5.13：

A.

画图：

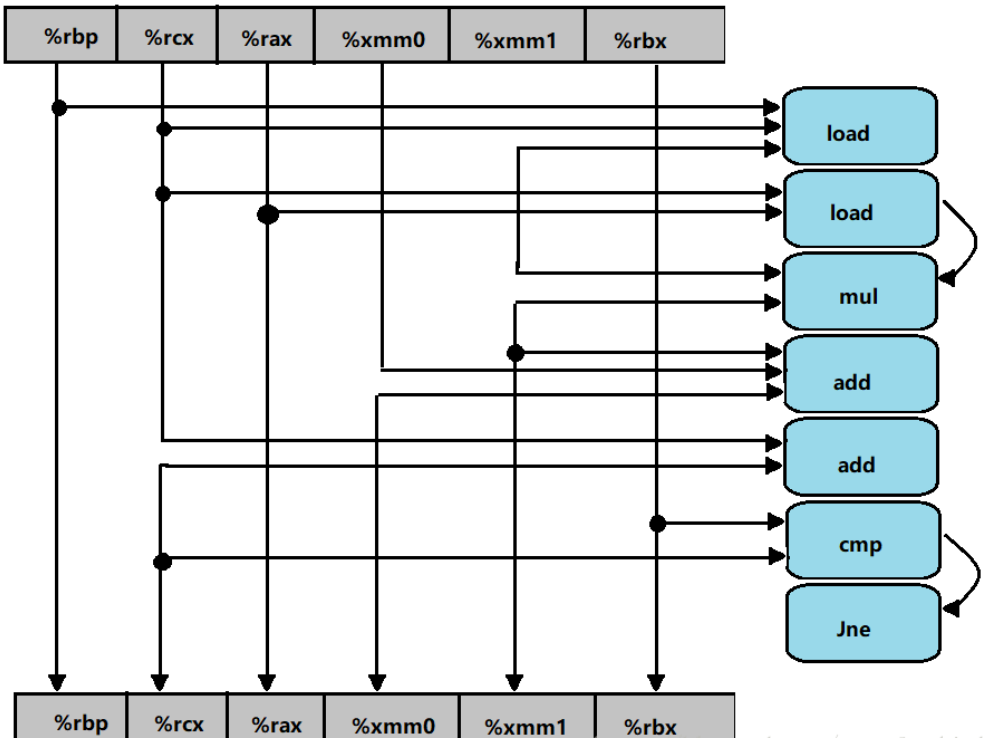

图 1 机器级代码的初步图形化表示图

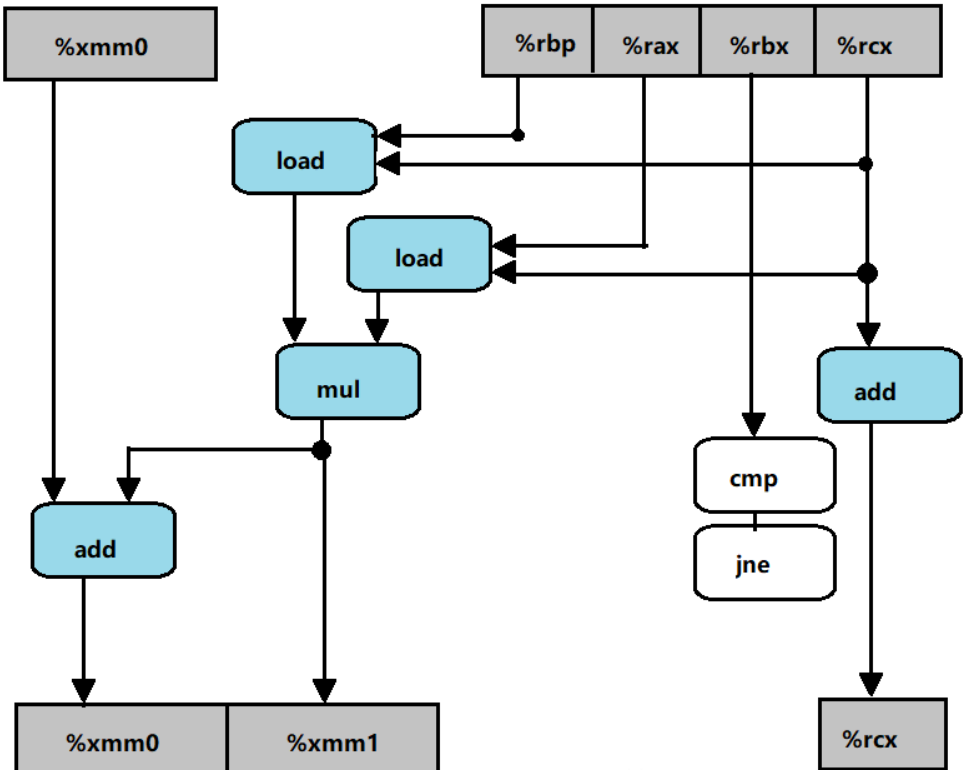

