第六次实验

姓名: 傅申 学号: PB20000051

本次实验使用 C++ 来实现之前写过的程序.

1. 程序
    1. lab0l
    2. lab0p
    3. fib
    4. fib-opt
    5. rec
    6. mod
    7. prime
2. 问题
    1. 评价性能
    2. 为什么写高级语言更容易
    3. 给 LC-3 添加一条指令
    4. 从 LC-3 中学到的技巧

1 程序

1.1 lab0l

原汇编程序如下

```
LOOP    ADD     R7, R7, R1
        ADD     R0, R0, -1
        BRnp    LOOP
STOP    HALT
```

可以看出程序大致实现了一个 do-while 循环, 写成 C++ 高级语言函数如下

```
void lab0l(short &r0, short &r1, short &r7)
{
    do
    {
        r7 += r1;
        r0 --;
    } while (r0 != 0);
}
```

1.2 lab0p

原汇编程序如下

```
        ADD     R0, R0, #0
        BRz     Stop

        BRp     Pos
        NOT     R0, R0      ; Negate R0
        ADD     R0, R0, #1
        NOT     R1, R1      ; Nagate R1
        ADD     R1, R1, #1

Pos     ADD     R2, R2, #1
Loop    AND     R3, R0, R2
        BRnp    BitOne

```

```
13
14          ADD     R1, R1, R1
15          ADD     R2, R2, R2
16          BRnzp   Loop
17
18  BitOne  ADD     R7, R7, R1
19          ADD     R1, R1, R1
20          ADD     R2, R2, R2
21          ADD     R4, R0, #-1 ; Remove the lowest
22          AND     R0, R4, R0  ; 1 in R0
23          BRnp    Loop
24
25  Stop    HALT
```

其中 R2 作为掩码判断 R0 中的各位是否为 1, 如果是 1 就通过位运算 R0 & (R0 - 1) 消去, 将其转化成 C++ 函数如下

```cpp
void lab0p(short &r0, short &r1, short &r7)
{
    if (!r0)
    {
        return;
    }
    else if (r0 < 0)
    {
        r0 = - r0;
        r1 = - r1;
    }
    short mask = 1;
    while (r0)
    {
        if (mask & r1)
        {
            r7 += r1;
            r0 &= r0 - 1;
        }
        r1 += r1;
        mask += mask;
    }
}
```

1.3 fib

原汇编程序如下

```
        ADD     R7, R7, #1 ; F(n)
        ADD     R1, R1, #1 ; F(n + 1)
        ADD     R2, R2, #2 ; F(n + 2)
        LD      R4, MOD
LOOP    ADD     R3, R7, R7 ; temp = 2 * F(n - 1)
        ADD     R7, R1, #0
        ADD     R1, R2, #0
        ADD     R2, R3, R1 ; F(n + 2) = F(n + 1) + 2 * F(n - 1)
        ADD     R0, R0, #-1
        BRp     LOOP
        AND     R7, R7, R4
        HALT
MOD     .FILL   #1023
```

显然这是一个递推程序, `R7, R1, R2` 分别为 $f(n), f(n+1), f(n+2)$, 写成 C++ 函数如下

```
void fib(short &r0, short &r7)
{
    short f = 1, f1 = 1, f2 = 2;
    do
    {
        temp = f + f;
        f = f1;
        f1 = f2;
        f2 = temp + f1;
        r0 --;
    } while (r0 > 0);
    r7 = f & 1023;
}
```

1.4 fib-opt

原程序是一个打表程序, 主体部分如下

```
        ADD R0, R0, #-16
        ADD R0, R0, #-4
        BRn NEG
        LD  R6, MOD
        AND R0, R0, R6
NEG     LD  R1, BASE
        ADD R1, R0, R1
        LDR R7, R1, #0
        HALT
MOD     .FILL #127
BASE    .FILL x301F     ; Addr of f(20)
; fib array here
; ....
```

