4.45

A. 不正确，当REG为%rsp时，这样会压入%rsp - 8而非%rsp

B. 对于 pushq REG:

1.movq REG, -8(%rsp)

2.subq $8, %rsp

4.49

这一题要求只能使用一个cmov指令实现书上的6-11行。我们首先分析一下4.48里面两条cmov冗余的地方：我们使用了两个cmovl ，其实这两个的条件码是一样的。于是我们可以使用一个cmovl 到一个寄存器A，然后在另一个寄存器B用算术运算将其线性关联起来（例如减法）。例如，如果我们要交换[A]，[B]，我们先将[A]存在%r1, -[A]存在%r2，cmovl [B], %r1，addq %r1, %r2，这时如果[A]大于[B]，%r1中为[B]，%r2中为[B]-[A]，否则%r1中为[A]，%r2中为0。随后，我们将%r1赋值给A，[B]-%r2赋值给B即可。

代码如下：

1. mrmovq (%rdx), %r8

2. rrmovq %rdx, %rsi

3. addq %r10, %rsi

4. mrmovq (%rsi), %rcx

5. rrmovq %rcx, %r10

6. subq %r8, %r10

7

8. cmovl %rcx, %r8

9. xorq %r12, %r12

10. mrmovq (%rdx), %r10

11. subq %r10, %r12

12. addq %r8, %r12

13. rmmovq %r8, (%rdx)

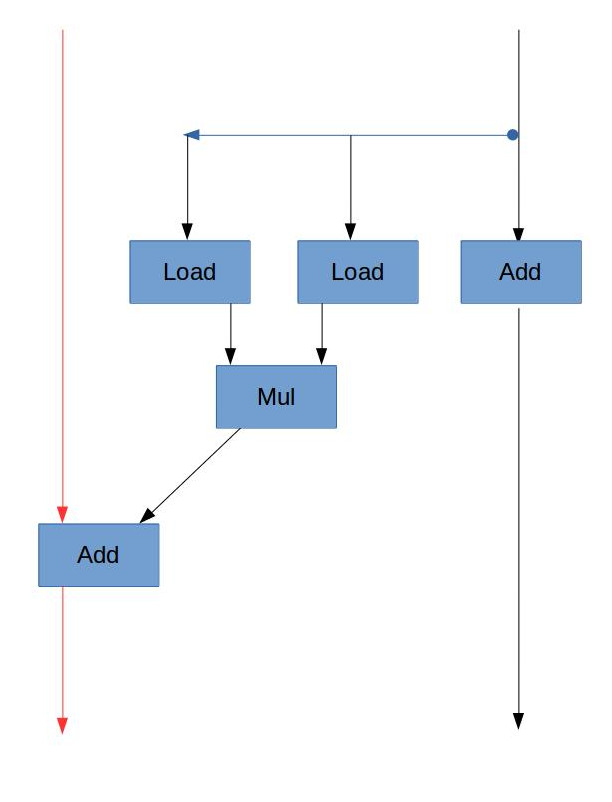
14. subq %r12, %rcx

15. rmmovq %rcx, (%rsi)

这里要注意一下，我这里为了方便使用了%r12寄存器，但是它是一个Callee saved的，所以我们要在bubblesort首尾分别加上pushq %r12和popq %r12 .

5.13

A.



B. 由浮点数加法的延迟，CPE的下界应该是3。

C. 由整数加法的延迟，CPE的下界应该是1.

D. 由A中的数据流图，虽然浮点数乘法需要5个周期，但是它没有“数据依赖”，也就是说，每次循环时的乘法不需要依赖上一次乘法的结果，可以各自独立进行。但是加法是依赖于上一次的结果的（sum = sum + 乘法结果），所以该循环的“关键路径”是加法这条链。而浮点数加法的延迟为3个周期，所以CPE为3.00。

5.17

1.#include <limits.h>

2.#define K sizeof(unsigned long)

3.void \*word\_memset(void \*s, int c, size\_t n)

4.{

5. if (n < K)

6. {

7. size\_t cnt = 0;

8. unsigned char \*schar = s;

9. while (cnt < n)

10. {

11. \*schar++ = (unsigned char)c;

12. cnt++;

13. }

14. }

15. else

16. {

17. unsigned long word = 0;

18. for (int i = 0; i < K; ++i)

19. {

20. word <<= K\*CHAR\_BIT;

21. word += (unsigned char)c;

22. }

23.

24. size\_t cnt = 0;

25. unsigned long \*slong = s;

26. while (cnt < n)

27. {

28. \*slong++ = word;

29. cnt += K;

30. }

31.

32. unsigned char \*schar = slong;

33. while (cnt < n)

34. {

35. \*schar++ = (unsigned char)c;

36. cnt++;

37. }

38. }

39. return s;

40.}

7.9

0000000000400526 <main>:

400526: 55 push %rbp

400527: 48 89 e5 mov %rsp,%rbp

40052a: 48 83 ec 10 sub $0x10,%rsp

40052e: 89 7d fc mov %edi,-0x4(%rbp)

400531: 48 89 75 f0 mov %rsi,-0x10(%rbp)

400535: e8 07 00 00 00 callq 400541 <p2>

40053a: b8 00 00 00 00 mov $0x0,%eax

40053f: c9 leaveq

400540: c3 retq

0000000000400541 <p2>:

400541: 55 push %rbp

400542: 48 89 e5 mov %rsp,%rbp

400545: be 26 05 40 00 mov $0x400526,%esi

40054a: bf e4 05 40 00 mov $0x4005e4,%edi

40054f: b8 00 00 00 00 mov $0x0,%eax

400554: e8 a7 fe ff ff callq 400400 <printf@plt>

400559: 90 nop

40055a: 5d pop %rbp

40055b: c3 retq

40055c: 0f 1f 40 00 nopl 0x0(%rax)

我们要关注printf的第二个参数，即%esi ，可以看到mov $0x400526,%esi ，其中0x400526就是main函数的地址 ，所以printf会默认输出开头的一个字节（char类型），即push的机器码55.

所以原因在于bar6.c中的main是一个weak类型链接的变量，而foo6.c中的main是一个strong类型的，所以再链接的时候bar6.o中的main会解析到foo6.o中的main，从而一直输入0x55.

这里要特别说明一点，在C中，函数名和数组名一样都是“二等公民”，是一个内存块的标识符，在进行算术运算的时候会“退化”为一个指针常量。这里链接的时候就没有发生退化（链接器不管编译器的事情）。如果我们将printf("0x%x\n", main);这句话放到foo6.o中则会输出400526。

7.13

A.

ar -t /usr/lib/x86\_64-linux-gnu/libc.a |wc -l1579

ar -t /ysr/lib/x86\_64-linux-gnu/libm.a | wc -1471

B.

不一样

c.

ldd /sr/bin/gcc