

实验报告——lab6 Learn from the past

PB20151793 宋玮

part1: lab01 (lab1 L version)

该实验是用尽量短的程序完成 r0 和 r1 的乘法，结果存储在 r7 中。

算法思想：

首先判断 R0 正负：如果是正数（或 0），则每次对 R7 进行加 R1 操作后，R0 减 1，重复操作，直至 R0 减为 0。由于该过程对 R1 加了 R0 次。因此 R7 中的结果为 $R0 \times R1$ 。如果是负数，则每次对 R7 进行加 R1 操作后，R0 加 1，重复操作，直至 R0 变为 0。最后对 R7 中的结果取反加一，可得到 $R0 \times R1$ 。

用 c 语言撰写的版本如下，每一条语句基本遵循对应的 lc-3 汇编指令。

其中，变量名与 lc-3 中寄存器名相同。

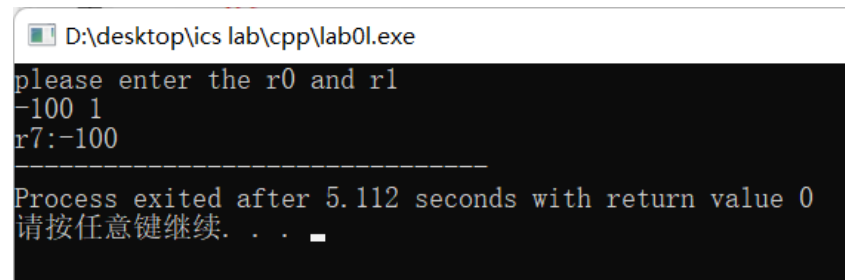
不同的是，BR 指令用循环（while 语句）代替。

```
1  #include<stdio.h>
2  int main(){
3      short int r0,r1,r7=0,r2=0;
4      printf("please enter the r0 and r1\n");
5      scanf("%hd%hd",&r0,&r1);
6      r2=r0;
7      if(r2<0){
8          while(r2<0){
9              r7=r7+r1;
10             r2=r2+1;
11         }
12         r7=-r7;
13         printf("r7:%hd",r7);
14         return 0;
15     }
16     else if(r2==0){
17         printf("r7:%hd",r7);
18         return 0;
19     }
20     else{
21         while(r2>0){
22             r7=r7+r1;
23             r2=r2-1;
24         }
25         printf("r7:%hd",r7);
26         return 0;
27     }
28 }
```

运行结果：

需要输入你想测试的 r0 和 r1 的值，
之后程序将输出 r7 的值，也即乘法结果。

如下图： -100*1



```
D:\desktop\ics lab\cpp\lab01.exe
please enter the r0 and r1
-100 1
r7:-100
-----
Process exited after 5.112 seconds with return value 0
请按任意键继续. . .
```

part2: lab0p (lab1 P version)

该实验是用尽量少的指令数完成 r0 和 r1 的乘法，结果存储在 r7 中。

算法思想：

由于 R0,R1 均为 01 串。因此在进行列竖式乘法计算时，每一行结果要么是 R0，要么是 0，取决于 R1 对应位上是 1 还是 0。再通过对 R0 的移位后相加，则可以得到结果。目前，仅能实现的移位方式是左移，即通过 $R0=R0+R0$ ，可以实现对 R1 的左移一位。因此，本算法也采取自加左移方式。而判断 R1 对应位上是 0 还是 1，也需要对 R1 进行左移，然后通过判断正负，判断该位上是 0 还是 1。

注：当 R1 为负数时，采取 R1 和 R0 同时取反加一的操作，即同时取相反数，可以缩减指令执行条数。

用 c 语言撰写的版本如下，变量名与 lc-3 中寄存器名相同。

```
1  #include<stdio.h>
2  int main(){
3      short int r0,r1,r7=0,r2=0,r3=0,r4=0;
4      printf("please enter the r0 and r1\n");
5      scanf("%hd%hd",&r0,&r1);
6      r3=15;
7      if(r1<0){
8          r1=-r1;
9          r0=-r0;
10     }
11     r3=r3-1;
12     while(r3>0){
13         r1=r1+r1;
14         if(r1<0){
15             r2=r0;
16             r4=r3;
17             while(r4){
18                 r2=r2+r2;
19                 r4=r4-1;
20             }
21             r7=r7+r2;
22         }
23         r3=r3-1;
24     }
25     r1=r1+r1;
26     if(r1>=0){
27         printf("r7:%hd",r7);
28         return 0;
29     }
30     else{
31         r7=r7+r0;
32         printf("r7:%hd",r7);
33         return 0;
34     }
35 }
```

