**5.13：**

A.

画图：

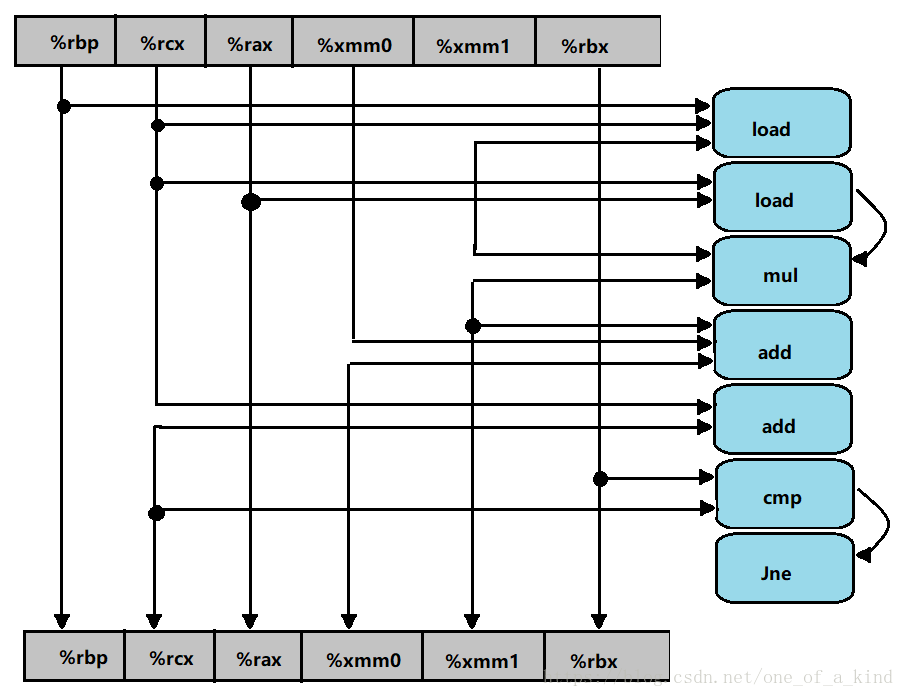


图1 机器级代码的初步图形化表示图

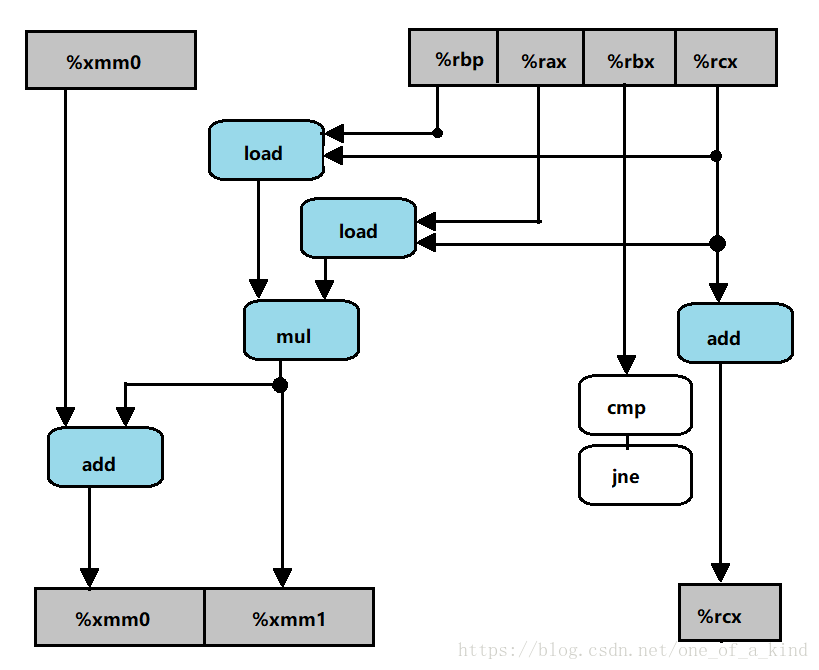


图2 重新排列操作符后表示数据相关

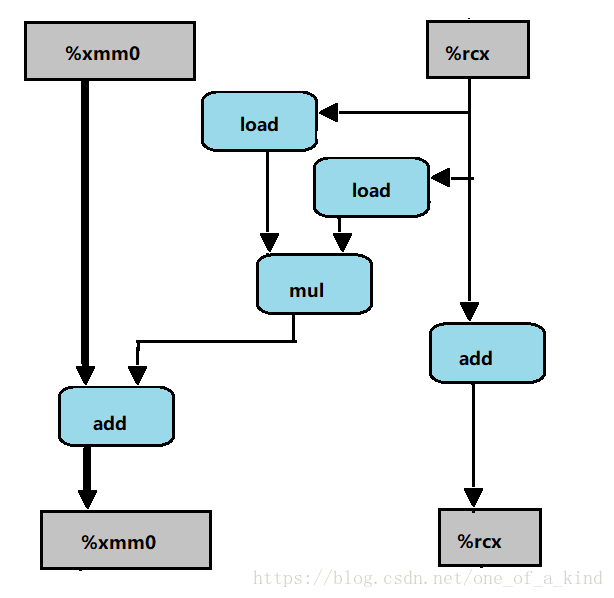


图3 关键路径示意图

关键路径图3加粗部分所示

B.

下界为浮点加法的延迟界限，CPE 为 3.00

C.

下界整数加法的延迟界限，CPE 为 1.00

D.

关键路径上只有浮点加法，所以CPE 为 3.00

**5.17：**

void \*new\_memset(void \*s, int c, size\_t n) {

unsigned long w; //题目给定的数据类型

unsigned char \*pw = (unsigned char \*)&w;//转为字节类型地址给pw

size\_t cnt = 0;

while (cnt < K) {//K为sizeof(unsigned long)

\*pw++ = (unsigned char)c;//用c进行赋值

cnt++;

}//现在w每个字节都是c

size\_t i;

unsigned char \*schar = s;//用schar指针按字节访问s

//防止刚开始K字节没有对齐，先按单字节处理

//直至对齐后开始K字节大批量赋值

for (i = 0; (size\_t)schar % K != 0 || i == n; i++) {

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

}

//i接上面继续一次K字节，防止最后一次不足K字节，设置limit

size\_t limit = n - K + 1;

for (; i < limit && (int)limit > 0; i += K) {

//对schar处的内容按用w赋值，一次多字节更快

\*(unsigned long \*)schar = w;

schar += K;

}

