**CSED211 Homework2**

20210643 김현준

**1. Exercise 3.59 on page 347**

x와 y의 sign bit 과 을 를 곱해 128bit로 확장하고, 아래 식을 이용하여 곱을 계산하는 형태의 알고리즘이다.

로 놓자. 그러면

부분은 128bit의 범위를 넘어가므로 overflow를 무시하면,

이 된다.

이는 p=과 같은 형태이다.

따라서 이를 계산하면, 128bit의 곱셈 결과값을 얻을 수 있다.

Asembly)

1 store\_prod: ; %rdi에 dest, %rsi에 x, %rdx에 y가 들어간다.

2 movq %rdx, %rax ; %rax는 y가 된다.

3 cqto ; Signed Extend %rax=>%rdx:%rax, %rdx=(=1일때), 0(=0일때)

4 movq %rsi , %rcx ; %rcx는 x가 된다.

5 sarq $63, %rcx ; %rcx를 63번 right shift 한다. %rcx=x\_h(=1일때), 0(=0d일때)이다.

6 imulq %rax, %rcx ; %rcx=%rax\*%rcx 계산을 한다. 즉, %rcx=y\*()이다.

7 imulq %rsi , %rdx ; %rdx=%rsi\*%rdx 계산을 한다. 즉, %rdx=x\*()이다.

8 addq %rdx, %rcx ; %rcx=%rdx+%rcx 계산을 한다. 즉, %rcx= y\*()이다.

9 mulq %rsi ; Unsigned full multiply => %rdx:%rax=%rax\*%rsi=ux\*uy이다.

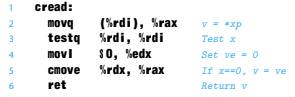
10 addq %rcx, %rdx ; %rdx=%rcx+%rdx=x\*()+y\*()+uy\*ux이다.

11 movq %rax, (%rdi) ; %rdi에 LSB쪽 64bits의 값을 넣는다. (lower 64 bits)

12 movq %rdx, 8(%rdi) ; 8(%rdi)에 MSB쪽 64bits의 값을 넣는다. (higher 64 bits)

13 ret ; 리턴한다.

**2. Exercise 3.61 on page 349**



Section 3.6.6을 보면 위와 같은 어셈블리가 나온다. Null ptr의 주소를 참조하는 문제를 해결하기 위하여, 삼항 연산자를 반대로 바꾸어 !xp ? 0 : \*xp로 바꾸어 보았다.

================================

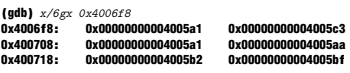
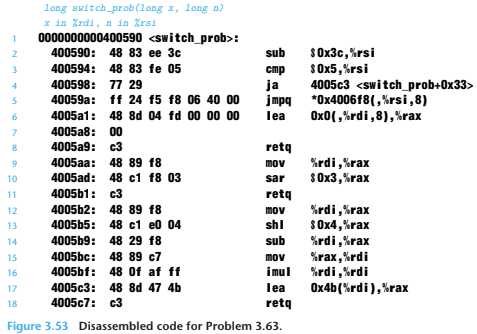
long cread\_alt (long \*xp) {

return (!xp ? 0 : \*xp);

}

================================

**3. Exercise 3.63 on page 350**



우선, %rsi에서 0x3c=60을 빼 주었고, 5와 비교하여 크면 4005c3으로 가서 리턴한다. 그리고 5번째 줄에서 indirect jump를 하는데, jump table에서 $rsi가 0일 때 4005a1으로, 1일 때 4005c3으로, 2일 때 4005a1으로, 3일 때 4005aa로, 4일 때 4005b2로, 5일 때 4005bf로 가서 각 연산을 수행하게 된다. 따라서 이를 c로 표현하면 아래와 같다.

주의해야 할 점은, 4005bf와 400c3에는 ret이 없어 break가 없다는 점이다.

