**CSED211 Main-HW3**

20210643 HyunJune Kim

**1. Practice Problem 3.46 on Page 318. In C code, ‘buf[4]’ is changed to**

**‘buf[8]’. In disassembly, ‘sub $0x10, %rsp’ is changed to ‘sub ‘$0x18, %rsp’.**

**Input string is “0123456789012345678901234567890123”**

**A.** rbx는 0x01234567ABCDEF이고, push되어서 값이 STACK에 들어가고 rsp가 0x8만큼 빼진 다음, sub 명령어로 0x10만큼 빼어졌으므로 아래와 같이 나타난다.

|  |  |
| --- | --- |
| 00 00 00 00 00 40 00 76 |  |
| 01 23 45 67 89 AB CD EF |  |
|  |  |
|  |  |
|  | (rsp) 위치 |

**B.** input string이 34개의 정수로 구성되어 있으므로, 빈 공간에 24byte만큼을 채우고 그 아래 주소의 STACK에도 10byte를 더 채우게 된다. String이므로 null character 00까지 들어가게 된다.

|  |  |
| --- | --- |
| 00 00 00 00 00 00 33 32 |  |
| 31 30 39 38 37 36 35 34 |  |
| 33 32 31 30 39 38 37 36 |  |
| 35 34 33 32 31 30 39 38 |  |
| 37 36 35 34 33 32 31 30 | (rsp) 위치 |

**C.** 원래의 getline의 리턴 주소가 buffer overflow로 인해 0x3332로 바뀌었기 때문에, 프로그램은 0x3332로 리턴하려고 시도할 것이다.

**D.** %rbx가 위의 스택 그림과 같이 0x3130393837363534로 바뀌게 될 것이다. STACK에 저장해 놓았던 값을 get\_line 리턴 전에 가져올 것이기 때문이다.

**E.** malloc 부분에서 strlen(buf)+1으로 써야 하고, NULL을 리턴하지는 않는지 확인할 필요가 있다. strlen 함수는 널 문자에 도달하기까지 버퍼의 길이를 반환하기 때문이다.

**2. Practice Problem 3.47 on Page 322**

**A. Approximate range with the addresses ranged from 0x7FFF7F01B680 to 0x7FFFFFA53B40**

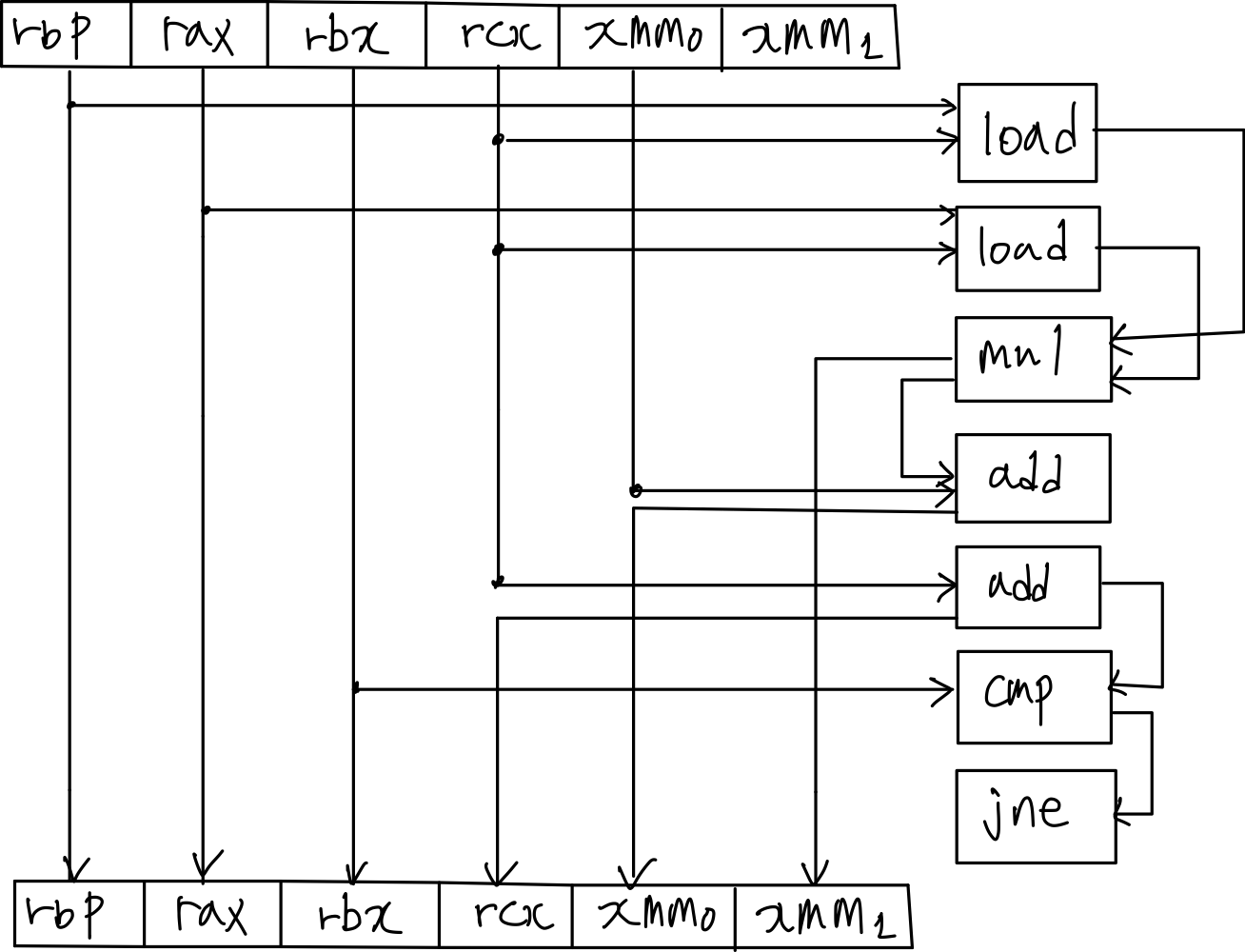
0x7FFFFFA53B40-0x7FFF7F01B680=0x80A384C0이고, 이를 10진수로 바꾸면 2158200000이다. 이는 대략 이므로, Approximate range: 로 생각할 수 있다.

