lab8

> *本实验在Linux环境（Ubuntu 22.04）下完成*

实验目的

使用高级语言C++重新完成lab1~lab4，体会汇编语言编程与高级语言编程思路的异同。

实验方法

lab1：Counting zero

实验思路同lab1：

1. 与1相与来判断奇偶，若是偶数要取其相反数；
2. 左移（与自己相加）16次，每次通过判断正负来判断最高位是否为0，是则让计数器加1。
   代码如下：

```
int16_t lab1(int16_t n) {
    // initialize
    int ret = STUDENT_ID_LAST_DIGIT;  // 返回结果
    if ((n & 1) == 0) {               // 偶数
        n = -n;                       // 负数补码
    }

    // calculation
    for (int i = 0; i < 16; i++) {  // 左移16次
        if (n >= 0) {
            ret++;
        }
        n = n + n;  // 左移1位
    }

    // return value
    return ret;
}
```

lab2：Pingpong sequence

实验思路大致同lab2：

1. 初始化为$n = 1$的情况，循环计算；
2. `d`的值设为2或-2，这样更新`v`时可以直接`v = v + v + d`，需要改变方向时取其相反数即可；

3. 由于实验文档禁止使用取余运算符，因此对4096和8的取余可以通过分别与4095和7相与来完成；
4. 至于判断个位数是否为8，在lab2中我选择保存每一次的个位数来递推以减小复杂度，但考虑到C++易于编写，这里直接逐次-1000，-100，-10至个位即可。

```
int16_t lab2(int16_t n) {
    // initialize
    int v = 3, d = 2;  // N = 1

    // calculation
    for (int i = 2; i <= n; i++) {  // 从N = 2开始
        v = v + v + d;              // 更新v
        v = v & 4095;               // 对4096取余
        if (v & 7 == 0) {           // v可被8整除
            d = -d;                 // 改变方向
            continue;
        }
        int t = v;
        while (t > 1000) {
            t = t - 1000;
        }
        while (t > 100) {
            t = t - 100;
        }
        while (t > 10) {
            t = t - 10;
        }
        if (t == 8) {  // v的个位为8
            d = -d;    // 改变方向
        }
    }

    // return value
    return v;
}
```

lab3：String compare

实验思路大致同lab3：

1. 逐次比较两个字符串的字符，直至遇到`'\0'`；
2. 指针的解引用操作很方便地替代了LC-3中的`LDR`指令，使得整个函数只需要两个指针变量就能完成，无需像汇编程序那样需要两个额外的寄存器来取出字符。

```
int16_t lab3(char s1[], char s2[]) {
    // calculation
    while (*s1 != '\0' && *s2 != '\0') {
        if (*s1 == *s2) {  // 两个字符相等，则继续
            s1++;
            s2++;
```

```
        } else {
            break;
        }
    }

    // return value
    return *s1 - *s2;
}
```

lab4：Baguenaudier

实验思路大致同lab4，根据 `REMOVE` 和 `PUT` 的递推式：

$$R(0) = nothing\ to\ do, R(1) = remove\ the\ 1^{st}\ ring,$$

$$R(i) = R(i-2) + remove\ the\ i^{th}\ ring + P(i-2) + R(i-1), i \geq 2.$$

$$P(0) = do\ nothing,\ P(1) = put\ the\ 1^{st}\ ring$$
$$P(i) = P(i-1) + R(i-2) + put\ the\ i^{th}\ ring + P(i-2)$$

