**HW2**

20210774 김주은

**1. Exercise 3.59 on page 347**

먼저, store\_prod의 코드를 보면 mulq 명령어를 볼 수 있는데, 이는 unsigned multiplication을 한다. 이를 통해 unsigned multiplication을 사용하여 xy값을 구하는 것을 알 수 있다.

다음으로, unsigned와 signed의 관계를 살펴본다.

에 따라서,

여기서, 은 overflow가 발생한 부분이므로 무시하면

이고, 이다.

따라서 아래 assembly code에서도 , 값을 사용하여 값을 구한다.

store\_prod: // dest in %rdi, x in %rsi, y in %rdx

movq %rdx, %rax // rax register에 y값 저장 -> %rax = y

cqto // cqto : 64bit %rax를 128bit %rdx : %rax로 확장하는 instruction

// (int128\_t)y

// y의 sign bit (= 이 1이면 1111….1이므로 -1 (= -1\*1)

// y의 sign bit (= 이 0이면 000…….0이므로 0 (= -1\*0)

// 즉, if y < 0 , %rdx = -1 이고, else if y > 0, %rdx = 0

// 결론적으로, %rdx = -1 \*

move %rsi, %rcx. // rcx register에 x값 저장 -> %rcx = x

sarq $63, %rcx // rcx에 들어있는 값(x)을 63bit만큼 Shift arithmetic right

// x의 sign bit 이 1이면 1111….1이므로 -1

// x의 sign bit 이 0이면 000…….0이므로 0

// 즉, if x < 0 , %rcx = -1 이고, else if x > 0, %rcx = 0

// 결론적으로, %rcx = -1 \*

imulq %rax, %rcx // %rcx \*= %rax

// %rcx = %rcx \* %rax = (-1\*) \* y

imulq %rsi, %rdx. // %rdx \*= %rsi

// %rdx = %rdx \* %rsi = (-1\*) \* x

addq %rdx, %rcx // %rcx = %rcx + %rdx = (-1\*) \* y + (-1\*) \* x

mulq %rsi // unsigned multiplication

// one operand is %rax, the other operand is given in the instruction, %rsi

// product is stored in %rdx(high order part) and %rax(low order part) -> full 128-bit result

// %rdx:%rax 에 ux\*uy 저장

addq %rcx, %rdx // %rdx = %rcx + %rdx = ((-1\*) \* y + (-1\*) \* x) + high 64 bit of ux\*uy

// 결론적으로, %rdx:%rax 에 ((-1\*) \* y + (-1\*) \* x) \* 가 더해지는 셈

movq %rax, (%rdi) // %rdx:%rax에서 %rax값(low 64 bit)만 %rdi의 값이 가리키는 메모리에 저장

// (%rdi) : \*dest

movq %rdx, 8(%rdi) // %rdx:%rax에서 %rdx값(high 64 bit만 %rdi+8의 값이 가리키는 메모리에 저장

ret

**2. Exercise 3.61 on page 349.**

Attach your c-code and compiled assembly code to show that the generated code uses a conditional move.

long cread(long\* xp)

{

return (xp ? \*xp : 0);

}

교재의 3.6.6 section에서 위 함수의 assembly code를 아래와 같이 보여준다.

cread:

movq (%rdi), %rax

testq %rdi, %rdi

movl $0, %edx

cmove %rdx, %rax

ret

assembly code를 분석해보면 %rax에 %rdi값이 가리키는 곳의 값을 대입하는 과정에서 \*xp를 계산함을 알 수 있는데, xp가 null pointer인 경우 문제가 발생할 수 있다. 이러한 문제를 방지하기 위해서는 movq (%rdi), %rax를 할 때 %rdi가 null pointer 인 경우가 발생하지 않으면 될 것이다.

그러므로, 미리 %rdi값이 0인지 확인하여 0의 값 저장한 곳을 가리키도록 한다.

즉, movl $0, %eax을 먼저 진행한 후

testq %rdi, %rdi

cmovne (%rdi), %rax

가 진행되어야 한다.

그러므로 c코드는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 이를 컴파일하면 다음과 같다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**3. Exercise 3.63 on page 350**

<jump table>

텍스트이(가) 표시된 사진

자동 생성된 설명

<code>

테이블이(가) 표시된 사진

자동 생성된 설명

먼저 sub 0x3c, %rsi를 통해 n – 60을 하고 cmp $0x5, %rsi를 통해 5와 n-60을 비교한다.

여기서, n-60>5이면 4005c3으로 가는 것을 알 수 있는데, 이를 통해 n>65에 대해서는 default로 jump함을 알 수 있고, 4005cs 코드들이 default 에 해당함을 알 수 있다.

그 다음 구문인 jmpq \*0x4006f8(,%rsi,8)을 통해 jumptable의 시작주소 + 8 rsi만큼, 즉 rsi에 따라 jump table의 각각 다른 값으로 이동함을 알 수 있다.

4005a1은 jump table에 따라 x = 0,2일 때이며 0+8rdi = 8x 를 Return한다.

4005aa는 jump table에 따라 x = 3일 때이며 x>>3을 return한다.

4005b2는 jump table에 따라 x= 4일 때이며 x = x << 4 – x 을 진행하고 바로 종료하지 않는다.

4005bf는 jump table에 따라 x = 5일 때이며 x \*= x 를 진행하고 바로 종료하지 않는다.

그리고 default의 경우도 진행하며 x + 0x4b의 값을 return한다.

이를 반영한 c 코드이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

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

A.

3.1 equation =

는 시작주소, L은 type크기, c는 column값이다.

A[i][j][k]에서 먼저 A[i]에 대해 L\*(S\*T) 크기만큼 공간이 배정되므로 A[i]의 시작 주소는

이다.

A[i][j]에는 L\*T만큼 공간이 배정되므로 A[i][j] 의 시작 주소는 이다.

마지막으로, A[i][j][k] 에는 L 만큼의 공간이 배정되므로, A[i][j][k] 의 시작주소는 이다.

즉, 이다.

B.

