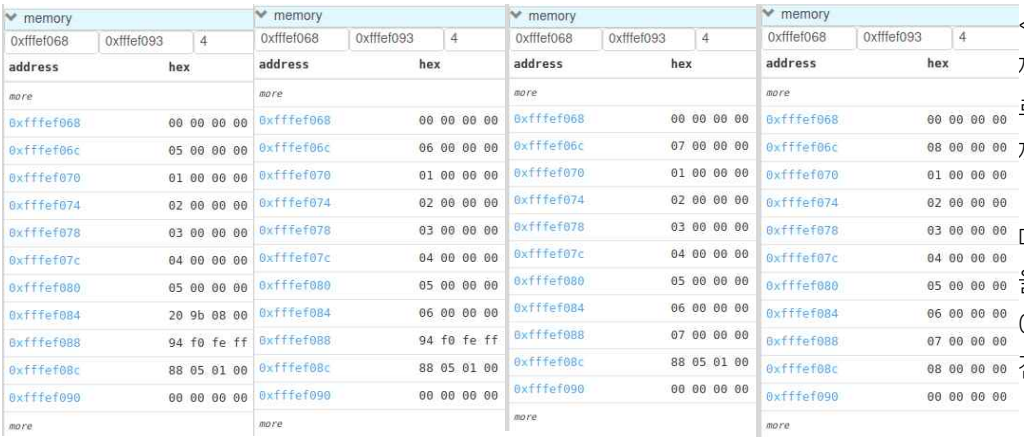


<disassembly 화면> <메모리>

14번 줄에 [address1] bl [address2] <example4> 형태를 찾을 수 있고 대입해보면, [address1] = 0x10584 [address2] = 0x106e4이다. return후 수행 할 명령어 주소는 0x10584 + 4인 0x10588 임을 알 수 있고, 이는 example4가 실행된 후, 스택으로 푸시되어 메모리 0xfffef08c에 저장되어 있는 모습을 볼 수 있다.

(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.)



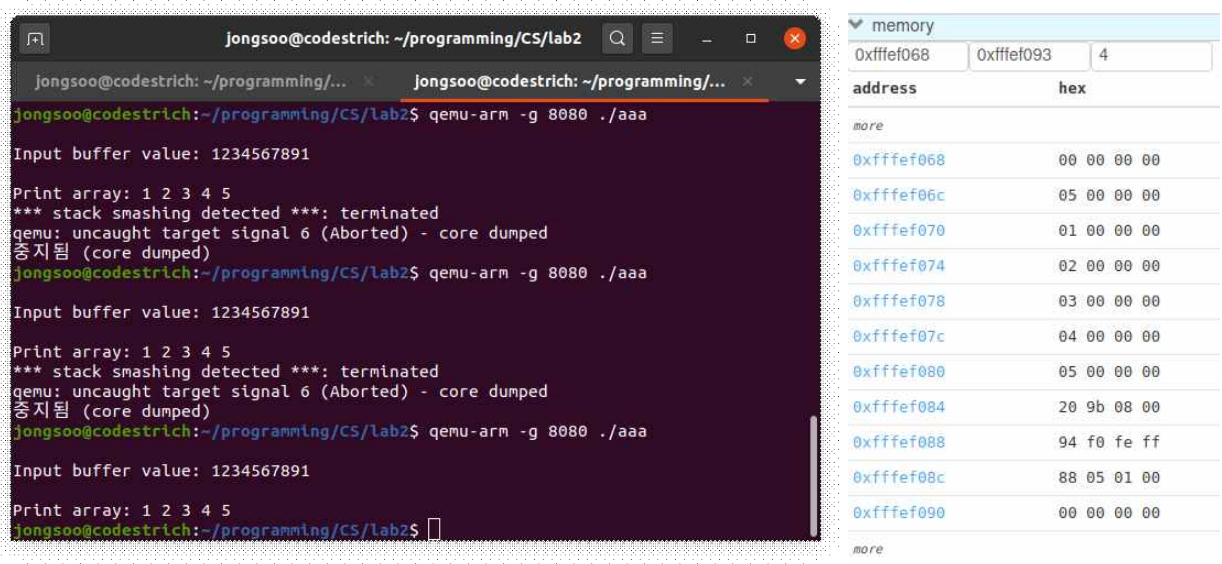
<좌측부터 메모리가 깨지기 3 루프 전, 2 루프 전, 1 루프 전, 깨진 후의 사진>

마지막 사진에서 lr 값을 저장하던 메모리 0xfffef08c의 값이 깨진 것을 볼 수 있다.

(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.)



<16번 줄까지는 오류가 발생했고, 17번 줄 부터는 정상적으로 실행되었다.> <lr 값을 침범하지 않았다.>

화면 캡쳐 결과들을 2020-2-ca-hw@q.ssu.ac.kr로 제출하시오.

제출마감: 11월 1일(일) 23:59분(중간고사기간이어서 최대한 늦추었습니다)