1. Exercise 3.60 on page 348

long loop(long x, long n)

{

long result = 0;

long mask;

for (mask = 1; mask != 0;mask=mask<<n)

result |= (x & mask);

}

1. 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\_alt(long \*xp)

{

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

}이건 컴파일 했으나 if로 만 컴파일 됨 .

long cread\_alt(long \*xp)

{

int zero = 0;

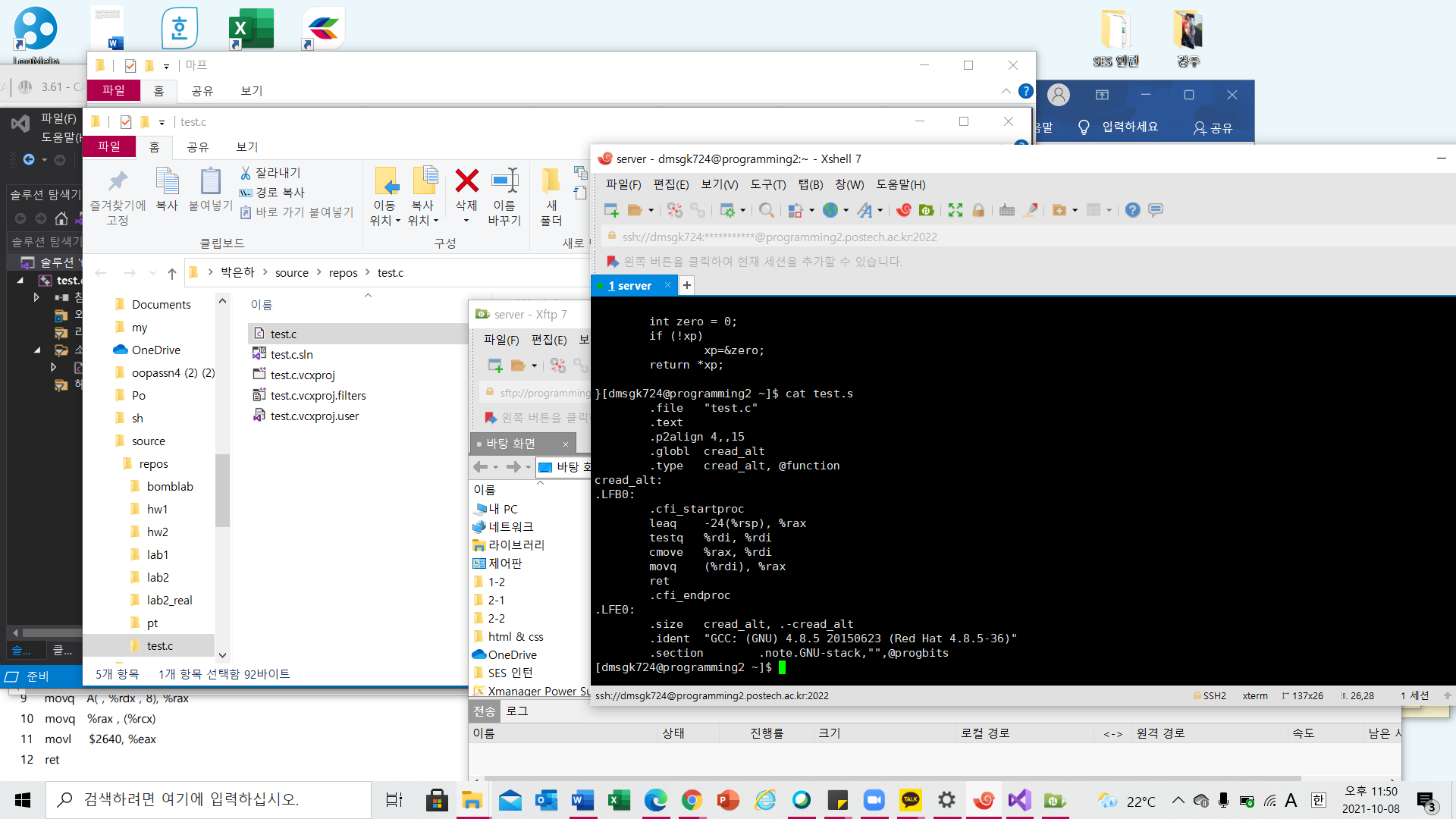
if (!xp)

xp=&zero;

return \*xp;

}

따라서 이방법을 썼다.