먼저,

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

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

에서 %rsi는 j이므로, %rax에 11\*%rsi값인 11j를 대입한다.

movq %rdi, %rsi // rsi에 i를 대입한다.

salq $6, %rsi // rsi에 64\*i를 대입한다.

leaq (%rsi, %rdi, 2), %rdi // %rdi에 2%rdi+%rsi를 대입하므로, %rdi에 2i+64i = 66i 대입

addq %rax, %rdi // %rax + %rdi = 11j + 66i 를 %rdi에 대입  
addq %rdi, %rdx // %rdx + %rdi = k + 11j + 66i 를 %rdx에 대입

movq A( , %rdx , 8), %rax // %rax에 8\*rdx + A가 가리키는 곳의 값을 넣는다.

movq %rax , (%rcx) // (%rcx)는 dest가 가리키는 곳

// 이 구문은 \*dest = A[i][j][k]에 해당된다.

// 즉, 8\*rdx + A가 가리키는 곳의 값이 A[i][j][k]이므로

// A + 8(k + 11j + 66i) = &A[i][j][k]

A 문제에 따라 와 대응시켜보면

L = 8, S = 6, T = 11 임을 알 수 있다.

마지막으로 R의 값만 구하면 되는데,

movl $2640, %eax 구문에서 sizeof(A)가 2640이라고 했으므로,

6\*11\*8\*R = 2640 , R = 5 임을 알 수 있다.

즉, R = 5, S = 6, T = 11 이다.

**5. Exercise 3.67 on page 354**

A. values on the stack prior to calling process.

x in %rdi, y in %rsi, z in %rdx

<The codes Before calling process>

subq $104, %rsp

// 먼저, %rsp값을 104만큼 빼주므로 104 크기의 stack 구조가 생긴다.

movq %rdx, 24(%rsp)

// %rsp+24 이 가리키는 위치에 %rdx값인 z가 저장된다.

leaq 24(%rsp), %rax

// %rax에 %rsp+24값이 저장

movq %rdi, (%rsp)

// %rsp가 가리키는 위치에 %rdi값인 x가 저장된다.

movq %rsi, 8(%rsp)

// %rsp+8이 가리키는 위치에 %rsi값인 y가 저장된다.

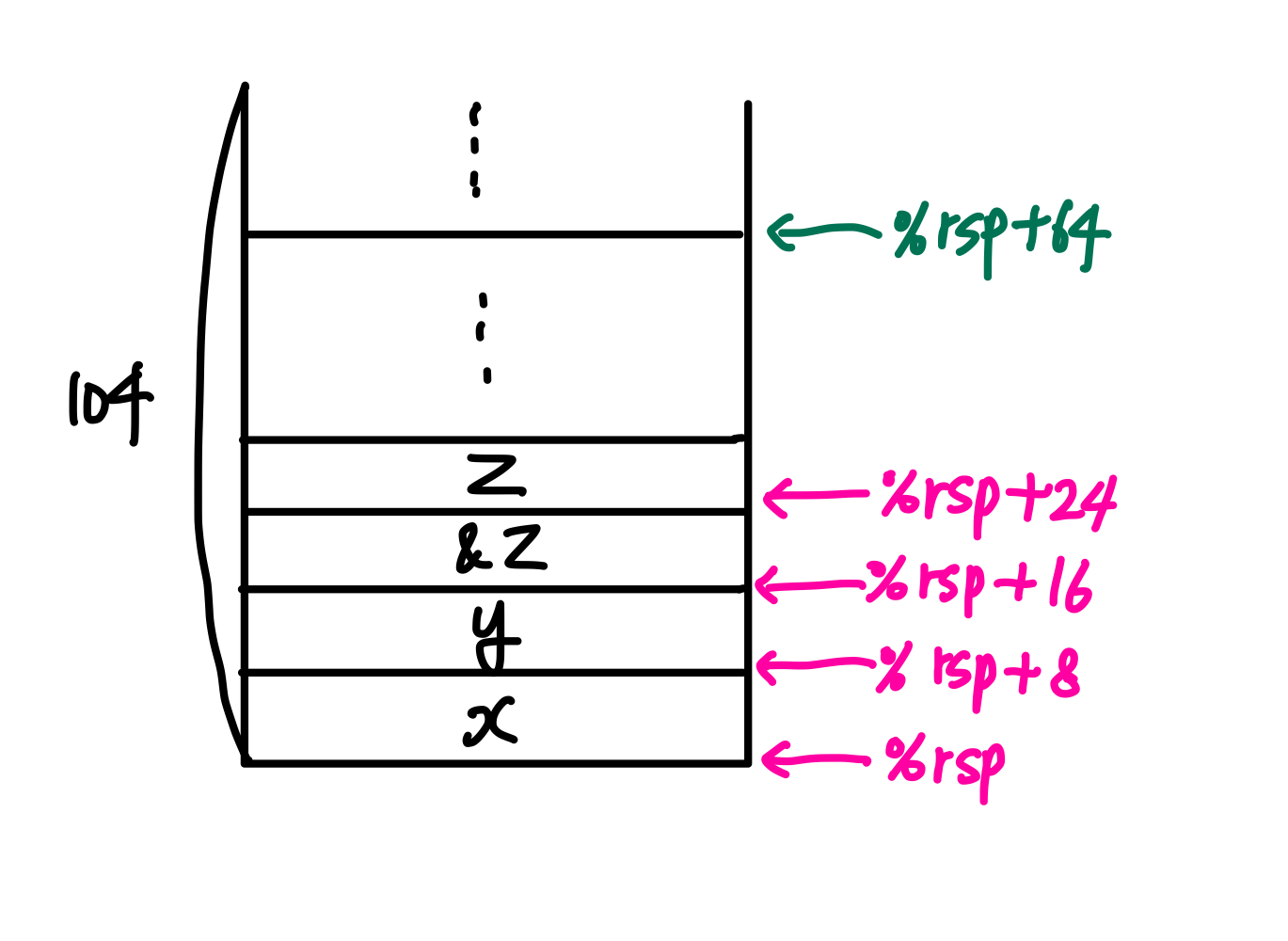
movq %rax, 16(%rsp)

// %rsp+16이 가리키는 위치에 %rax값인 %rsp+24 값이 저장되고, 이는 z가 들어가 있는 주소이다.

leaq 64(%rsp), %rdi

// %rdi에 %rsp+64 값을 저장한다.