定义两个递归函数 `remove` 和 `put`：

```
void remove(int16_t* memory, int16_t& state, int16_t n, int& move) {
    // @param memory: 存储状态的数组
    // @param state: 当前状态
    // @param n: 要处理的珠子数
    // @param move: 当前已经作用的次数

    if (n == 0) {  // n == 0, do nothing
        return;
    }
    if (n == 1) {                // n == 1, remove the 1st ring
        state = state + 1;       // flip the rightmost bit from 0 to 1
        memory[move++] = state;  // 存入`memory`
    } else {                     // n >= 2
        remove(memory, state, n - 2, move);
        int addend = 1;
        for (int i = 2; i <= n; i++) {
            addend = addend + addend;
        }
        state = state + addend;  // remove the n-th ring (flip the n-th rightmost bit
from 0 to 1)
        memory[move++] = state;  // 存入`memory`
        put(memory, state, n - 2, move);
        remove(memory, state, n - 1, move);
    }
}

void put(int16_t* memory, int16_t& state, int16_t n, int& move) {
    // @param memory: 存储状态的数组
    // @param state: 当前状态
```

```
    // @param n: 要处理的珠子数
    // @param move: 当前已经作用的次数

    if (n == 0) {  // n == 0, do nothing
        return;
    }
    if (n == 1) {                // n == 1, put the 1st ring
        state = state - 1;       // the rightmost bit
        memory[move++] = state;  // 存入`memory`
    } else {                     // n >= 2
        put(memory, state, n - 1, move);
        remove(memory, state, n - 2, move);
        int subend = 1;
        for (int i = 2; i <= n; i++) {
            subend = subend + subend;
        }
        state = state - subend;  // remove the n-th ring (flip the n-th rightmost bit
from 1 to 0)
        memory[move++] = state;  // 存入`memory`
        put(memory, state, n - 2, move);
    }
}
```

1. 它们除了接受`state`和`n`两个参数外，还有`memory`和`move`两个参数，以便在remove / put某一环后可以将状态存入`memory[move]`内，并递增`move`。
2. `state`和`move`都以引用的方式传入函数，使得函数可以原地修改它们的值。
3. remove / put第i环的方式同lab4，即将1左移$i-1$位来设置掩码，对应做加法或减法即可。主函数调用`REMOVE(n)`即可，代码如下：

```
int16_t lab4(int16_t* memory, int16_t n) {
    // initialize
    int16_t state = 0;
    int move = 0;

    // calculation
    remove(memory, state, n, move);

    // return value
    return move;
}
```

使用C++编写的递归程序无需像LC-3那样考虑各种寄存器的保存，更加简易。

遇到的Bug与解决方法

由于本次实验较为简单，故只遇到了一个很微妙的bug。在lab1判断奇偶时，我最开始编写的与1相与的代码是这样的：

```
if (n & 1 == 0)
```

结果测试的时候奇数的结果总是对的，偶数总是错的，说明偶数处理的逻辑出现了问题。
于是我将断点设在偶数处理的那一行，进行调试，结果发现程序根本到不了那里。
我百思不得其解，最后发现自己忘了运算符的优先级：`&`的优先级小于`==`，所以这句代码会先执行后面的`1 == 0`，这是恒错的，所以这段`if`永远不会被执行。
我加上括号后问题就解决了。

程序测试

文档内含样例测试

编译和运行程序如下：

```
g++ lab8.cpp -DLENGTH=1 -o lab8
./lab8
```

输出为：

```
===== lab1 =====
21
===== lab2 =====
786
===== lab3 =====
115
===== lab4 =====
0000000000000001
0000000000000101
0000000000000100
0000000000000110
0000000000000111
```

