LAB_05 取模解密(RIDDLE)实验报告

代码分析

当我们将c程序进行LC3汇编语言转译时，我们需要进行判断有哪些功能是LC3的基本指令所不能实现的。然后针对这些功能我们需要设计相应的算法。

```
int judge(int r0) {
    int i = 2;
    r1 = 1;
    while (i * i <= r0) {   //LC3没有平方指令
        if (r0 % i == 0) {  //LC3没有取模指令
            r1 = 0;
            break;
        }
        i++;
    }
    return r1;
}
```

如上图所示，我们需要对平方和取模判断两种功能单独进行设计。

设计思路

主函数

对于函数主体，我们只需要进行调用JUDGE函数即可。

```
LD R1 NUMB; R1初始化
LD R2 NUMA; R2初始化
JSR JUDGE;  调用JUDGE函数
HALT;       结束
        ;下面其他函数
        ;... ...
        ;下面是初始化区域
NUMA .FILL #2;
NUMB .FILL #1;
ST7 .BLKW #1;
```

JUDGE 函数

我们在JUDGE函数中，会用到平方和取模两种操作，所以我们会在这些地方进行JSR操作，为了保证能顺利的RETURN回主函数，我们需要将R7的值进行存储。

```
JUDGE ST R7,ST7;        将返回地址存入ST7中
```

```
DECIDE JSR SQR;          调用平方函数，执行取i^2操作，它的值存储在R4中
NOT R4,R4;
ADD R4,R4,#1;            将R4取反加一，得到-i^2
ADD R4,R4,R0;            R4=r0-i^2
BRn RETURN2;             如果R4<0,则不满足while循环的条件
JSR MOD;                 判断r0是否是i的整数倍，判断标准是R4是否为0
ADD R4,R4,#0;
BRp PLUS;                如果不满足判断条件，则直接执行i++
AND R1,R1,#0;            进入if语句中，执行r1 = 0;
BRnzp RETURN2;           break;
PLUS ADD R2,R2,#1;       执行i++
BRnzp DECIDE;            循环
RETURN2 LD R7,ST7;       将原地址存回R7
RET;                     RETURN
```

平方函数(SQR)

这里我们采用最朴素的算法，将i自加i次，结果存入R4中，这里用到两个寄存器，R3、R4初始赋值均为R2(存储值为i)，其中R4执行i次R4+i；R3作为计数器，R4每执行一次加i操作则R3减1，直到R3归0，则代表函数完成操作，结束调用并返回。

```
SQR AND R4,R4,#0;        初始化，将R4清零
ADD R3,R2,#0;            R3=i;
BRz RETURN1;             如果i=0，则代表平方值为0，无意义
LOOP ADD R4,R2,R4;       R4=R4+R2
ADD R3,R3,#-1;           R3--
BRp LOOP;                如果R3不为零代表未完成
RETURN1 RET;             R3为0，结束相加，RETURN
```

模除判断函数(MOD)

通过LAB4，我们已经知道对于非2的整数倍的取余操作是比较困难的，而我们在这里只是为了判断r0是否是i的整数倍，所以我们可以采取一个更取巧的方法：将R2（i）存入R3中，不断自加，得到i的各正整数倍的值。将R0（r0）取反加一得到它的负数值，存入R5中，通过把R5和R3的和放入R4中判断能否模除为0（$R4 = -r0 + n * i$）,r0是i的正整数倍，则R4会出现值为0的情况，否则会直接出现正数，我们将此作为判断依据。

```
MOD AND R4,R4,#0;        R4清零
AND R3,R3,#0;            R3清零
NOT R5,R0;
ADD R5,R5,#1;            R5=-r0
OPR ADD R3,R3,R2;        R3=R3+i;
ADD R4,R3,R5;            R4=R3+R5(R4=-r0+n*i)
BRn OPR;                 R4为负数，代表还没有达到判断要求，继续执行循环
RET;                     RETURN
```

完整代码及正确性检验

通过上述各模块设计思路的整合，我们可以得到如下的实验代码。

```
.ORIG x3000
LD R1 NUMB;
LD R2 NUMA;
JSR JUDGE;
HALT

JUDGE ST R7,ST7;
DECIDE JSR SQR;
NOT R4,R4;
ADD R4,R4,#1;
ADD R4,R4,R0;
BRn RETURN2;
JSR MOD;
ADD R4,R4,#0;
BRp PLUS;
AND R1,R1,#0;
BRnzp RETURN2;
PLUS ADD R2,R2,#1;
BRnzp DECIDE;
RETURN2 LD R7,ST7;
RET;

SQR AND R4,R4,#0;
ADD R3,R2,#0;
BRz RETURN1;
LOOP ADD R4,R2,R4;
ADD R3,R3,#-1;
BRp LOOP;
RETURN1 RET;

MOD AND R4,R4,#0;
AND R3,R3,#0;
NOT R5,R0;
ADD R5,R5,#1;
OPR ADD R3,R3,R2;
ADD R4,R3,R5;
BRn OPR;
RET;

NUMA .FILL #2;
NUMB .FILL #1;
ST7 .BLKW #1;
.END
```

为了检验代码的正确性，我们可以将题干中的C程序补全，得到如下代码。

```
int judge(int r0) {
    int i = 2;
    int r1 = 1;
    while (i * i <= r0) {
        if (r0 % i == 0) {
            r1 = 0;
            break;
        }
        i++;
```

```
    }
    return r1;
}

int main(){
    int r0;
    printf("请输入R0的值:");
    scanf("%d",&r0);
    printf("R1的值是%d\n",judge(r0));
    system("pause");
    return 0;
}
```

带入2, 3, 4, 7, 456, 993, 997, 1569, 9293, 121, 9339, 1437，得到结果依次是1，1，0，1，0，0，1，0，1，0，0，0。

正确无误。