이를 토대로 stack 구조를 그리면 다음과 같다.



B.

eval 함수에서 process 함수로 인자로 전달할 때 하나의 인자를 전달한다.

calling process 전에 leaq 64(%rsp), %rdi 구문을 통해서 %rdi 값에 인자로 넘길 값을 저장하여 eval 함수에서 process 함수로 인자를 넘긴다. 즉, %rsp + 64 가 인자로 넘기는 값이다.

C.

Process의 c code를 보면 구조체 변수 r의 각 원소에 s 값을 대입하는 것을 알 수 있다. 그리고 구조체 s의 원소들 값은 이미 A에서 언급한 stack구조의 %rsp부터 %rsp+24 들이 가리키는 곳에 저장되어 있다. 그리고 process를 부르면서 rsp가 -8만큼 감소되어 있으므로 이를 참고하여 해석해야 한다.

실제 코드에서도

movq %rdi, %rax

// %rax에 %rdi, 즉 인자 값 %rsp+64가 저장됨

movq 24(%rsp), %rdx

// %rdx에 %rsp+24를 넣는다. 즉 &z값을 넣는다.

movq (%rdx), %rdx

// %rdx에는 %rdx의 값이 가리키는 곳의 값을 저장한다. 이는 z값을 넣는다.

movq 16(%rsp), %rcx

// %rcx에 %rsp + 16가 가리키는 값을 넣는다. 즉 y를 넣는다.

movq %rcx, (%rdi)

// %rdi가 가리키는 곳에 %rcx값인 y값을 넣는다.

movq 8(%rsp), %rcx

// %rcx가 가리키는 곳에 %rsp+8이 가리키는 값을 넣는다. 즉, x를 넣는다.

movq %rcx, 8(%rdi)

// %rdi+8이 가리키는 곳에 x값을 넣는다.

movq %rdx, 16(%rdi)

// %rdi+16이 가리키는 곳에 %rdx값인 z값을 넣는다.

ret

즉, %rsp+16, %rsp+8, %rsp+24 등을 사용하여 s값에 접근하여 값을 가져온 후 c 에 복사하는 형식이다. 그러므로, s의 원소에 접근하는 방법은 %rsp+offset 형식으로 접근한다고 할 수 있다.

D.

위 코드에서 봤듯이

movq %rcx, (%rdi)

// %rdi가 가리키는 곳에 %rcx값인 y값을 넣는다.

movq %rcx, 8(%rdi)

// %rdi+8이 가리키는 곳에 x값을 넣는다.

movq %rdx, 16(%rdi)

// %rdi+16이 가리키는 곳에 %rdx값인 z값을 넣는다.

이 부분들을 보게 되면 %rdi는 process에서 인자로 받은 %rsp+64 값이므로, 여기서 0, 8, 16을 더한 주솟값이 가리키는 곳에 각각 result structure r 의 원소들을 저장한다.

E.

Process에서 return값인 rax에 %rsp+64를 대입한 후 return했다.