运行结果：

需要输入你想测试的 r0 和 r1 的值，
之后程序将输出 r7 的值，也即乘法结果。
如下图：999*10



```
D:\desktop\ics lab\cpp\lab0p.exe
please enter the r0 and r1
999 10
r7:9990

Process exited after 4.218 seconds with return value 0
请按任意键继续. . .
```

part3: fib (lab2 fibonacci)

该实验是求 $f(n)$ ，其中 n 存放在 $r0$ 中。

算法思想：

$R2$ 存放 $f(k-3)$, $R3$ 存放 $f(k-2)$, $R4$ 存放 $f(k-1)$. 首先对于特殊情况, $n=1$, $n=2$, 做特殊情况判断, 直接处理得到对于结果。对于一般数据, 每一次循环, 进行 $R7=R4+2*R2$ 操作, 并且对 $R7$ 进行模 1024 操作。接着用此时的 $R3$ 替换 $R2$, $R4$ 替换 $R3$, $R7$ 替换 $R4$, 得到新的 $f(k-3)$, $f(k-2)$, $f(k-1)$ 。并且对计数减一。重复循环, 直至计数为 0, 跳转至结束 (halt)。 $R7$ 中存放的即为 $f(n)$ 。

用 c 语言撰写的版本如下, 变量名与 lc-3 中寄存器名相同。

需要初始输入 $r0$, 程序输出为 $r7$, 即 $f(n)$ 。

```
1  #include<stdio.h>
2  int main(){
3      short int r0,r2=0,r3=0,r4=0,r7=0;
4      printf("please enter the r0(n)\n");
5      scanf("%hd",&r0);
6      r2=1;
7      r3=1;
8      r4=2;
9
10     r0=r0-2;
11     if(r0<0){
12         r7=1;
13         printf("r7(fn):%hd",r7);
14         return 0;
15     }
16     else if(r0==0){
17         r7=2;
18         printf("r7(fn):%hd",r7);
19         return 0;
20     }
21     else{
22         while(r0>0){
23             r2=r2+r2;
24             r7=r2+r4;
25             r7=r7%1024;
26             r2=r3;
27             r3=r4;
28             r4=r7;
29             r0=r0-1;
30         }
31         printf("r7(fn):%hd",r7);
32         return 0;
33     }
34 }
```

运行结果：输入 $n=20$

```
D:\desktop\ics lab\cpp\fib.exe
please enter the r0(n)
20
r7(fn):930
-----
Process exited after 2.586 seconds with return value 0
请按任意键继续. . .
```

part4: fib-opt (lab3 fibonacci)

算法思想：

主要是在 lab2 fibonacci 做了一些改进，并且发现 $f(n)$ 存在周期，因此可以利用这个性质，对于 $n \geq 20$ 的数，先对其进行减 20，mode 128，再加 20 的操作。对于 $n < 20$ 的数，先不做操作，再利用改进的 lab2 求解。

用 c 语言撰写的版本如下，变量名与 lc-3 中寄存器名相同。

要注意的一点是：由于在 c 语言中 % (取模运算)，对负数取模的值可能还是负数，与 lc-3 汇编程序中 and 操作不同。因此在最后一个部分，我加上一个 if 语句。

```
27 |         if(r7<0) r7=r7+1024;
```

需要初始输入 r0，程序输出为 r7，即 $f(n)$ 。

```
1  #include<stdio.h>
2  int main(){
3      short int r0,r1=0,r2=0,r3=0,r6=0,r7=0;
4      printf("please enter the r0(n)\n");
5      scanf("%hd",&r0);
6
7      r1=1;
8      r2=1;
9      r3=2;
10     r6=r0-20;
11     if(r6>=0){
12         r0=r0-20;
13         r0=r0%128;
14         r0=r0+20;
15     }
16     r7=r7+r0;
17     r0=r0-2;
18     while(r0>0){
19         r7=r3+r1;
20         r7=r7+r1;
21         r1=r2;
22         r2=r3;
23         r3=r7;
24         r0=r0-1;
25     }
26     r7=r7%1024;
27     if(r7<0) r7=r7+1024;
28     printf("r7(fn):%hd",r7);
29
30     return 0;
31 }
```