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

1. Exercises 3.62 and 3.63 show differently generated assembly codes for ‘switch’ statement in c program. Explain the factors that affect the compiler to generate assembly code like 3.62 or 3.63.

3.62같은 경우는 직접 점프를 이용했고 3.63같은 경우는 간접점프를 이용했다. 직접 점프는 점프목적지가 인스트럭션의 일부로 인코딩 될 때 사용되고 간접 점프는 점프대상을 레지스터나 메모리 위치로부터 읽어야되면 사용된다.

간접 점프의 경우 다음 실행할 코드가 변수에 의해 결정된다면 유용하게 사용될 수 있다. Jump table에서 변수의 값으로 다음 인스트럭션의 주소를 쉽게 찾을 수 있기 때문이다. 따라서 브랜치수가 많은데 동일한 변수에 의해서 다음 인스트럭션의 주소가 결정된다면 간접 점프를 쓰는 것이 좋다.

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

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

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

4 movq %rdi, %rsi

5 salq $6, %rsi

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

7 addq %rax, %rdi

8 addq %rdi, %rdx

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

10 movq %rax , (%rcx)

11 movl $2640, %eax

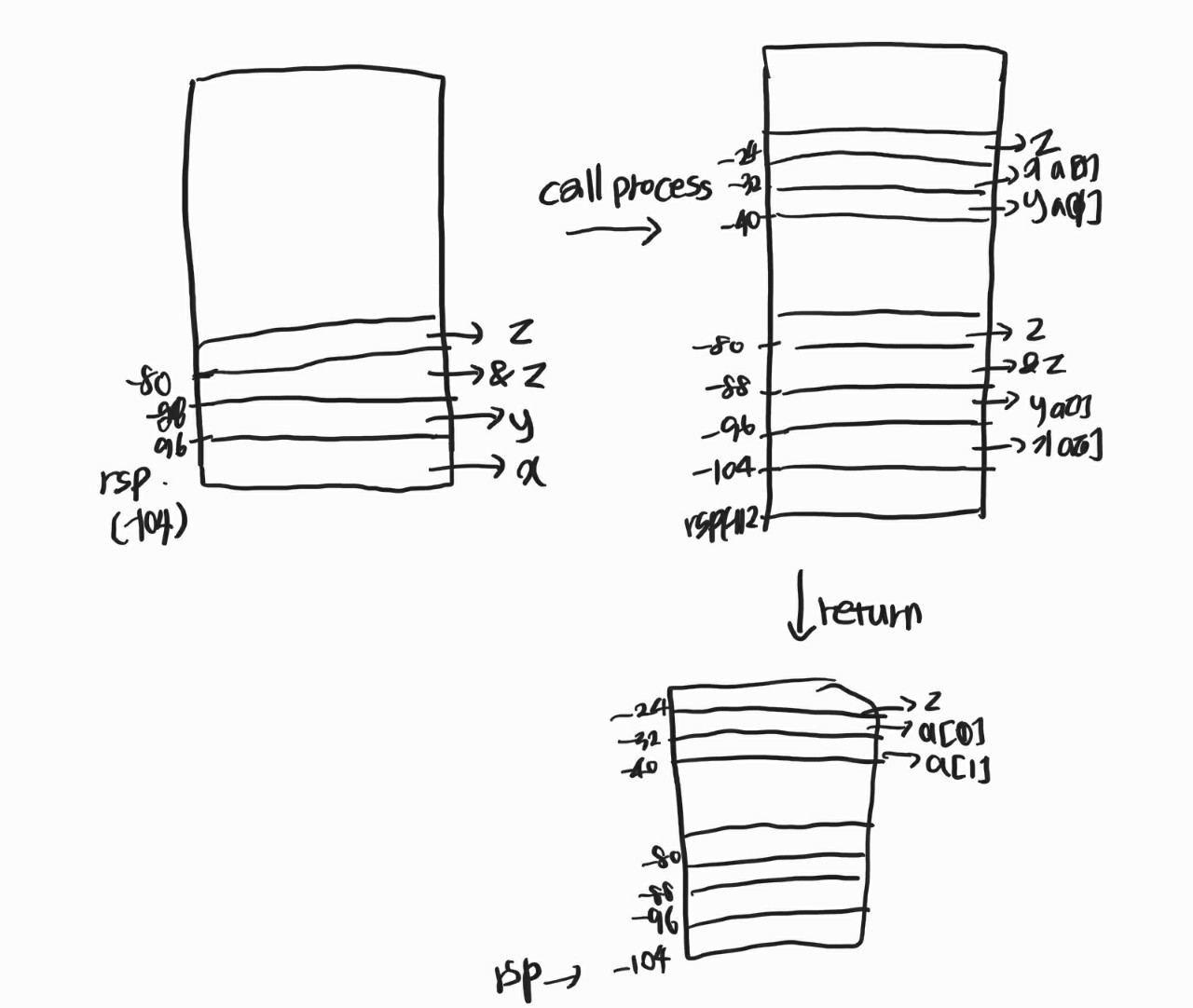
12 ret

1. &A[i][j][k]=A+sizeof(\*\*\*A)(i\*S\*T+j\*T+k)
2. R=5 T=11 S=6

8(66i+11j+k)+A가 A[i][j][k]의 주소이고 A의 size가 2640임.

따라서 T=11이고 T\*s=66이므로 S는6이다. 6\*11\*8\*R이 2640이므로 R은 5이다.

1. Exercise 3.67 on page 354



a. process를 호출하기 전 z, &z, y,x가 스택에 저장되어있다.

b. %rsp+64라는 주소값을 rdi register에 저장하여 넘겨준다.

c. rsp를통해 접근한다.

d. %rcx와 %rdx에 저장한후 argument로 넘겨받은 rdi의 주소를 통해 각각의 값을 주소값에해당하는 메모리 공간에 저장한다.

e. rsp로부터의 위치를 통해 접근한다. 위의 그림에서 보듯이 stack pointer를 이용해 %rsp+72, %rsp+64, %rsp+80과 같은 방식으로 접근한다.