Process를 call한 이후의 eval의 코드를 보게 되면

move 72(%rsp), %rax

addq 64(%rsp), %rax

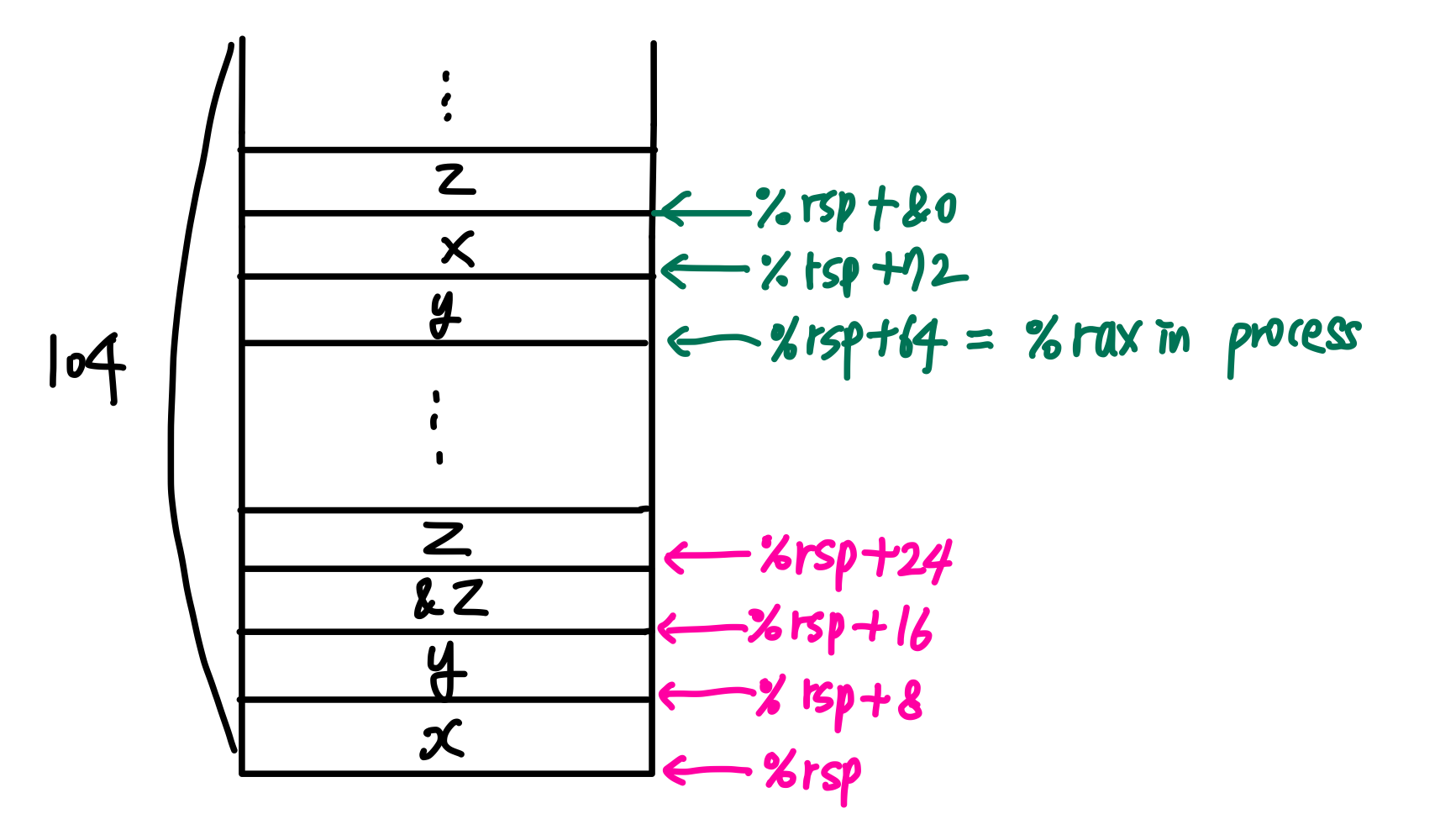
addq 80(%rsp), %rax

addq $104, %rsp

ret

인데,

%rsp+72, %rsp+62, %rsp+80을 통해서 r의 원소들에 접근하는 것을 볼 수 있다.



F.

함수는 먼저 stack에 공간을 잡고, callee 함수에 그 공간의 시작 주소를 넘긴다. 그리고 callee 함수는 그 공간에 값들을 저장하고 다시 그 공간의 시작주소를 caller 함수에 return 한다.

**6. Exercise 3.69 on page 357**

A.

test 함수 코드를 보면,

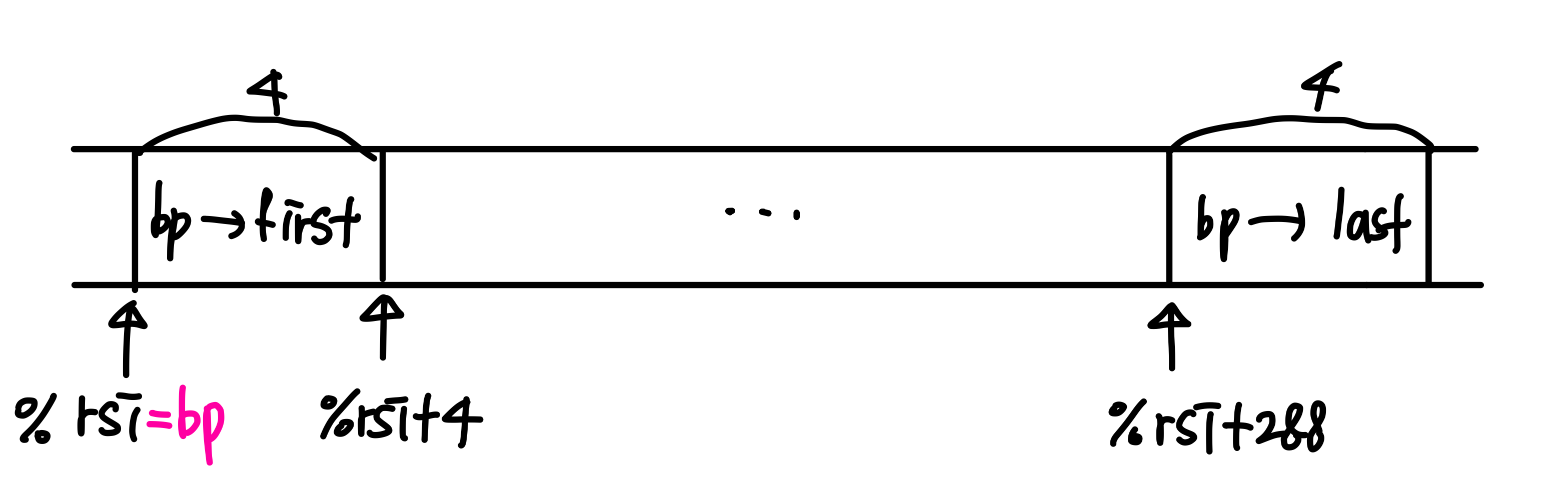
mov 0x120(%rsi), %ecx // %ecx에 %rsi+288이 가리키는 주소의 값을 넣는다

add (%rsi), %ecx // %ecx에 %rsi가 가리키는 주소의 값을 더해준다.

// %ecx = %rsi+288의 값 + %rsi의 값

// 그리고 n = bp->first + bp->last와 같다.

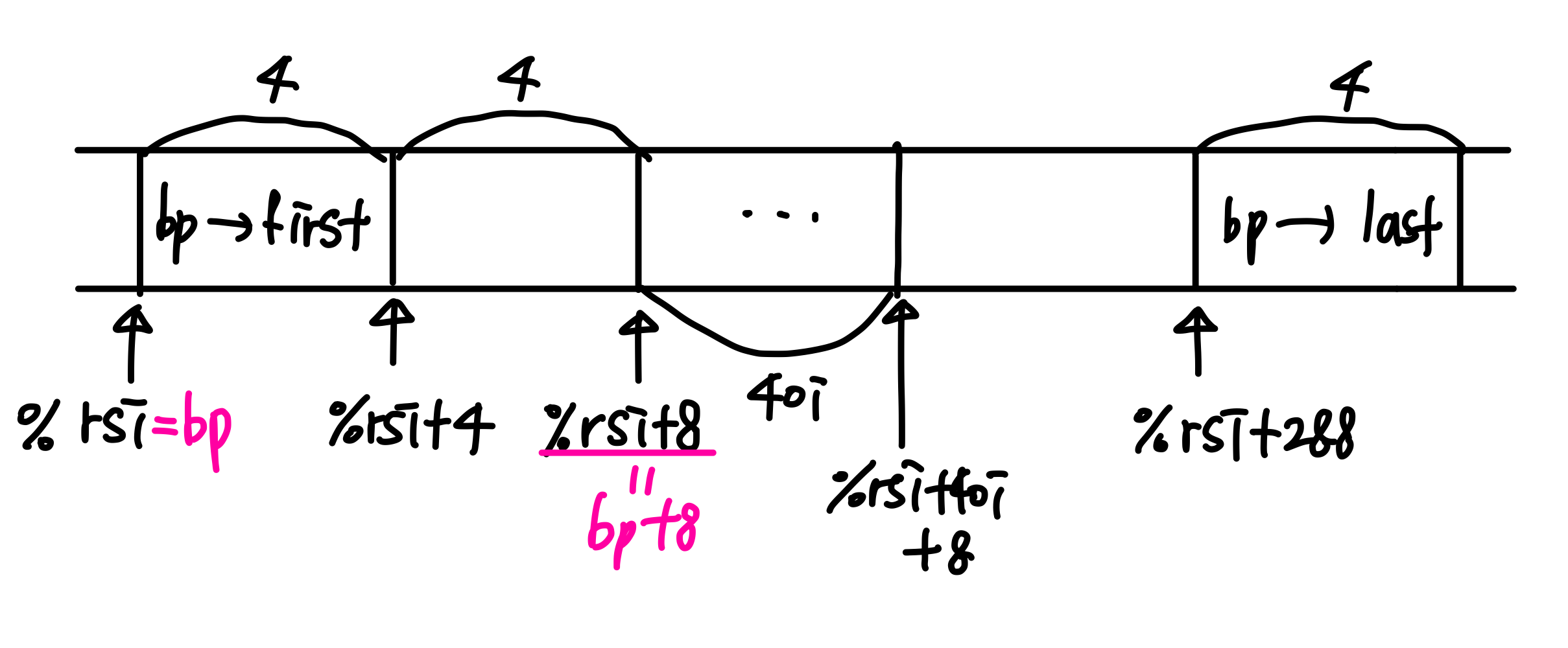
// %rsi는 bp->first의 주솟값, %rsi+288은 bp->last의 주솟값이다.