运行结果：输入 n=93

```
D:\desktop\ics lab\cpp\fib-opt.exe
please enter the r0(n)
93
r7(fn):6
-----
Process exited after 2.177 seconds with return value 0
请按任意键继续. . .
```

part5: rec (lab4 task1 rec)

这个实验是一个填空实验。整体实现思想与 c 语言有一些不同。

首先，lc-3 汇编程序中，不管是实现主程序，还是实现子程序，寄存器的值都在时刻改变，因此在利用 c 语言撰写时，我们应将寄存器变量设置为全局变量。

```
2 short int r0,r1=0,r2=0,r7=0,memx3019;  
3 short int mem[10]={0};
```

其次，lc-3 汇编程序中，JSR 指令跳到子程序前，r7 会存储 JSR 下一条指令地址，方便返回时，直接 return r7，即返回该指令地址。而在 c 语言实现时，不涉及到指令地址，因此我们需要模仿 lc-3 汇编程序，设置该“虚拟地址”。

```
36 r2=0x300f;  
37 r7=0x3004;
```

再者，整个程序中，只涉及到两条 JSR 指令，则只涉及两个子程序。因此我们可以根据判断当前 r7 的值，然后再选择调用哪个子程序。

最后还有一点，由于在整个程序中，我们频繁用到 $r7 = \text{mem}(r2)$ ， $\text{mem}(r2) = r7$ ，因此我用了一个数组 `mem[]` 来实现该操作。并且利用 `memx3019` 变量来表示 lc-3 汇编程序中的 `mem(x3019)`。

```
6 r7=mem[r2-0x300f];
```

用 c 语言撰写的版本如下：

子程序 1:

```
4 void sub(){  
5     r2=r2-1;  
6     r7=mem[r2-0x300f];  
7     if(r7==0x300c){  
8         sub();  
9         return;  
10    }  
11    return;  
12 }
```

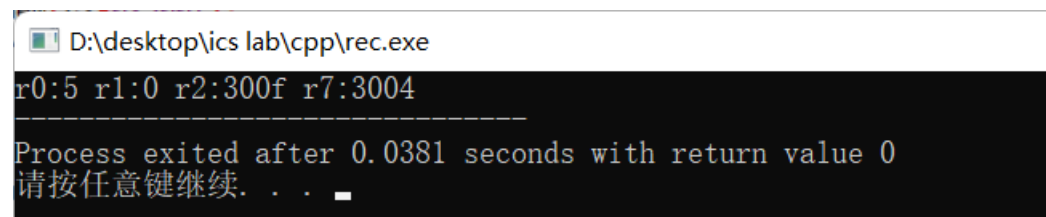
子程序 2:

```
13 void subroutines(){  
14     mem[r2-0x300f]=r7;  
15     r2=r2+1;  
16     r0=r0+1;  
17     r1=memx3019;  
18     r1=r1-1;  
19     memx3019=r1;  
20     if(r1==0){  
21         r2=r2-1;  
22         r7=mem[r2-0x300f];  
23         if(r7==0x300c){  
24             sub();  
25             return;  
26         }  
27         return;  
28     }  
29     r7=0x300c;  
30     subroutines();  
31     return;  
32 }
```

主程序:

```
33 int main(){
34     memx3019=5;
35     r0=0;
36     r2=0x300f;
37     r7=0x3004;
38     subroutines();
39     printf("r0:%hx r1:%hx r2:%hx r7:%hx",r0,r1,r2,r7);
40     return 0;
41 }
```

运行结果: 输出为主要的四个变量 r0, r1, r2, r7



```
D:\desktop\ics lab\cpp\rec.exe
r0:5 r1:0 r2:300f r7:3004
-----
Process exited after 0.0381 seconds with return value 0
请按任意键继续. . .
```

part6: mod (lab4 task2 mod)

算法思想:

用 c 语言撰写的版本, 每一条语句基本遵循对应的 lc-3 汇编指令。

即子程序实现 $r1/8$;

