Lab6 Learn from the past

实验要求

- 将前几次的实验的汇编语言或机械码代码用高级语言（如c\c++）复现
- 要求算法一致
- 程序列表：

> lab0l (lab1 L version)
> lab0p (lab1 P version)
> fib (lab2 fibonacci)
> fib-opt (lab3 fibonacci)
> rec (lab4 task1 rec)
> mod (lab4 task2 mod)
> prime (lab5 prime)

实验过程

Lab0l

算法与之前的相同，即采用把乘法拆分为加法来实现两个short变量的乘法运算。
代码入下：

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    short a,b,c=0;
    scanf("%hd%hd",&a,&b);//input a,b
    do{
        c=c+a;
        b=b-1;
    }while(b!=0);//计算a*b
    printf("%hd",c);//output c=a*b
}
```

Lab0p

算法实现保持一致，把乘法分为移位和加法,把16位的每一位对应的权值按规则相加。
代码入下：

```
#include<stdlib.h>
#include<stdio.h>
int main(){
    short a,b,c,d,e;
    scanf("%hd%hd",&a,&b);//input a,b
    //以下为取a（总共16位）的每一位，若为1则加上对应的权值
    c=1;
    e=0;
    d=c&a;
    if(d!=0){
        e=e+b;
```

```
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
```

```
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    b=b+b;
    c=c+c;
    d=c&a;
    if(d!=0){
        e=e+b;
    }
    printf("%hd",e);//output e=a*b
}
```

Fib

算法即按照fib数列的定义，依次从前往后递推出每一个值，直到要求的F(N)。
代码入下：

```
#include<stdlib.h>
#include<stdio.h>
int main(){
    short r0;
    scanf("%hd",&r0);//input r0
    short r1=0,r2=0,r3=1;
    short r4=1023;
    short r7=0,r5=-3;
    do{
    r5=r5+1;
    if(r5>0){
        r1=r1+r1;
    }
    r7=r1+r3;
    r7=r7&r4;//mod 1024
    r1=r2;
```

```
    r2=r3;
    r3=r7;
    r0=r0-1;
    }while(r0>=0);
    printf("%hd",r7);//output r7=F(r0)
}
```

Fib-opt

算法：由于数列从20到147是循环的，所以可以把所有情况都归结到1-147，这样可以减少递推的次数。首先判断r0是否属于小于20，若小于20则不用改变r0的值，若大于20，则把r0的值照着循环归结到20-147即可，然后再递推的计算F（R0）。

代码入下：

```
#include<stdlib.h>
#include<stdio.h>
int main(){
    short r1=0,r2=0,r3=0;
    short r4=0,r6=0;
    short r7=0,r5=0;
    short r0;
    scanf("%hd",&r0);//input r0
    //把r0归结到1-148
    r0=r0-20;
    if(r0>=0){
        do{
        r6=-128;
        r0=r0+r6;
        }while(r0>=0);
        r7=128;
        r0=r0+r7;
    }
    //按照Fib中的算法递推一遍
    r0=r0+20;
    r3=1;
    r5=-3;
    r4=1023;
    do{
    r5=r5+1;
    if(r5>0){
        r1=r1+r1;
    }
    r7=r1+r3;
    r7=r7&r4;
    r1=r2;
    r2=r3;
    r3=r7;
    r0=r0-1;
    }while(r0>=0);
    printf("%hd",r7);//output r7=F(r0)
}
```

Rec

算法一致，但在原先的机械码代码中额外有R2与R7分别为返回地址栈的指针和储存返回的地址，但在c++中不需要这一步，函数会在调用时自动创建一个栈，并在结束时自动弹出栈返回函数。
代码入下：

```
#include<stdlib.h>
#include<stdio.h>
void A(short &r0,short &r1,short &t);
int main(){
    short t=5,r1;//t为内存某处存的值，初始为5
    short r0=0;
    A(r0,r1,t);
    printf("%hd\n",r0);//output r0=5
    printf("%hd\n",r1);//output r1=0
    return 0;
}
void A(short &r0,short &r1,short &t){
    r0=r0+1;
    r1=t;
    r1=r1-1;
    t=r1;
    if(r1!=0){
        A(r0,r1,t);//递归调用
    }
    return;
}
```

Mod

算法:
$x=(7+1)*n+m;$
$xmod7=(n+m)mod7;$
代码入下：