===============code below===============

long switch\_prob(long x, long n){

long result = x;

switch(n){

//4005a1

case 60:

case 62:

result = x\*8;

break;

//4005aa

case 63:

result = (x>>3);

break;

//4005b2

case 64:

result = (x<<4)-x;

x = result;

//4005bf

case 65:

result = x\*x;

//4005c3

default:

result = x+0x4b;

}

return result;

}

======================================

**4. Modified Exercise 3.64 on page 352 (solve A & B for the following code)**

leaq (%rsi, %rsi, 4), %rax

leaq (%rsi, %rax, 2), %rax

movq %rdi, %rsi

salq $6, %rsi

leaq (%rsi, %rdi, 2), %rdi

addq %rax, %rdi

addq %rdi, %rdx

movq A( , %rdx , 8), %rax

movq %rax , (%rcx)

movl $2640, %eax

ret

**A)** Equation 3.1: 에서 three dimensions로 확장하면,

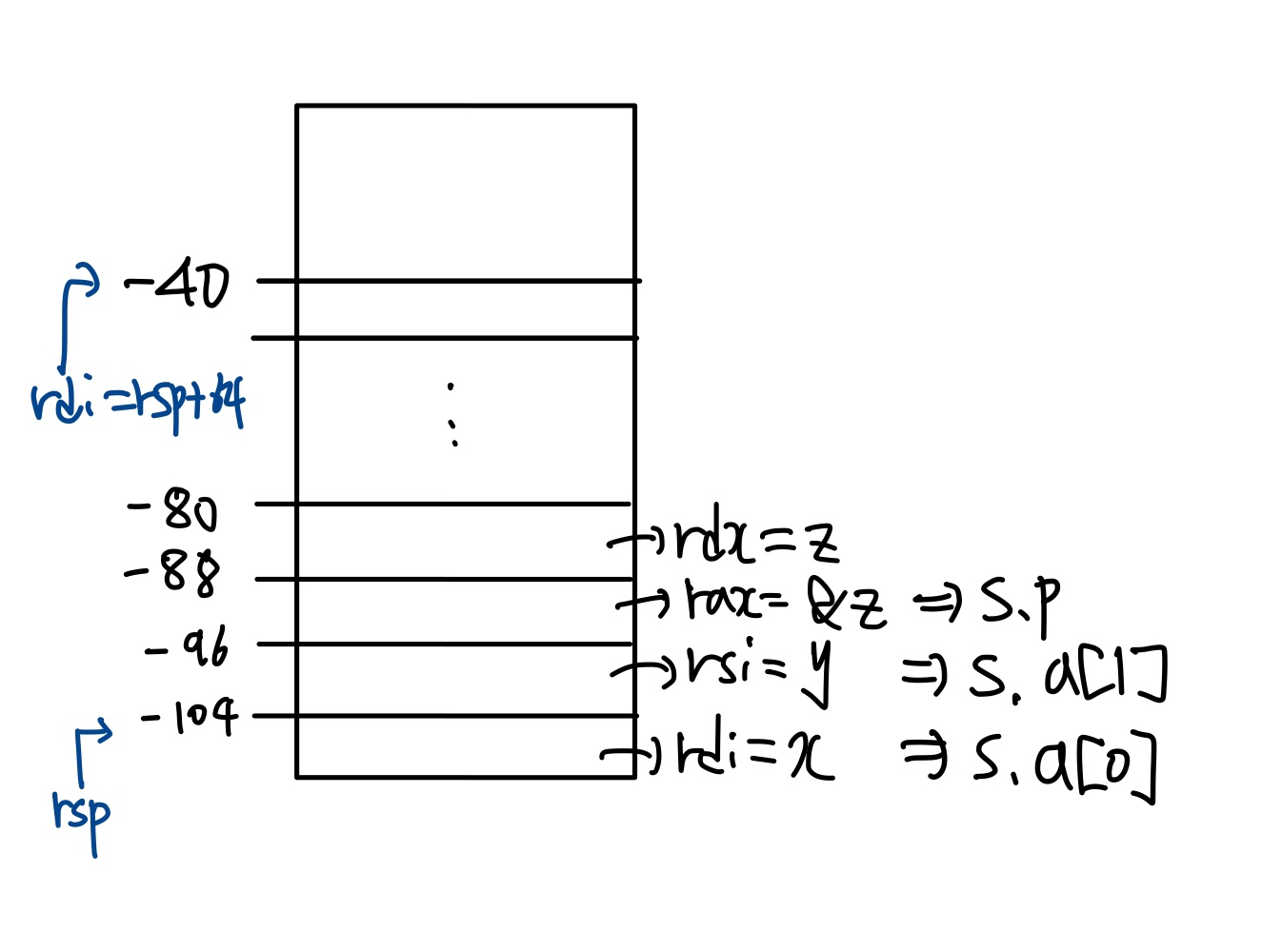
**B)** 위 어셈블리의 첫째 둘째 줄에서 11j가 계산되고, 3~5번째 줄에서 66i가 계산된다. 그리고, A[i][j][k]의 주소는 2640이고, 이다.