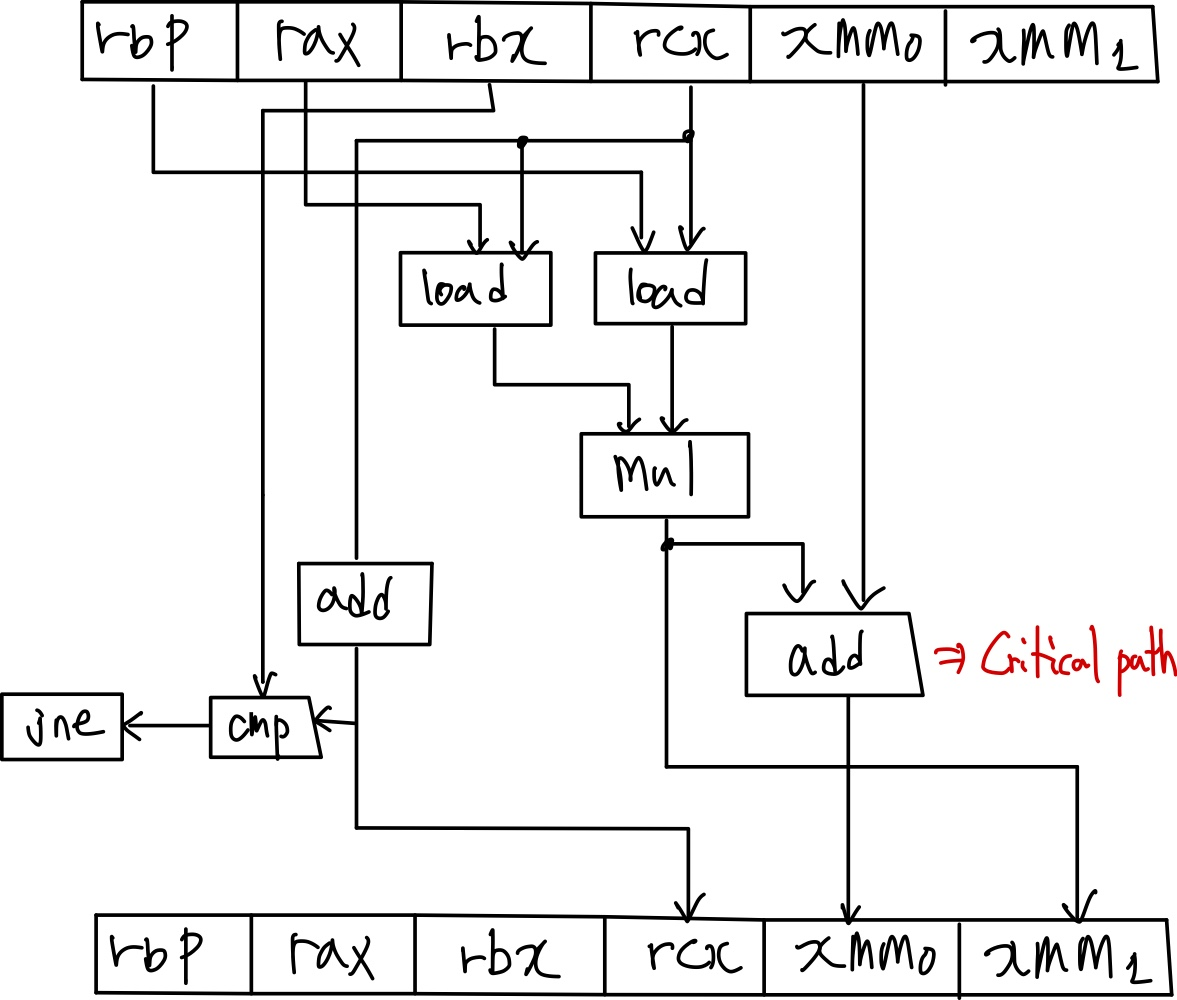
**B. How many attempts with 256-byte nop sled?**

