Lab06 Learn from the past

[实验目的]

> 使用一门高级语言，实现*lab1~lab5*的程序，不改变程序的算法

[实验过程]

lab1_L

> 思路:用加法实现乘法

原程序:

```
0011 0000 0000 0000 ; start the program at location x3000
0101 000 000 1 00000 ; set R0 to 0
0101 001 001 1 00000 ; set R1 to 0
0101 111 111 1 00000 ; set R7 to 0,the location to store the result
0001 000 000 1 00100 ; R0 <-- 4
0001 001 001 1 00101 ; R1 <-- 5

0001 010 001 1 00000 ; R2 <-- R1
0000 011 000000100 ; R2 >= 0?
1001 000 000 111111 ; R0 <-- ~R0
0001 000 000 1 00001 ; R0 <-- R0 + 1
1001 001 001 111111 ; R1 <-- ~R1
0001 001 001 1 00001 ; R1 <-- R1 + 1
0000 010 000000011 ; R2 = 0?
0001 111 111 0 00 000 ; R7 <-- R7 + R0
0001 001 001 1 11111 ; R1 <-- R1 -1
0000 001 111111101 ; back to "R7 <-- R7 + R0"
1111 0000 00100101 ; halt
```

python程序

```
def lab1_1():
    R0 = int(input("请输入被乘数(整数,范围:-32768~32767): \n"))
    R1 = int(input("请输入乘数(整数,范围:-32768~32767): \n"))
    if (R1 < 0):
        R1 = -R1
        R0 = -R0
    R7 = 0
    while (R1 > 0):
        R7 += R0
        R1 -= 1
    print("结果为{0}\n".format(R7))
```

lab1_P

> 思路：用加法实现乘法，但将乘数分解为16*m+n的形式，并将较大数作为累加的量，另一个作为计数量,加快速度

原程序

```
0011 0000 0000 0000 ; start the program at location x3000
0101 000 000 1 00000 ; set R0 to 0
0101 001 001 1 00000 ; set R1 to 0
0101 111 111 1 00000 ; set R7 to 0,the location to store the result
0101 110 110 1 00000 ; set R6 to 0
0001 000 000 1 00100 ; R0 <-- 4
0001 001 001 1 00101 ; R1 <-- 5

;确定可以接受的逐个相加的基数为16

;1.let R1,R0 to be positive and record  "+-" of the result
0001 010 001 1 00000 ; R2 <-- R1
0000 011 000001001 ; R2 >= 0?
;R1<0
1001 001 001 111111 ; R1 <-- ~R1
0001 001 001 1 00001 ; R1 <-- R1 + 1,let R1 to be positive or zero
0001 011 000 1 00000 ; R3 <-- R0
0000 011 000000011 ; R3 >0
;;R0<0
1001 000 000 111111 ;  R0 <-- ~R0
0001 000 000 1 00001 ; R0 <-- R0 + 1
0000 111 000000111 ; 无条件跳转
;;R0>=0
0001 110 110 1 00001 ; R6 <-- 1 ,R6为1代表结果为负，为0代表结果为正,let R0 to
be positive or zero
0000 111 000000101 ;无条件跳转
;R1>=0
```

```
0001 011 000 1 00000 ; R3 <-- R0
0000 011 000000011 ; R3 >0
;;R0<0
1001 000 000 111111 ;  R0 <-- ~R0
0001 000 000 1 00001 ; R0 <-- R0 + 1
0001 110 110 1 00001 ; R6 <-- 1 ,R6为1代表结果为负，为0代表结果为正,let R0 to
be positive or zero
;;R0>=0

;2.compute the result,make R0 store the bigger
1001 100 001 111111 ; R4 <-- ~R1
0001 100 100 1 00001 ; R4 <-- R4 + 1
0001 101 000 0 00 100 ; let R5 to be R0-R1
0000 011  000000011   ; R5>=0?
;R0<R1
0001 100 000 1 00000 ; R4 <-- R0
0001 000 001 1 00000 ; R0 <-- R1
0001 001 100 1 00000 ; R1 <-- R4

;make R5 to be 16*R0
0101 101 101 1 00000 ; set R5 to 0
0001 101 101 0 00 000 ; R5 <-- R5 + R0
0001 101 101 0 00 101 ; R5 <-- R5 + R5
0001 101 101 0 00 101 ; R5 <-- R5 + R5
0001 101 101 0 00 101 ; R5 <-- R5 + R5
0001 101 101 0 00 101 ; R5 <-- R5 + R5

0001 100 001 1 00000 ; R4 <-- R1
0000 010 000001101  ;R4 = 0 ?
;;R1>0
0001 100 100 1 10000 ;R4 <-- R4-16
0000 100 000000100 ;R4<0?
0001 111 111 0 00 101 ; R7 <-- R7 + R5
0001 001 001 1 10000 ;R1 <-- R1-16
0000 001 111111011 ; back to "R4 <-- R4-16"
0000 010 000000011 ; R1 =0?

0001 111 111 0 00 000 ; R7 <-- R7 + R0
0001 001 001 1 11111 ; R1 <-- R1 -1
0000 001 111111101 ; back to "R7 <-- R7 + R0"

0001 110 110 1 00000 ; R6 <-- R6
0000 010 000000010 ;R6=0?
1001 111 111 111111 ;  R7 <-- ~R7
0001 111 111 1 00001 ; R7 <-- R7 + 1
1111 0000 00100101 ; halt
```