lea (%rdi, %rdi, 4), %rax

lea (%rsi, %rax, 8), %rax // %rax에 40\*%rdi + %rsi값을 대입하고, 이는 bp + 40i 이다.

mov 0x8(%rax), %rdx // %rdx 에 %rax+8, 즉 bp+40i+8 주소에 해당하는 값을 대입한다.



// bp->first의 끝주소인 %rsi+4가 아닌 %rsi+8에서부터 40i를 하는데

//%rsi+4~%rsi+8의 internal padding을 통해 align한 것이다.

//즉 bp+8부터 ap의 주소이고, 40\*i를 하는 것을 통해

//a\_struct의 크기가 40byte임을 알 수 있다.

//a\_struct a[CNT]을 선언한 것이 %rsi+8~%rsi+288에 들어가있으므로

//총 배열의 크기는 280byte이므로 CNT = 280/40 = 7이다.(A답)

답은 7이다.

B.

mov 0x8(%rax), %rdx // %rdx에는 %rax+8, 즉 bp+40i+8 주소의 값이 대입된다.

// 해당 값은 ap가 가리키는 곳의 값이다.

// 즉, ap가 가리키는 a\_struct 구조체 변수의 첫번째 값이다.

mov %rcx, 0x10(%rax, %rdx, 8) // 8\*%rdx+%rax+16의 주소에 %rcx 값인 n이 대입된다.

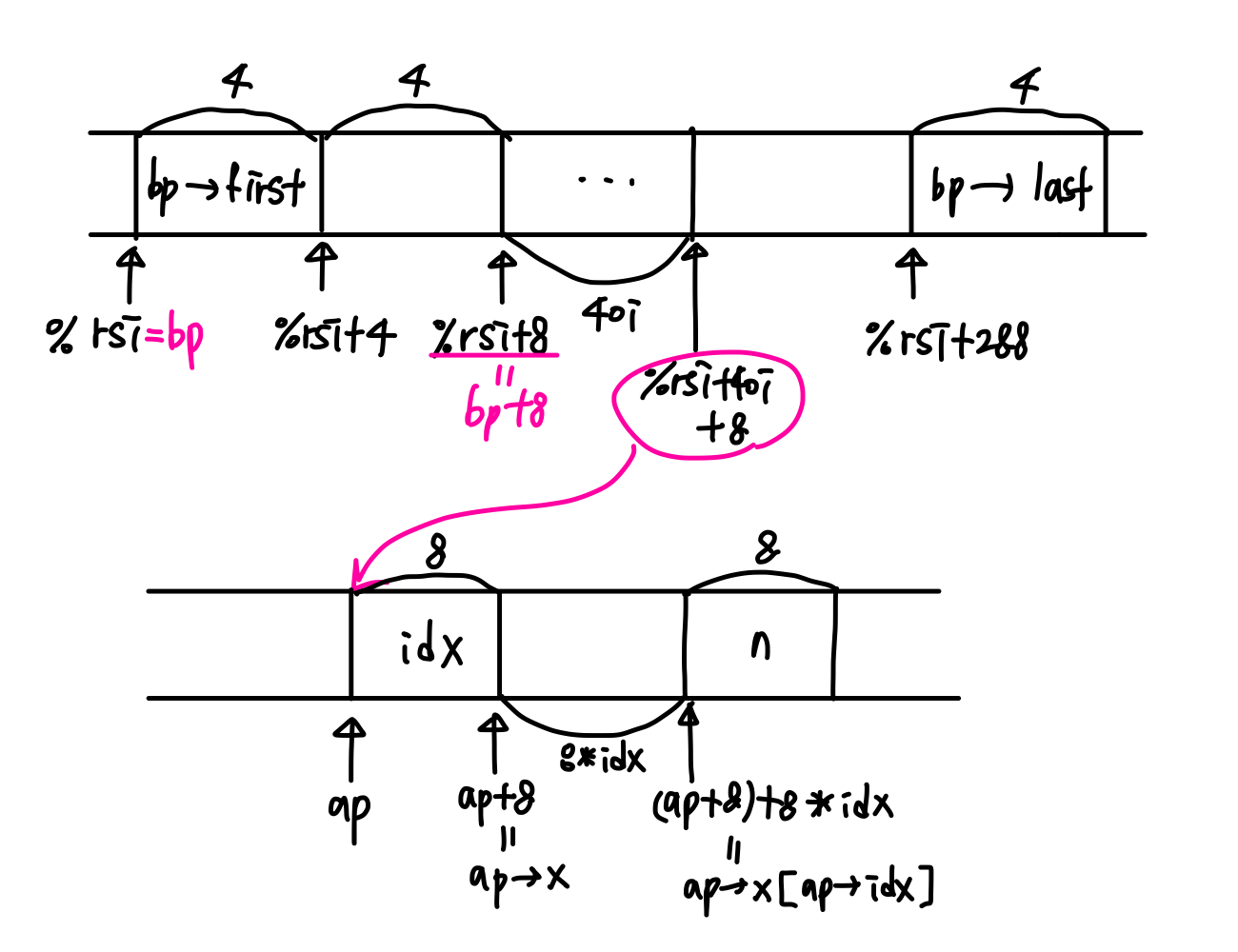
// 8\*%rdx + %rax + 8 + 8 = 8\*%rdx + (%rsi + 40i + 8) + 8