写成 C++ 函数的形式

```
void fib_opt(short &r0, short &r7)
{
    short fib[] =
    {
        1, 1, 2, 4, 6, 10, 18, 30, 50, 86,
        146, 246, 418, 710, 178, 1014, 386, 742, 722, 470,
        930, 326, 242, 54, 706, 166, 274, 662, 994, 518,
        818, 758, 770, 358, 850, 342, 34, 710, 370, 438,
        834, 550, 402, 22, 98, 902, 946, 118, 898, 742,
        978, 726, 162, 70, 498, 822, 962, 934, 530, 406,
        226, 262, 50, 502, 2, 102, 82, 86, 290, 454,
        626, 182, 66, 294, 658, 790, 354, 646, 178, 886,
        130, 486, 210, 470, 418, 838, 754, 566, 194, 678,
        786, 150, 482, 6, 306, 246, 258, 870, 338, 854,
        546, 198, 882, 950, 322, 38, 914, 534, 610, 390,
        434, 630, 386, 230, 466, 214, 674, 582, 1010, 310,
        450, 422, 18, 918, 738, 774, 562, 1014, 514, 614,
        594, 598, 802, 966, 114, 694, 578, 806, 146, 278,
        866, 134, 690, 374, 642, 998, 722, 982
    };
    short *base = fib + 20;
    r0 -= 20;
    if (r0 >= 0) r0 &= 127;

```

```
    r7 = *(base + r0);
}
```

1.5 rec

将 rec.txt 翻译为汇编代码后为

```
        LEA     R2, #14     ; x300F → R2
        AND     R0, R0, #0  ; x0000 → R1
        JSR     FUNC
        HALT
FUNC    STR     R7, R2, #0
        ADD     R2, R2, #1  ; PUSH R7
        ADD     R0, R0, #1  ; R0 + 1 → R0
        LD      R1, #17     ; mem[x3019] → R1
        ADD     R1, R1, #-1 ; R1 - 1 → R1
        ST      R1, #15     ; R1 → mem[x3019]
        BRz     #1
        JSR     FUNC
        ADD     R2, R2, #-1
        LDR     R7, R2, #0  ; POP R7
        RET
        .BLKW   #10         ; x300F ~ x3018
        .FILL   #5          ; x3019
```

可以看出程序调用了一个递归的子程序, 这里称之为 FUNC, 除了出入函数栈操作外, FUNC 还不断地让 R0 自增, mem[x3019] 自减, 并且在 mem[x3019] 为 0 时退出递归. 根据以上分析, 写出程序的 C++ 函数如下

```
void rec(short &r0, short &mem)
{
    r0 = 0;
    func(r0, mem);
}

void func(short &r0, short &mem)
{
    r0 ++;
    short temp = mem;
    temp --;
```

```
    mem = temp;
    if (temp == 0) return;
    func(r0, mem);
}
```

1.6 mod

将 mod.txt 翻译为汇编代码后为

```
        .ORIG   x3000
        LD      R1, #21     ; x3016 → R1
LOOP1   JSR     DIV         ; Call DIV
        AND     R2, R1, #7  ; R1 % 8 → R2
        ADD     R1, R2, R4  ; R2 + R4 → R1
        ADD     R0, R1, #-7 ; R1 - 7 → R0
        BRp     LOOP1
        ADD     R0, R1, #-7 ; R1 - 7 → R0
        BRn     #1
        ADD     R1, R1, #-7 ; R1 - 7 → R1
        HALT
DIV     AND     R2, R2, #0  ; clear R2
        AND     R3, R3, #0  ; clear R3
        AND     R4, R4, #0  ; clear R4
        ADD     R2, R2, #1  ; R2 + 1 → R2
        ADD     R3, R3, #8  ; R3 + 8 → R3
LOOP2   AND     R5, R3, R1  ; R3 % R1 → R5
        BRz     #1
        ADD     R4, R2, R4  ; R4 + R2 → R4
        ADD     R2, R2, R2  ; R2 + R2 → R2
        ADD     R3, R3, R3  ; R3 + R3 → R3
        BRnp    LOOP2
        RET
        .FILL   #288        ; x3016
```

其中被调用的 DIV 子程序是通过掩码实现右移, 整个程序写成 C++ 函数的形式为

```
void mod(short n, short &r1)
{
    r1 = n;
    short quotient, remainder;
```

```
    do
    {
        div(r1, quotient);
        remainder = r1 & 7;
        r1 = quotient + remainder;
    } while (r1 > 7);
    r1 = (r1 == 7) ? 0 : r1;
}

void div(short &r1, short &r4)
{
    short mask0 = 8, mask1 = 1;
    do
    {
        if (r1 & mask0) r4 += mask1;
        mask0 = mask0 + mask0;
        mask1 = mask1 + mask1;
    } while (mask0);
}
```

1.7 prime

原汇编代码主要判断程序部分为:

```
; judge subroutine: R1 = R0.isPrime()
JUDGE   ST    R7, SaveR7
        AND   R1, R1, #0
        ADD   R1, R1, #1  ; R1 = 1
        NOT   R2, R0
        ADD   R2, R2, #1  ; R2 = -R0
        AND   R3, R3, #0
        ADD   R3, R3, #2  ; R3 = 2
AGAIN   JSR   SQUARE
        ADD   R4, R4, R2  ; R4 = R4 - R0
        BRp   DONE
MODULE  ADD   R4, R2, #0
AGAINm  ADD   R4, R4, R3
        BRn   AGAINm
        BRz   BAD
        ADD   R3, R3, #1
```

```
        BRnzp AGAIN
BAD     AND   R1, R1, #0
DONE    LD    R7, SaveR7
        RET
```