256=byte nop sled이면, /==8388608 번의 attempt가 있어야 해당 범위에서 모든 starting address들을 test해볼 수 있다. 따라서 8388608번이 필요하다.

**3. Exercise 5.13 on page 606**

**A.**



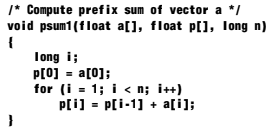


**B.** Critical path는 위 그림과 같이 vaddsd %xmm1, %xmm0, %xmm0에서 add하는 부분에서 나타난다. 따라서 double data type의 경우 lower bound of CPE: 3.00이다. (latency of floating point addition)

**C.** integer data의 경우 lower bound of CPE: 1.00이다.

**D.** multiplication operation은 5 clock cycles을 필요로 하지만, multi issuing과 register renaming을 통해 빠르게 계산될 수 있다. 따라서 critical path를 구성하지 않으므로 addition의 dependency를 가지는 critical path를 이루게 된다.

**4. Exercise 5.19 on page 609**



psum1의 원래 C code는 위와 같았고, 먼저 two-way unrolling을 해 보았다.

Two way는 p[i]=temp+a[i]과 p[i+1]=temp+a[i]+a[i+1]로 나타낼 수 있어서, 문제에서 나온 대로 3번의 addition이 들어갔다. (temp=p[0]=a[0]로 시작)

다음으로 four-way unrolling을 해 보았고 아래와 같이 나타낼 수 있었다.

k1=temp+a[i];

p[i]=k1;

p[i+1]=a[i+1]+k1;

k3=a[i+1]+a[i+2]+k1;

p[i+2]=k3;

p[i+3]=a[i+3]+k3;

그러면 총 5번의 addition이 필요한 것을 확인할 수 있었다. 그런데, CPE가 1.67이므로, 1.67=15/9=3.00\*(5/9)로 생각할 수 있었다. 따라서 이는 9번의 연산 중에서 5번이 critical path인 경우라고 볼 수 있었고, 이제 six-way unrolling으로 확장시켜보면 아래와 같은 코드를 만들 수 있었다.

void psum1(float a[], float p[], long n){

long I;

float k2, k4;

float temp= p[0] = a[0];

for(int i=1; i<n-6; i=i+6){

p[i]=temp+a[i];

k2=temp+(a[i]+a[i+1]);

p[i+1]=k2;

p[i+2]=a[i+2]+k2;

k4=a[i+2]+a[i+3]+k2;

p[i+3]=k4;

p[i+4]=a[i+4]+k4;

p[i+5]=a[i+4]+a[i+5]+k4;

temp=p[i+5];

}

for(;i<n;i++){

p[i]=temp+a[i];

}

}

따라서 addition의 개수를 세면 총 9개이고 그 중에서 critical path를 찾아야 한다. Critical path는 병렬적으로 수행이 불가해 data dependency를 가지는 addition일 것이므로, k2의 계산 부분, p[i+2] 계산, k4 계산, p[i+4] 계산, p[i+5] 계산 부분에서 각 1개씩 총 5개일 것으로 생각하였다. 결론적으로 floating point의 latency: 3.00이므로, (5/9)\*3.00=1.67의 CPE가 six-way unrolling에서 나타나는 것을 확인할 수 있다.

**5. Exercise 6.24 on page 685**

Rotational rate: 18000RPM, , Average number of sectors/track: 2000, Surfaces: 4, Sector size: 512 bytes이다. 이다. 그리고, 2mb이므로 2mb=byte로 생각하면 개의 512byte block이 있는 것으로 생각할 수 있다.

**A.** Best case의 경우 에서, 부분을 4096\*(3.33/2000)으로 계산할 수 있을 것이다. 나머지 값들은 4096이 곱해지지 않는다. 따라서 구한 값들로 optimal time을 구해보면, 8ms+0.5\*3.33ms+2\*3.33ms=16.48ms이 나온다.

**B.** Random case의 경우 에서, 부분에서, A번과 같이 계산핸 값에 개수 4096을 곱해서, (8+3.33/2+(3.33/2000))\*4096=39954ms가 나온다.