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’.

a)

|  |
| --- |
| 00 00 00 00 00 40 07 76 |
| 01 23 45 67 89 ab cd ef |
|  |
|  |
| (rsp) |

b)

|  |
| --- |
| 00 00 00 00 00 40 07 76 |
| 01 23 45 67 89 ab 00 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 |

c) 0x 40 07 06 (return address는 변하지 않는다)

d %rbx가 0x0123456789abcd3f로 세팅되었다.

e) result의 길이는 strlen(buf)+1이 되어야 하고, strcpy해야함. Strcpy는 널문자를 포함하여 string을 복사하지만, strlen은 널문자에 도달할때까지의 버퍼의 길이를 반환하는 함수이기 때문이다, 또한 리턴되는 값이 null이 아닌지를 확인해야한다.

2. Practice Problem 3.47 on Page 322

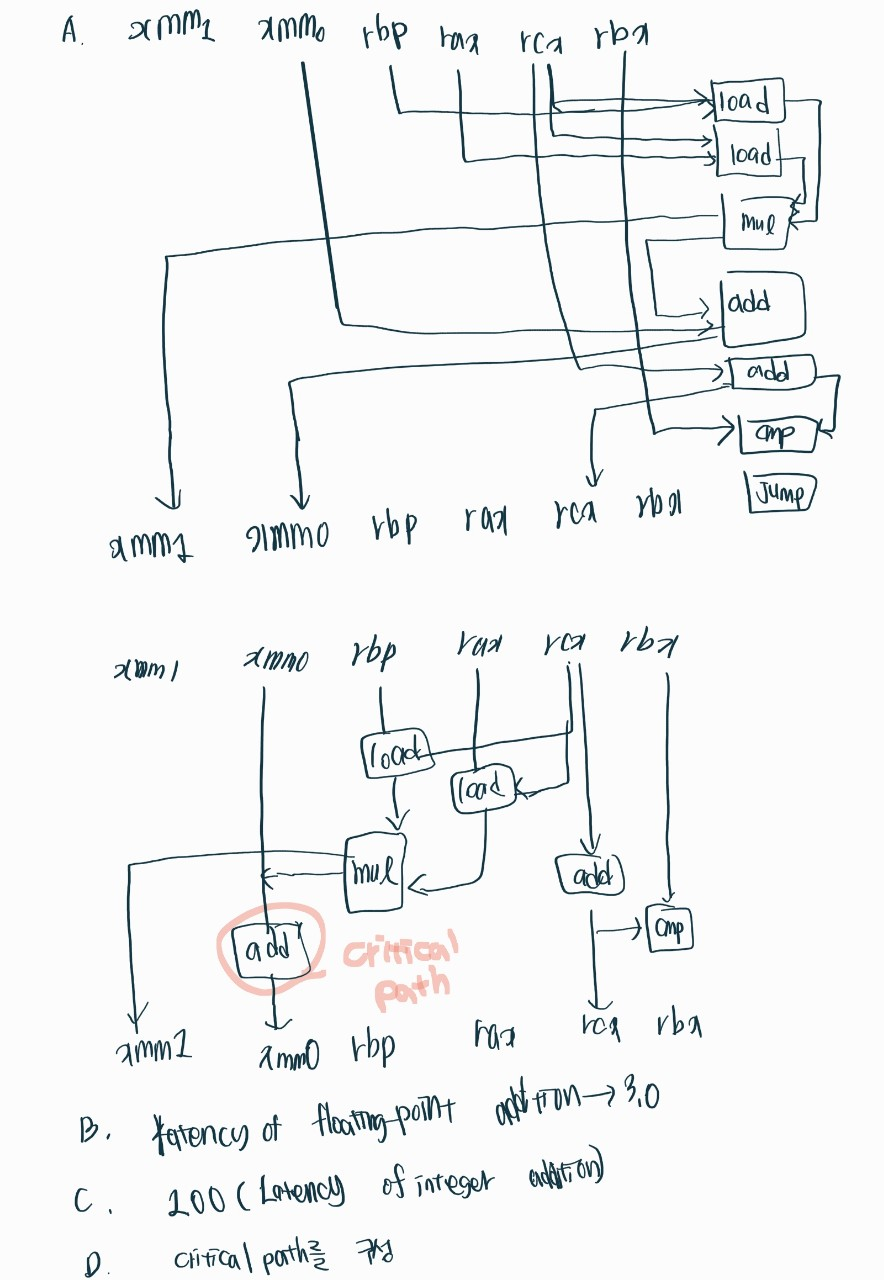
A. approximate range with the addresses ranged from 0x7fff7f01b680 to 0x7fffffa53b40