따라서 T=11이 되고, 이 된다. 그리고 6\*11\*8\*R=2640 => R=5이다.

따라서 답: R=5, T=11, S=6

**5. Exercise 3.67 on page 354**

**A.** process를 호출하기 전에는 스택에 그림과 같이 z, &z, y, x가 들어있다.

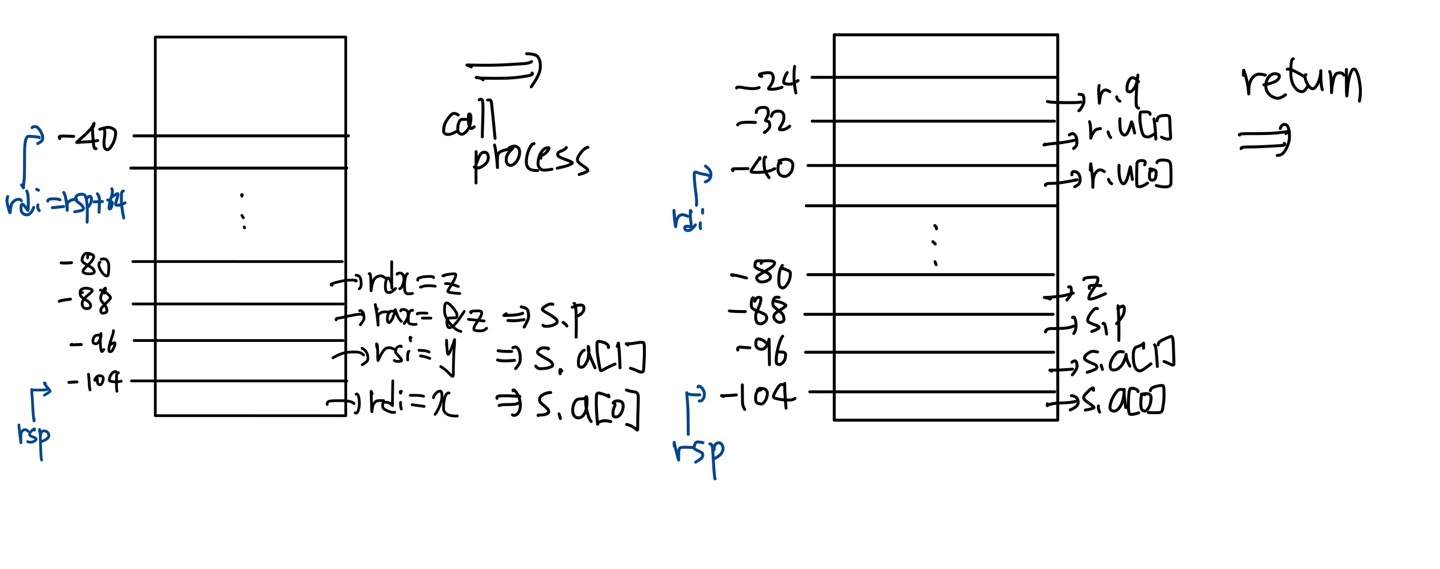


**B.** eval 함수는 %rsp+64를 %rdi에 저장하여 process함수의 인자로 전달한다.

**C.** process의 두 번째 줄인 24(%rsp) 와 같이, rsp를 이용하여 접근한다.

**D.** process의 3줄, 5줄로 %rdx와 %rcx에 저장한 다음, eval로부터 인자로 받은 %rdi를 이용한다. 8줄, 9줄과 같이 %rdi의 주소를 이용하여 각 값을 해당하는 메모리에 저장한다.

**E.** 아래의 그림과 같고, %rsp로 부터의 거리를 이용하여 접근한다. eval의 10줄과 같이 %rsp+72와 같은 방법으로 접근한다.



**F.** 이를 통해 알 수 있는 점은 주소+거리를 인자로 주어 호출이 되는 함수는, 먼저 값을 저장하고 나서 다시 주소를 리턴하기 때문에 구조체의 값들이 전달이 된 다음에 다시 리턴이 되는 특성을 보인다.

**6. Exercise 3.69 on page 357**

**A.** test에서 1줄에 %rsi+288을 %ecx에 넣는다. 그리고 다시 %ecx에 %rsi를 더한다. 그리고 %rax에 40\*%rdi+%rsi를 넣고 %rdx에 40\*%rdi+8+%rdx가 된다. 따라서 ap의 주소가 40i+bp+8이므로, A\_struct의 하나의 element가 40byte이고, B\_struct의 align은 8byte임을 알 수 있다. 따라서 288-8=280을 40byte로 나누면 7개 이므로, CNT의 값이 7임을 알 수 있다. 따라서 답: 7

**B.** 8번째 줄에서 16+8%rdx+%rax에 %rcx를 넣는다. 여기서 C코드를 생각해봤을 때, ap->x[ap->idx]에서, 8(40i+bp+8)+40i+bp+8+8로 나타낼 수 있다. 그런데 ap의 시작 주소에 8을 더하므로, idx는 long 형이라고 추측할 수 있고, (40i+bp+8)에 8이 곱해져 있으므로 8은 x의 element의 size라고 추측할 수 있다. 그런데 A의 크기가 40이므로, 그 안에서 x는 4개가 들어가므로, A\_struct는 아래와 같은 구조를 가지는 것으로 생각할 수 있다.

================