// = 8\*%rdx + ap + 8

// ap->x[ap->idx] = n; 부분에 해당하므로

// rdx는 idx이며, ap + 8이 ap->x의 시작주소임을 알 수 있다.

// 즉, (ap + 8) + 8\*idx



다음과 같이 ap에 대해 표현할 수 있다. 이로써 a\_struct 구조체의 첫번째 변수는 idx이고, 8byte이므로 long이며, 두번째 변수는 long type의 배열임을 알 수 있다.

그리고 총 a\_struct 구조체의 size는 40이므로 40-8 = 32만큼의 size를 배열이 차지하므로 배열의 element 개수는 32/8 = 4 이다.

그러므로,

typedef struct {

long idx,

long x[4]

} a\_struct

이다.

**7. Exercise 3.70 on page 358**

A.

e1.p : 0

e1.y : 8

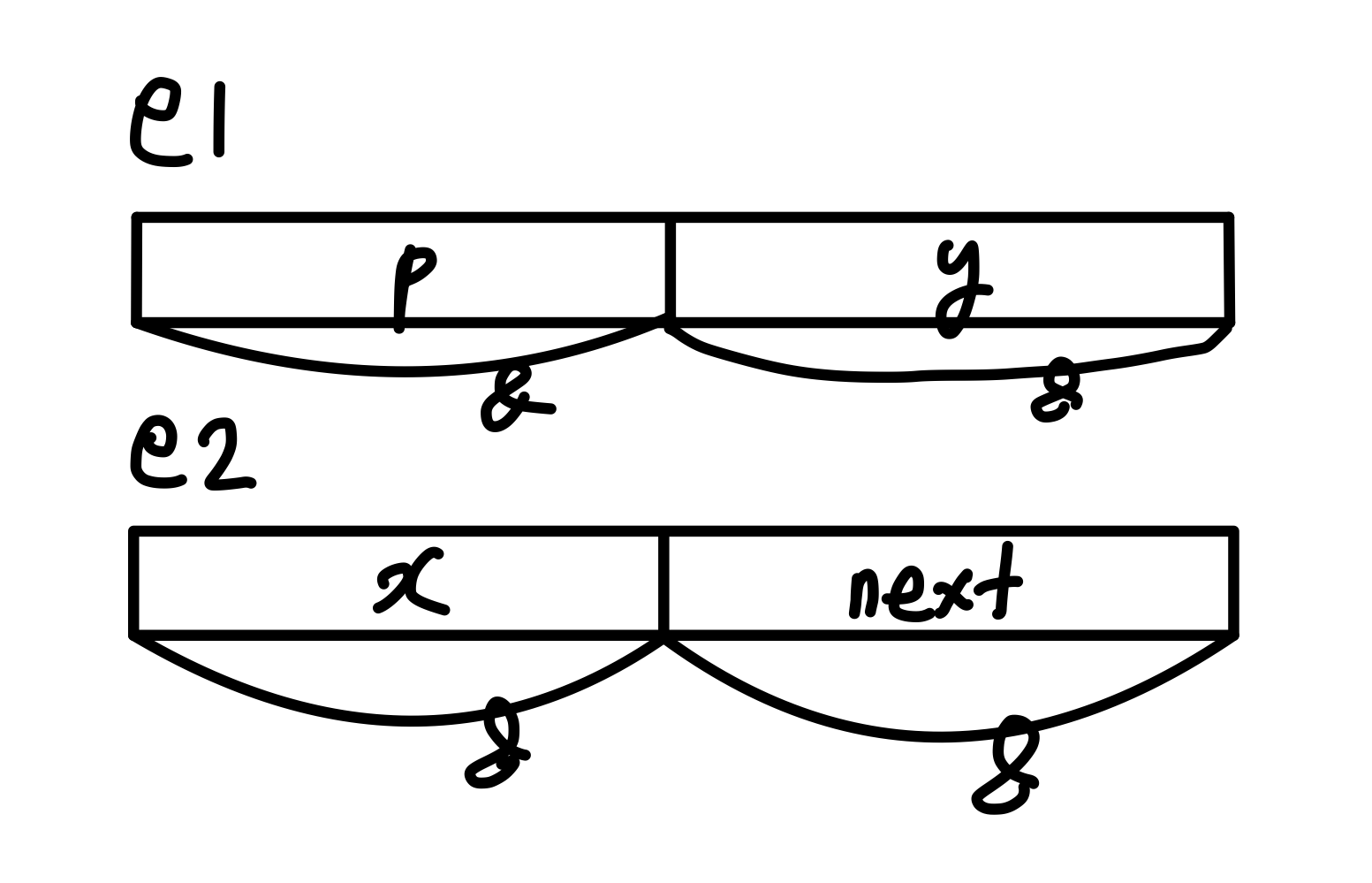
e2.x : 0

e2.next : 8

B.

Total 16 bytes

C.



proc:

movq 8(%rdi), %rax // %rax에 %rdi+8, 즉 up+8 에 있는 값을 대입한다.

// \*(up+8) : next값 or y값

movq (%rax), %rdx // %rdx에 \*(\*(up+8))을 대입

// \*y는 불가능하므로 \*(up+8) = up->e2.next

// \*(\*(up+8)) = \*(up->e2.next)

movq (%rdx), %rdx // %rdx 에 \*(\*(up->e2.next)) 을 대입한다.