python程序

```
def lab1_p():
    R0 = int(input("请输入被乘数(整数,范围:-32768~32767): \n"))
    R1 = int(input("请输入乘数(整数,范围:-32768~32767): \n"))
    R6 = 0                          # 符号位,0代表为正,1代表为负
    if(R1>0):
        if(R0<0):
            R6 = 1
            R0 = -R0
    else:
        R1 = -R1
        if(R0>0):
            R6 = 1
        else:
            R0 = -R0
    if(R0<R1):
        R0,R1 = R1,R0
    R5 = 16 * R0
    R4 = R1
    R7 = 0
    while(True):
        R4 -= 16
        if(R4 < 0 ):
            break
        R7+=R5
        R1-=16
    while(R1>0):
        R7+=R0
        R1-=1
    if(R6):
        R7 = -R7
    print("结果为{0}\n".format(R7))
```

lab2

> 思路:求出类斐波那契数列中某项的值，可通过从*1*开始遍历求解每一项并与代求量比较获得结果

原程序

```
        .ORIG x3000
        AND R7,R7,#0
        AND R6,R6,#0
        AND R5,R5,#0
        AND R4,R4,#0
        AND R3,R3,#0
        AND R2,R2,#0
        AND R1,R1,#0
```

```
        ADD R1,R1,#1
        ADD R2,R2,#2
        ADD R3,R3,#4
        LD  R4,MOD

DECIDE  ADD R0,R0,#-1
        BRz OVER

CALC    ADD R7,R3,#0
        ADD R3,R3,R1
        ADD R3,R3,R1
        AND R3,R3,R4
        ADD R1,R2,#0
        ADD R2,R7,#0
        BRnzp DECIDE

OVER    ADD R7,R1,#0
        TRAP x25
MOD     .FILL #1023
FA      .FILL #930
FB      .FILL #18
FC      .FILL #710
FD      .FILL #370
```

python程序

```
def lab2():
    R7 = 1
    R1 = 1
    R2 = 1
    R3 = 2
    R4 = 1023
    R0 = int(input("请输入代求数(整数,范围:1~16384): \n"))
    R0 -= 1;
    if(R0==0):
        print("结果为{0}\n".format(R7))
    while(True):
        R7 = R3
        R0 -=1
        if(R0 == 0):
            print("结果为{0}\n".format(R7))
            break
        else:
            R3 += R1
            R3 += R1
            R1 = R2
            R2 = R7
```

```
            R3 = R3&R4
```

lab3

> 思路：采用预先存储要求的所有值并采用哈希表的思想可快速求出*lab2*中的代求值

原程序

```
.ORIG x3000
LD R1 base
ADD R1 R1 R0
LDR R7 R1 #0
HALT
base .FILL x3005
.FILL #1
.FILL #1
.FILL #2
.FILL #4
.FILL #6
.FILL #10
.FILL #18
.FILL #30
.FILL #50
.FILL #86
.FILL #146
.FILL #246
.FILL #418
.FILL #710
.FILL #178
.FILL #1014
.FILL #386
.FILL #742
.FILL #722
.FILL #470
.FILL #930
.FILL #326
.FILL #242
.FILL #54
.FILL #706
.FILL #166
.FILL #274
.FILL #662
.FILL #994
.FILL #518
.FILL #818
.FILL #758
.FILL #770
.FILL #358
.FILL #850
```

```
.FILL #342
.FILL #34
.FILL #710
.FILL #370
.FILL #438
.FILL #834
.FILL #550
.FILL #402
.FILL #22
.FILL #98
.FILL #902
.FILL #946
.FILL #118
.FILL #898
.FILL #742
.FILL #978
.FILL #726
.FILL #162
.FILL #70
.FILL #498
.FILL #822
.FILL #962
.FILL #934
.FILL #530

.end
```

由于原程序过长，这里做截断处理

python程序

```
def lab3():
    R0 = int(input("请输入代求数(整数,范围:1~16384): \n"))
    list = []
    list.append(1)
    list.append(1)
    list.append(2)
    for i in range(3,16385):
        x = (list[i-1]+2*list[i-3])%1024
        list.append(x)
    R7 = list[R0]
    print("结果为{0}\n".format(R7))
```

lab4_rec

> 思路: 通过尝试填补缺失的代码,发现其主要功能为将内存中的一个值赋到Ro,用python模拟时，为观测寄存器的变化，设置Ro~R7标记变量,高级语言函数返回自动实现，用python实现时，通过手动更改标记变量以反映PC的变化