主程序求 $r1 \text{ mode } 7$ 的值。

与 part5 同样的道理, 由于寄存器的值时刻在变, 我们需要在 c 语言程序中, 把变量都定义为全局变量。

并且在 and 7 的操作部分, 我直接用了 %8, 在操作数是正整数的情况下, 两种是等效的。最后输出的 r1, 为初始 $r1 \text{ mode } 7$ 的值。

```

1  #include<stdio.h>
2  short int r0,r1=0,r2=0,r3=0,r4=0,r5=0;
3  void subroutines(){
4      r2=0;
5      r3=0;
6      r4=0;
7      r2=r2+1;
8      r3=r3+8;
9      while(1){
10         r5=r3&(r1);
11         if(r5) r4=r2+r4;
12         r2=r2+r2;
13         r3=r3+r3;
14         if(r3==0) break;
15     }
16     return;
17 }
18 int main(){
19     r1=0x120;
20     do{
21         subroutines();
22         r2=r1%8;
23         r1=r2+r4;
24         r0=r1-7;
25     }
26     while(r0>0);
27     r0=r1-7;
28     if(r0>=0) r1=r1-7;
29     printf("r1:%hd",r1);
30     return 0;
31 }

```

运行结果：

```

D:\desktop\ics lab\cpp\mod.exe
r1:1
-----
Process exited after 0.03593 seconds with return value 0
请按任意键继续. . .

```

part7: prime (lab5 prime)

算法思想：

与上述两个 part 相同，由于寄存器的值时刻在变，我们需要在编写 c 语言程序中，注意到这一点。与前面两个 part 不同的是，由于该程序只有一个子程序，我们完全可以利用引用参数（&）的做法，来完成寄存器的值也随子程序变化。

其他部分基本相同。完全按照 lc-3 汇编程序的思路编写。

子程序：

```
2 void subroutines(short int &r0, short int &r1, short int &r2, short int &r3, short int &r4, short int &r5, short int &r6){
3     while(1){
4         r4=r2-1;
5         r5=r2;
6         while(r4>0){
7             r5=r5+r2;
8             r4=r4-1;
9         }
10        r5=-r5;
11        r5=r0+r5;
12        if(r5<0) return;
13        r3=r0;
14        r6=r2;
15        r6=-r6;
16        while(r3>0){
17            r3=r3+r6;
18            if(r3==0){
19                r1=0;
20                return;
21            }
22        }
23        r2=r2+1;
24    }
25    return;
26 }
```

主程序：

```
28 int main(){
29     short int r0,r1=0,r2=0,r3=0,r4=0,r5=0,r6=0,r7=0;
30     printf("please enter the r0\n");
31     scanf("%hd",&r0);
32
33     r1=1;
34     r2=2;
35     subroutines(r0,r1,r2,r3,r4,r5,r6);
36     printf("r1:%hd\n",r1);
37     return 0;
38 }
```

运行结果：输入为 r0，输出为 r1

```
D:\desktop\ics lab\cpp\prime.exe
please enter the r0
333
r1:0

-----
Process exited after 4.088 seconds with return value 0
请按任意键继续. . .
```


总结：

通过用 c 语言编写之前的 7 个 lc3 程序，我有了如下一些体会。

高级语言程序（c 语言）在编写上更加简单明了，并且操作更加灵活。

我觉得原因在于，高级语言给我们提供了更多的框架，如 if 判断语句，while 循环语句，这更加符合我们的逻辑思维。而不用像在 lc-3 汇编程序中思考 BR 指令应该怎么判断什么，要跳转至哪里，以及 BR 指令要放到哪个位置。

并且 lc-3 汇编指令有限，而 c 语言定义的操作丰富且灵活。比如 lc-3 汇编指令甚至都没有乘法指令。当你需要用到乘法时，c 语言一条语句可以搞定，而 lc-3 汇编程序需要很多的逻辑组合，指令组合才能实现这样一个简单的功能。

我认为乘法指令和除法指令需要被添加至 lc3 中，毕竟乘法和除法都是基础的运算，如果添加进去，将使得程序实现更加简单。

但是，我们也可以从 lc3 中学习到的这一点是，返回（return）问题。在 lc3 中，返回时，是 return r7，可以顺利跳转至 r7 所在的指令位置。这样的逻辑非常清楚。而在 c 语言中，在调用多个函数以后，甚至在多个函数互相调用以后，这个逻辑将变得十分复杂。甚至不知道 return 将跳转至哪个地方。我觉得，这是 lc3 中比较出彩的一点。