图 2 重新排列操作符后表示数据相关

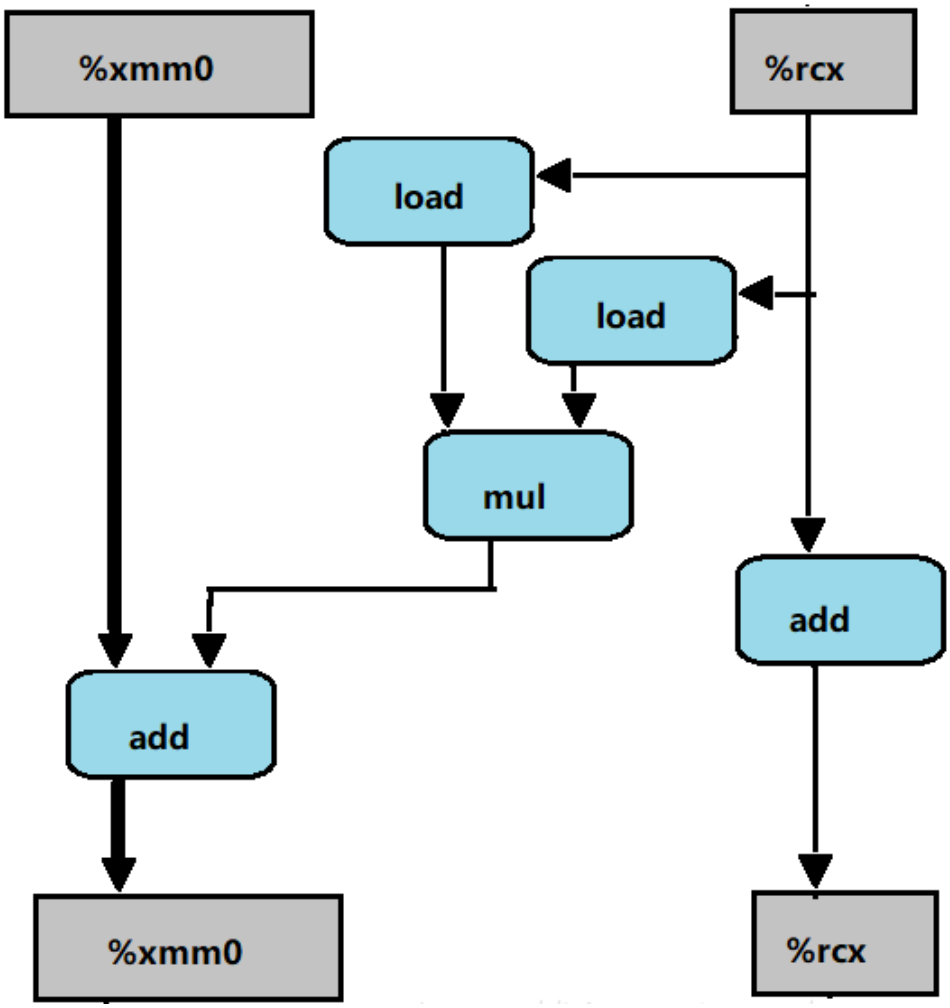

图 3 关键路径示意图

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