原程序

```
1110010000001110
0101000000100000
0100100000000001
1111000000100101
0111111010000000
0001010010100001
0001000000100001
0010001000010001
0001001001111111
0011001000001111
0000010000000001
0100111111111000
0001010010111111
0110111010000000
1100000111000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000101
```

python程序

```
def lab4_rec():
    mem = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5] #前十五个placeholder代表代码
    R2 = 15           # mem[15]的位置
    R0,R1,R3,R4,R5,R6 = 0,0,0,0,0,0
    def f():
        nonlocal R7,R0,R1,R2,mem
        mem[R2] = R7
        R2 += 1
        R0 += 1
        R1 = mem[-1]
```

```
            R1 -= 1
            mem[-1] = R1

            if (R1 != 0):
                R7 = 12
                f()
            R2 -= 1
            R7 = mem[R2]
        R7 = 3
        f()
        print("R0:{0},R1:{1},R2:{2},R3:{3},R4:{4},R5:{5},R6:{6},R7:
{7}".format(R0,R1,R2,R3,R4,R5,R6,R7))
```

lab4_mod

> 思路：求一个数模7的余数，利用 *l = 8*m+n (o<=n<8)* 则*l mod 7 = (m+n) mod 7*

原程序

```
0010001000010101
0100100000001000
0101010001100111
0001001010000100
0001000001111001
0000001111111011
0001000001111001
0000100000000001
0001001001111001
1111000000100101
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
0101101011000001
0000010000000001
0001100010000100
0001010010000010
0001011011000011
0000001111111010
1100000111000000
0000000100100000
```

python程序

```
def lab4_mod():
    target = int(input("请输入参数：\n"))
    R0, R1, R2,R3, R4, R5, R6,R7 = 0, 0, 0, 0, 0, 0,0,0
    R1 = target
    def f():
        nonlocal R1,R2,R3,R4,R5
        R2,R3,R4 = 0,0,0
        R2 += 1
        R3 += 8
        while(True):
            R5 = R3&R1
            if(R5!=0):
                R4 += R2
            R2 +=R2
            R3 +=R3
            if(R3>=0b1000000000000000):
                break
    while(True):
        f()
        R2 = R1 & (0b111)
        R1 = R2 + R4
        R0 = R1 - 7
        if (R0 <= 0):
            break
    R0 = R1 -7
    if(R0>=0):
        R1 = R1-7
    print("结果为{0}".format(R1))
```

lab5

> 思路:将给定的一段*C++*代码用汇编实现，关键在于函数调用及返回机制，通过*JSR,JSRR,RET*等实现，由于高级语言函数返回自动实现，用*python*实现时，通过标记变量，并手动更改标记变量以反映*PC*的变化

原程序

```
        .ORIG x3000
        JSR JUDGE
        HALT
JUDGE   AND R6,R6,#0
        ADD R6,R6,#2
        AND R1,R1,#0
        ADD R1,R1,#1
HEAD    AND R4,R4,#0
        ADD R5,R6,#0
LOOP    ADD R4,R4,R6
        ADD R5,R5,#-1
```

```
        BRp LOOP
        NOT R4,R4
        ADD R4,R4,#1
        ADD R4,R0,R4
        BRn Tail
        NOT R4,R6
        ADD R4,R4,#1
        ADD R3,R0,#0
Calc    ADD R3,R3,R4
        BRp Calc
        BRz BREAK
        ADD R6,R6,#1
        BRnzp HEAD
BREAK   AND R1,R1,#0
Tail    RET
        .END
```

python程序

```
def lab5():
    R0 = int(input("请输入参数：\n"))
    def Judge():
        R6 = 2
        R1 = 1
        while(True):
            R4 = 0
            R5 = R6
            while(R5 > 0):
                R4 += R6
                R5 -= 1
            R4 = -R4
            R4 += R0
            if(R4 <0):
                return R1
            else:
                R4 = -R6
                R3 = R0
                while(R3 >0):
                    R3 += R4
                if(R3 == 0):
                    R1 = 0
                    return R1
                R6 += 1
    R1 = Judge()
    print("结果为{0}\n".format(R1))
```

[实验总结]

通过本次实验可以发现低级语言与高级语言的差别，对于高级语言，基于内存方面的基本操作大大抽象简化，同时基于地址方面的基本操作也大大抽象简化，对于PC的在地址空间中的反复变化并抽象掉了，程序员不需再考虑，大大降低了思考的复杂度，但另一方面也丧失了对内存，地址更细致的控制，不过结果显然是值得的，同时在高级语言中模块化及代码的组织也大大简化了思考的复杂度，而在低级中不同模块可能彼此交错，仅通过BR等跳转命令分割，容易让人搞混。