//由于最后可能不足K字节，i接上面继续一次一字节，收尾工作直到处理完毕

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

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

}

//对s操作完毕

return s;

}

**7.7：**

/\* bar5.c \*/

//使用static声明x，使其链接为内部链接：

static double x;

void f()

{

x = -0.0;

}

**7.8：**

A.//正常

main.1

main.2

B.//未知，int与double的x未赋初值

unknown

unknown

C.//错误，int与double的x都赋了初值造成冲突

error

error

**7.10：**

A.// 链接器看到.o文件会直接更新E，U，D，后面即使有依赖也不用包含

gcc p.o libx.a

B.

gcc p.o libx.a liby.a libx.a

C.

gcc p.o libx.a liby.a libx.a libz.a

**7.12：**

A.// 由r.type = R\_X86\_64\_PC32知为PC相对寻址。

ADDR(s) = ADDR(.text) = 0x4004e0

ADDR(r.symbol) = ADDR(swap) = 0x4004f8

refaddr = ADDR(s) + r.offset = 0x4004ea//运行时地址

\*refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr) = 0xa

//更新该引用，使得它在运行时指向swap程序

B.

ADDR(s) = ADDR(.text) = 0x4004d0

ADDR(r.symbol) = ADDR(swap) = 0x400500

refaddr = ADDR(s) + r.offset = 0x4004da

\*refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr) = 0x22