f. caller는 address와 space를 주고 호출되는 함수는 data를 저장 한 후 다시 address를 리턴하므로 구조체의 값들이 전달되고, 리턴된다.

1. Exercise 3.69 on page 357
2. CNT=7

First와 last사이 288바이트가 있다.

Ap의 주소가 40i+bp+8인데 이는 a배열의 element하나가 40byte이고 alignment는 8byte로 되었음을 알 수 있다. 따라서 288byte에서 first가 해당하는 영역 8을 빼면 280이 되고 이를 40으로 나누면 CNT의 값이 7임을 알 수 있다.

1. Ap->x[ap->idx]=8(40i+bp+8)+40i+bp+8+8로 나타낼 수 있다. 이때 a의 크기는 40이다. Ap의 시작주소에 8이 더해지므로 idx는 long형이라고 생각할 수 있다. 8(40i+bp+8)에서 앞에 곱해진 8은 x 배열의 element의 size라고 볼수 있고 따라서 x는 long형 element이다. A의 크기가 40이므로 x는 4개 있어야 함 따라서 a\_struct는 다음과 같은 구조를 갖는다.

Typedef struct{

Long idx;

Long x[4];

} a\_struct

1. Exercise 3.70 on page 358

A. a. 0 b. 8 c.0 d.8

B. 16

C. up->e2.x=\*(\*(up->e2.next).e1.p)-\*(up2->next).e1.y

1. Exercise 3.72 on page 359
2. S2=S1-(8\*n+30)&0xFFFFFFF0
3. P=(15+s2)&0xFFFFFF0
4. E1은 s1과 p배열 끝부분 사이

S2의 경우 n이 홀수인 경우 s2=s1-(24+8n) 짝수인 경우는 s2=s1-(16+8n)

&P[0]=(15+s2)&0xffffff0이다. 따라서 p는 16으로 aligned 되어있다..

E1이 최솟값일때 s2로부터 가장 멀리 p가 떨어져있다. 이때 n은 짝수여야 한다.n은 짝수 일 때 s1으로부터 s2가 더 적게 떨어져 있기 때문이다. E1+e2가 따라서 16이된다. 이때 e1이 0이면 s2와 p가 같게 되므로 e1은 0이될 수 없고 따라서 e1은 1이 되어야 한다. 이 경우는 s2에 15를 더해야 16의 배수가 되는 경우이므로 s2는 16으로 나누었을 때 1이되는 수이다. s1=s2+8n+16이므로 s1또한 16으로 나누었을 때 1이되는 수이다.

E1이 최댓값일 때 e2가 최소가 되어야 하므로 s2로부터 p가 가장 가까이 있어야 한다. e1=24인데, 이때 n은 홀수이다. 이때는s2에 15를 더할 필요가 없기 때문에 s2가 16의 배수이다. 따라서 s1=s2+8n+24이므로 s1%16==0이되는 수이다.

1. P는 16으로 aligned되고, s2는 8n의 space를 주는 16의 배수 중 가장 작은 수이다.