// up->e2.next가 가리키는 union의 e1.p가 가리키는 값

// \*(\*(up->e2.next).e1.p)

subq 8(%rax), %rdx // %rdx 에 (%rdx값 – %rax+8가 가리키는 값)을 대입

// %rax = up->e2.next = \*(up->e2.next).e1.p의 주소

// %rax+8 = up->e2.next + 8 = \*(up->e2.next).e1.y의 주소

// \*(\*(up->e2.next).e1.p) – \*(up->e2.next).e1.y

movq %rdx, (%rdi) // %rdi는 up이므로

// up가 가리키는 값에 위에서 계산된 %rdx값 대입

// up가 가리키는 값 = up->e2.x

// up->e2.x에 %rdx값 대입

그러므로, 정리하면

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

**8. Exercise 3.72 on page 359**

A.

leaq 30(,%rdi,8), %rax

andq $-16, %rax

subq %rax, %rsp

%rax = 8\*%rdi + 30을 한 뒤, 값을 -16과 and연산을 취한 후 %rsp에 저장된 s1 값에서 계산된 값을 빼준다.

먼저 %rdi에 n이 들어가 있으므로, 8n + 30 이 되고 이를 -16과 and 연산을 취한다.

-16은 0xFFFFFFF0이므로

이다.

3

Let n = 2a+1

Let n = 2a

그러므로,

1) n is odd

2) n is even

B.

leaq 15(%rsp), %r8

andq $-16, %r8

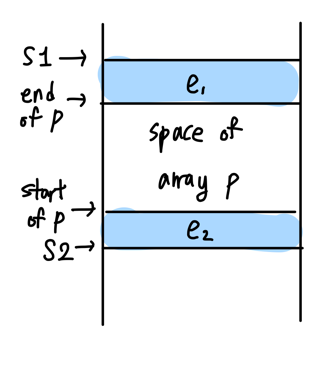
%rsp값이 s2이므로, s2+15한 값에 -16과 and 연산을 취한 값이 p이다.

C.

B에 따르면,

If n is odd, s1 – s2 = 8n + 24

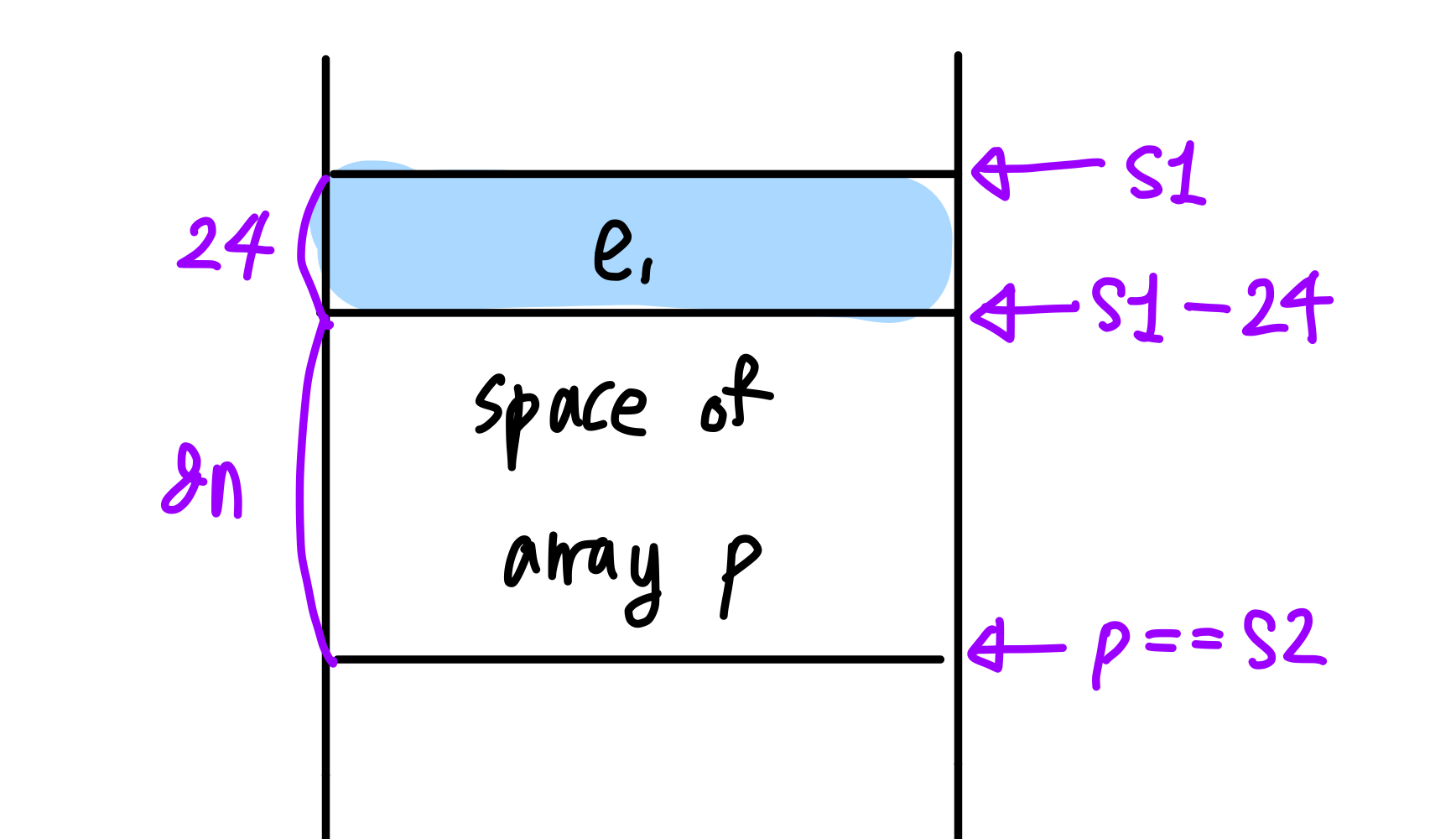
If n is even, s1 – s2 = 8n + 16



array p가 차지하는 공간은 n\*sizeof(long\*)이므로 8n이다.