2,158,200,000=대략 2^31

B.How many attemps with 256-byte nop sled?

2^23=8,388,608 attempts needed

3. Exercise 5.13 on page 606



b. latency of floating-point addition:3

c. latency of integer addition:1.00

d. multiplication은 multi -issue나 register renaming을 통해 빠르게 이루어질 수 있고, 따라서 critical path를 구성하지 않기 때문이다. 덧셈이 data dependency를 갖는critical path를 이룬다.

4. Exercise 5.19 on page 609

최적화와 루프 풀기를 적용하면 2중풀기에는 3번 4중풀기에는 5번이 적용된다고 했다.

따라서 이중풀기와 4중풀기를 해보았다.

이중풀기는

p[i]=last\_val+a[i]

P[i+1]=last\_val+a[i]+a[i+1]

과 같이 나타낼 수 있고 따라서 덧셈연산이 4번쓰인다..

4중풀기를 해보면

K1=last\_val+a[i]

P[i]=k1;

P[i+1]=k1+a[i+1]

K3=k1+(a[i+1]+a[i+2])

P[i+2]=k3;

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

이렇게 총 5번의 덧셈 연산으로 구현할 수 있다. 우리의 machine의 CPE가 1.67이라하였으므로 1.67을 3\*5/9로 분해해보았다. 따라서 9번의 연산 중 5번이 critical path가 되는 경우라고 생각했다. 위의 4중루프 풀기 방식을 6중 루프 풀기 방식으로 확장해보면

Void psum1a(float a[],float p[],long n)

{

long i; float k2,k4;

float last\_val=p[0]=a[0];

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

{P[i]=last\_val+a[i];

k2=last\_val+(a[i]+a[i+1]);

P[i+1]=k2;

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

K4=k2+(a[i+2]+a[i+3])

P[i+3]=k4;

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

P[i+5]=k4+(a[i+4]+a[i+5]);

last\_val=p[i+5];

}

for(;i<n;i++)

{

P[i]=last\_val+a[i];

}

}

덧셈 연산이 loop안에서 9번 나오는 걸 볼 수 있다.

이중에 criticalpath를 구성하는 add의개수를 확인해보았다.

Critical path는 병렬적으로 수행될 수 없어 데이터 의존성을 가지는 덧셈 연산으로 생각할수있었다. 따라서 k2에서 한 개, p[i+2]에서 1개 k4에서 1개,p[i+4]에서 1개 p[i+5]에서 1개로 총 5개라고 생각했다. 부동 소수점 덧셈의 latency 가 3이므로 5/9를곱하면 1.67의 값이 나온다.

5. Exercise 6.24 on page 623

T rot=60/18000RPM\*1000ms=3.33ms

2mb파일은 2,000,000이므로 512byte로 나누면 대략 4000섹터에 있다.

최적의 경우에는 Tavg seek+Tavg rotation+ 2\*Tmax rotation이고,

랜덤의 경우에는 4000(Tavg seek +Tavg rotation+T transfer)

1. Best

8+3.33/2+3.33\*2=16.325ms

1. Randomly 4000\*(8+3.33/2+3.33/2000)=38,666ms

혹은 Q&A에 있는 조건대로 하면

2mb은 4096개의 512byte block이 있다. 이럴경우에는

A. 8+3.33/2+(3.33/2000)\*4096=16.48

B. randomly

4096\*(8+3.33/2+3.33/2000)=39,954ms