typedef struct {

long idx;

long x[4];

} a\_struct;

================

**7. Exercise 3.70 on page 358**

**A.** offsets - e1.p: 0, e1.y: 8, e2.x: 0, e2.next: 8

**B.** 16 bytes

**C.** 어셈블리 각 줄을 해석해 보면,

첫 줄에서 %rax=\*(%rdi+8)이고, 둘째, 셋째 줄에서 %rdx=\*(%rdi+8), %rdx=\*(\*(%rdi+8)), 넷째 줄에서 sub하여 %rdx=\*(\*(%rdi+8))-\*(\*(%rdi+8)+8)이 된다. 최종적으로 %rdi=\*(\*(%rdi+8))-\*(\*(%rdi+8)+8)이므로, 이를 C 코드로 바꾸어 보면 아래와 같은 코드를 구할 수 있었다.

void proc (union ele \*up) {

up->e2.x = \*(up->e2.next->e1.p) - up->e2.next->e1.y;

}

**8. Exercise 3.72 on page 359**

**A.** $rsp에서 %rax를 빼서 를 구하는데, %rax는 30+8n에 -16을 and연산하므로 최종적으로 이다. n이 홀수면 , 짝수면 이다.

**B.** 에 15를 더하고 -16과 and하므로 이다.

**C.** 먼저, 이므로 16으로 align되어 있을 것으로 해석할 수 있다. 그리고 은 array p의 끝과 사이의 extra space이다. 따라서 이 최소일 때는 array p의 끝과 사이가 최대일 때이다. 이때 의 차이는 작아져야 한다. 따라서 n은 짝수여야 한다. 따라서 이때 이다.

그런데 이 0이면, 이므로 0이 아니고 따라서 이다. 이 때는 에서 는 16으로 mod한 값이 1이 되어야 한다. 결론적으로, n이 짝수이고 %16==1 인 일 때 이 minimum 값 1을 가진다.

다음으로 이 최대일 때는 위와 반대로 차이는 커야 하고, n은 홀수가 되며 이다. 는 16의 배수이다. 따라서, n이 홀수이고 %16==0이 될 때 최대값 24이다.

**D.** p는 16으로 align되어 있고, 는 8n만큼의 space를 주는 16의 배수 중에서 가장 작은 수가 된다.