lab1中我的学号最后一位是7，而代码中默认的是3，所以我的答案比文档提供的答案大4，其余完全一致。

自编样例测试

我准备了一份每个lab有34个测试样例的文件`test.txt`：

```
98
803
7246
13598
76
518
```

```
3854
30385
91
672
2008
25130
39
207
4912
19444
76
984
6360
11516
30
445
7121
24729
100
990
2177
29933
92
291
7295
5
15
6280
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27
28
29
30
32767
188
2992
1580kJHgfDSaQWeRtyUIoPmlKJhgfDsAzXcvbNmPoiuyTrewQlkjHgfdsaZxcvbnmQAZwsxEedcRfvTgbYhnUj
mIolkPOIUYtrewqLKJHGFdsa
kJHgfDSaQWeRtyUIoPmlKJhgfDsAzXcvbNmPoiuyTrewQlkjHgfDSajkljlHHZxcvbnmQAZwsxEedcRfvTgbYh
nUjmIolkPOIUYtrewqLKJHGFdsa
xYzWxyZ            xYzWxyYZA
qWeRtYuIo         qWeRtYiUoP
lMnOpQrSt         lMnOpQrUsV
zxcvbnmasd        zxcvbnmasad
jHgFdsA            jHgFdsBAaa
qWeAsD             qWeAsDEeee
UiOpJhKnL          UiOpJhKmLnn
RtyUfv             RtyUfvgaaaa
pOiUyTr           pOiUyTsjkjf
QaWsEdFr          QaWsEdFtyik
xCvBnM            xCvBnNhdghf
PlMkOnJh          PlMkOnJiwke
zXcVbNm           zXcVbNmdfsk
watchoutnowtakecareBewareOfFallingSwingersDroppingAllAroundYouThePainThatOftenMinglesI
nyourFingertipsbeWAREofDARKNESS
watchoutnowtakecareBewareOfFallingSwingersDroppingAllAroundYouThePainThatOftenMinglesI
nyouxFingertipsbeWAREofDARKNESS
AsDfGh            AsDfGiyyyrr
HjKlJhKl          HjKlJhKmjki
qWeR              qWeTrqqwwee
AbCdEf            AbCdEgggaaaa
ZxcvbnmLKJHGFDSlkjhgfdsaPOIUYTREWQqwertyuioplkjhgfdsmnbvcxz
ZxcvbnmLKJHGFDSlkjhGfdsaPOIUYTrEWQqwertyuioplkjhgfdsmnbvcxz
pOiuY              pOiuZghfgfff
LmNkLmN           LmNkLmOljklm
XcVbNm            XcVbNnpopopo
qWeRqWeR          qWeRqWeSdfgh
AsDfAsDf          AsDfAsDggggg
GvtrEPOINZMxBCXYRUjKShyueWqDLAplQkHsdrgfTIOJNMZXCvnmPLOKIUHytRvbnmwqxd
GvtrEPOINZMxBCXYRUjKShyueWqDLAplQkHsdrgfTIOJNMZX
PoiuPoiu          PoiuPoivdfgsa
LkjHgf            LkjHghaslkdf
XyZxYz            XyZxYvwqertt
QwErTy            QwErTzvnmzxc
mNoPqR            mNoPqSaasdfgh
zfz gfg
bfb bfb
DsTAs DsTA
1
2
3
```

```
4
5
6
7
8
9
10
11
12
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
0
0
```

每个实验的输入数据我都用python脚本处理成对应格式，在自动评测机上得到正确答案，再用python脚本提取答案。
以lab1为例，这是将每行一个数据的格式处理成自动评测机接收格式的脚本：

```python
with open("lab1_data.txt", "w") as f:
    for num in nums:
        f.write(str(num) + "\n")

with open("lab1_data_lc3.txt", "w") as f:
    for num in nums:
        f.write(str(num) + ":22000197,")
```

自动评测机的输出结果如下：

lab8 > ans.txt

```
失败 98:22000197, 指令数: 0, 输出: 0,0, 预期: 7,10
失败 803:22000197, 指令数: 0, 输出: 0,0, 预期: 7,18
失败 7246:22000197, 指令数: 0, 输出: 0,0, 预期: 7,14
失败 13598:22000197, 指令数: 0, 输出: 0,0, 预期: 7,15
失败 76:22000197, 指令数: 0, 输出: 0,0, 预期: 7,11
失败 518:22000197, 指令数: 0, 输出: 0,0, 预期: 7,10
失败 3854:22000197, 指令数: 0, 输出: 0,0, 预期: 7,14
失败 30385:22000197, 指令数: 0, 输出: 0,0, 预期: 7,14
失败 91:22000197, 指令数: 0, 输出: 0,0, 预期: 7,18
失败 672:22000197, 指令数: 0, 输出: 0,0, 预期: 7,14
失败 2008:22000197, 指令数: 0, 输出: 0,0, 预期: 7,16
失败 25130:22000197, 指令数: 0, 输出: 0,0, 预期: 7,13
```

我要的正确答案为每行最后一个逗号后的数字，因此使用如下脚本来提取：

```
with open("ans.txt", "r") as input_file:
    lines = input_file.readlines()
    extracted_numbers = [re.split(r'[,\s]+', line.strip())[-1] for line in lines]
```

4个lab全部提取出的答案存放在`ans_processed.txt`文件，形如：

lab8 > ans_processed.txt

```
-1
-1
67
-1
-2
4
-1
-1
19
0
115
===== lab4 =====
0000000000000001
0000000000000010
0000000000000011
0000000000000001
0000000000000101
0000000000000100
```

依次执行如下命令：

```
g++ lab8.cpp -DLENGTH=34 -o lab8
./lab8 >> out.txt
diff out.txt ans_processed.txt
```

没有输出，说明程序输出与标准答案完全一致。