```
#include<stdlib.h>
#include<stdio.h>
void A(short &r4,short &r1,short &r2,short &r3);
int main(){
    short r1,r2,r4,r3,r0;
    r1=288;//r1初始值为288
    //scanf("%hd",&r1);
    do{
    A(r4,r1,r2,r3);
    r2=r1&0x7;
    r1=r4+r2;
    r0=r1-7;
    }while(r0>0);//反复调用A直至r1<=7
    r0=r1-7;
    if(r0>=0){//if r1=7,则余0
        r1=r1-7;
    }
    printf("%hd\n",r1);//返回r1 mod 7 的值到 r1
    return 0;
}
void A(short &r4,short &r1,short &r2,short &r3){
```

```
    short r5;
    r2=1;//权值
    r3=8;//用于取r1的各个位
    r4=0;//存储算法中n
    do{
        r5=r1&r3;
        if(r5!=0){
            r4=r4+r2;
        }
        r2=r2+r2;
        r3=r3+r3;
    }while(r3!=0);
    return;
}
```

Prime

把r0对小于根号r0的值都取一遍余，由此来判断r0是否是质数。
代码入下：

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    short r0,r1,r2,r3,r4,r5,r6,r7;
    scanf("%hd",&r0);//input r0
    r5=-r0;
    r2=2;
    r3=0;
    r1=1;
    while(1){
    r3=r2;
    r4=0;
    do{r4=r4+r2;
    r3=r3-1;
    }while(r3!=0);
    r4=r4+r5;
    if(r4<=0) {
    r4=r0;
    r3=-r2;
    do{
        r4=r4+r3;
    }while(r4>0);
    if(r4<0){
        r2=r2+1;
    }
    else{
        r1=0;
        break;}
    }
    else break;
    }
    printf("%hd",r1);//output r1
    return 0;
}
```

比较与思考

评估高级语言程序的性能

- 首先可以估算算法的时间与空间复杂度，代表这程序执行所需要的时间和最大所要占据的空间，时间越短，说明程序执行越快，空间复杂度越小，说明程序所需要的空间越小，总体性能越好。
- 程序的鲁棒性也是程序的重要体现，对于各式各样的输入程序都能做出相应的回应，输入错误也有对应的错误提示输出，不至于程序崩溃会随机输出一些垃圾值。
- 程序要与用户有良好的交互，如在输入时有输入提示
- 程序的负荷也是一个重要指标，即程序运行时最大可以负载的量（如读文件时，可以读入最大文件的大小）。

高级语言比LC汇编语言3跟容易编写的原因

- 高级语言有许多现成的东西，比如减法，取模等可以直接拿来用，而LC3汇编语言只能从最基础的命令写起。
- 高级语言更加直观，跟接近人的逻辑思维，如赋值时，高级语言只需“=”，而LC3汇编语言需要通过add与.fill指令来实现。
- 高级语言的编写过程更加灵活，形式更加多变，允许编写者可以更灵活使用，而LC3格式死板，要求高。

LC3指令集中添加什么指令可以大大简化程序

- 首先可能是乘法，减法等语句，因为这些语句比较常用，如果每次都需要多步来操作的话，容易出错，且程序会比较长。
- 然后对栈的操作，可以加入pop与push操作（专门有个空间来储存），因为在LC3中调用子程序时，难免会调用栈来储存数据，这样就可以在不破坏数据的情况下入栈与出栈，大大简化了程序。

高级语言需要向LC3学习的

- 语言到最后都要转化为机械可以理解的电平信号，高级语言翻译成机械码的速度更慢，所以在写高级语言的时候，可以用一些跟接近基础的指令，加快编译速度（如printf比cin更快）。
- LC3汇编语言更加底层，直接对口机械语言，能到机械做到精确控制（如使用特定的内存与寄存器），而高级语言能做到最后执行的结果相同，中间的过程可能每次都不同，这是需要学习的。

实验总结

- 这次实验完成前几次实验的机械码或汇编语言向高级语言的转变，让我更加深刻的理解的高级语言和汇编语言之间的关系与区别。