写成 C++ 函数的形式为

```
void prime(short &r0, short &r1)
{
    r1 = 1;
    short i = 2, squ, temp;
    while (1)
    {
        square(i, squ);
        if (squ > r0) break;
        temp = -r0;
        do
        {
            temp += i;
        } while (temp < 0);
        if (temp == 0)
        {
            r1 = 0;
            break;
        }
        i++;
    }
}
```

而调用的 SQUARE 子程序如下, 与 lab0p 中的思想一致,

```
; square subroutine: R4 = R3 * R3
SQUARE  ST    R1, SaveR1
        AND   R1, R1, #0
        ADD   R1, R1, #1  ; R1 = 1
        ST    R2, SaveR2
        ADD   R5, R3, #0
        ADD   R6, R3, #0
        ST    R3, SaveR3
        AND   R4, R4, #0
```

```
AGAINs  AND   R2, R1, R5
        BRnp  BitOne
        ADD   R6, R6, R6
        ADD   R1, R1, R1
        BRnzp AGAINs
BitOne  ADD   R4, R4, R6
        ADD   R6, R6, R6
        ADD   R1, R1, R1
        ADD   R3, R5, #-1
        AND   R5, R5, R3
        BRnp  AGAINs
        LD    R1, SaveR1
        LD    R2, SaveR2
        LD    R3, SaveR3
        RET
SaveR1  .BLKW #1
SaveR2  .BLKW #1
SaveR3  .BLKW #1
SaveR7  .BLKW #1
        .END
```

直接对 lab0p 的 C++ 程序进行部分修改即可得到

```
void square(short &r3, short &r4)
{
    short mask = 1, a = r3, b = r3;
    r4 = 0;
    do
    {
        if (a & mask)
        {
            r4 += b;
            a &= (a - 1);
        }
        b += b;
        mask += mask;
    } while (a);
}
```

最后得到整个程序的 C++ 函数

```
void prime(short &r0, short &r1)
{
    r1 = 1;
    short i = 2, squ, temp;
    while (1)
    {
        square(i, squ);
        if (squ > r0) break;
        temp = -r0;
        do
        {
            temp += i;
        } while (temp < 0);
        if (temp == 0)
        {
            r1 = 0;
            break;
        }
        i++;
    }
}

void square(short &r3, short &r4)
{
    short mask = 1, a = r3, b = r3;
    r4 = 0;
    do
    {
        if (a & mask)
        {
            r4 += b;
            a &= (a - 1);
        }
        b += b;
        mask += mask;
    } while (a);
}
```

2 问题

2.1 评价性能

采用时间复杂度来评价各个程序的时间性能, 如下

程序	时间复杂度	n 所代表的参数
lab0l	$O(n)$	`r0`
lab0p	$O(\log n)$	`r0`
fib	$O(n)$	`r0`
fib-opt	$O(1)$	
rec	$O(n)$	`mem`
mod	$O(\log n)$	`n`
prime	$O(n \log n)$	`r0`

而对于空间性能

- lab0l, lab0p, fib, mod, prime 没有使用到数组或递归, 它们的空间性能较好.
- fib-opt 使用了一段定长内存空间 (数组), 它的空间性能中等.
- 而 rec 则使用了递归, 若输入的参数过大, 可能有函数栈溢出的风险, 其空间性能较差.

2.2 为什么写高级语言更容易

1. 高级语言将许多固定的逻辑结构抽象出来, 并且更容易被人理解.
2. 高级语言替我们对一些细节进行了处理. 比如我们写递归程序时不需要手动操作函数栈.

2.3 给 LC-3 添加一条指令

我希望给 LC-3 添加一条右移指令, 具体格式为

```
LSH     DR, SR, imm4
```

实际操作为 `DR <- SR >> imm4`, 这样, lab0p 的实现可以更加简单, 而 mod 中除以 8 的操作也可以通过 `LSH R4, R1, #3` 来实现.

2.4 从 LC-3 中学到的技巧

1. 一些取模操作可以通过位运算来减小时间代价, 如 `a % 8 = a & 7`.
2. 在处理递归时, 要考虑到递归层数.

3. 某些情况下可以利用数学技巧来优化程序.