(1) e1의 maximum value

e1의 maximum value = 8n+24 – 8n = 24 이다.



즉, e2 값이 0이고 s2와 p 값이 동일한 경우이다. 그리고 s1-s2 == 8n+24 이 되어야 하므로 n 은 odd이다. Odd 이므로 n값을 2a+1로 표현한다.

이며, p == s2여야 하기 때문에 s2는 16의 배수가 되어야 한다.

S2가 16의 배수이므로 16c라고 표현한다.

S1 = S2 + 8n + 24 = 16c + 8n + 24 = 16c + 8(2a+1) + 24 = 16c + 16a + 32 이므로 S1은 16의 배수가 되어야 한다.

(2) e1의 minimum value

e1의 minimum value를 0으로 가정해본다.

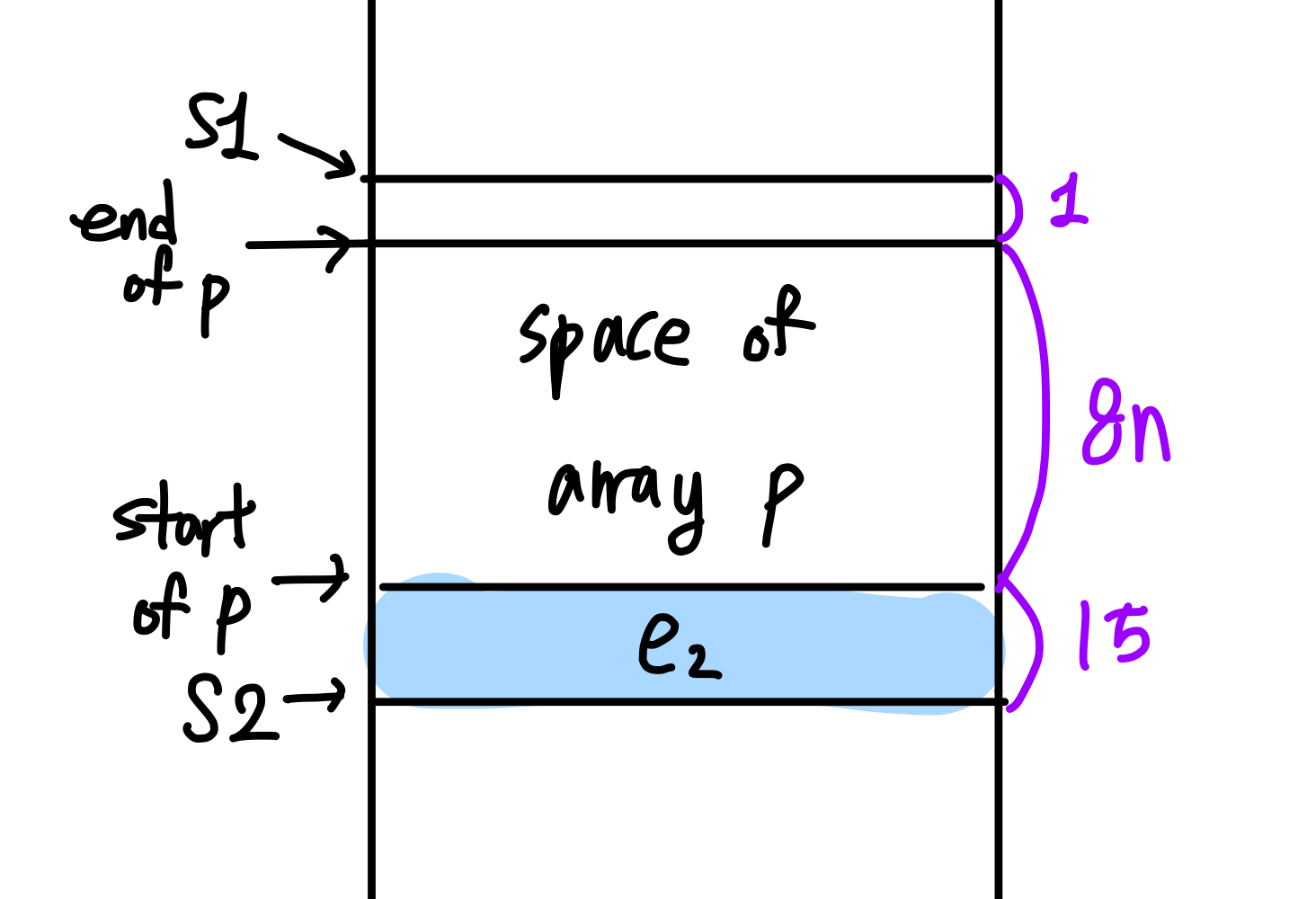
e1이 0이면 e2 = 24 or 16, p = (s2 + 15) & 0xFFFFFFF0

즉, e2 = 16이면 p = s2 + 16 = (s2 + 15 ) & 0xFFFFFFF0, e2 = 24면, p = s2 + 24 = (s2 + 15) & 0xFFFFFFF0

그러나 이를 만족하는 s2는 존재하지 않으므로 minimum value는 0이 될 수 없다.

e1의 minimum value를 1로 가정한다.

e1이 1이면 e2 = 23 or 15이고 이 중에 e2 = 15인 경우에 p = s2 + 15 = (s2 + 15) & 0xFFFFFFF0을 만족하는 s2가 존재하므로 minimum value는 1인 것을 알 수 있다.



minimum value가 1이고 e2 = 15이기 때문에 n = even이다. even이므로 2a로 표현해본다.

p = (s2 + 15) & 이며, p = s2+15이므로, s2 + 15 가 16의 배수이면 등식이 성립한다.

그러므로 , s2 는 16으로 나눴을 때 나머지가 1이어야 한다. S2 = 16c + 1

s1 = s2 + 16 + 8n = 16c + 16 + 8n + 1 = 16c + 16 + 8\*2a + 1 이므로, s1도 16으로 나누었을 때 나머지가 1이어야 한다.

D.

P = (s2+ 15) & 0xFFFFFFF0이므로 항상 16의 배수임을 알 수 있다. 즉, p는 16에 의해 align된다.

S2 의 특징은 8n size의 array p를 위한 공간을 보존하는 16의 배수이다.