QLanguage 指令集

简介

QLanguage 的虚拟机是一个基于寄存器的虚拟机，之所以采用寄存器式的虚拟机是因为它相对栈式的虚拟机拥有更快的运行速度。

存储结构

```
struct Instruction
{
    uchar op : 5; //  操作码 5 位，最多 32 种
    uchar ot : 3; // 3 个操作数的类型 3 位，1 为常数，0 为寄存器
    union
    {
        struct
        {
            uchar    ob1; // 0 为全局的寄存器或常数，1 为当前块的，2 为上一级的.....
            uchar    ob2; // 0 为全局的寄存器或常数，1 为当前块的，2 为上一级的.....
            ushort os1; //  源寄存器 1 或常数  2 字节
            ushort os2; //  源寄存器 2 或常数  2 字节
            uchar    obd; // 0 为全局的寄存器或常数，1 为当前块的，2 为上一级的.....
            ushort od ; //  目的寄存器  1 字节
        }Normal;
        struct
        {
            int       addr;     //  跳转或函数位置
            uchar    ext;       //  未使用或参数个数
            uint     unused; //  未使用
        }JmpCall;
    };
};
```

由上面的代码可见，在虚拟机中每条指令由 10 个字节构成。前 5bit 表示该条指令的操作码，根据操作码的不同所用到的后面的结构也不同。之后的 3bit 分别表示两个源寄存器（或常数）和一个目的寄存器（或常数）的类型是常数或寄存器。

首先介绍跳转和 call 指令，这两类指令会用到 JmpCall 域，其中 addr 是要跳转的位置，当虚拟机读到此类指令后，直接将指令寄存器置为 addr 所指的位置然后继续执行，若这条指令是一个 call 指令，则 ext 域指示了所传参数的个数，当前块必须有足够的寄存器来存储这些参数，否则提示用户函数参数过多，无法继续执行。

对于其他指令则会用到 Normal 域，其中 ob1 和 ob2 分别表示当前指令所用到的寄存器

或常数是当前块的或是全局的，从结构中可以看出最多将拥有 255 层调用关系，应此能够非常有效的防止调用深度过深导致栈不足的错误。os1 和 os2 则表示了当前指令所用到的寄存器或常数的编号，从结构中可以看出每个块最多拥有 65535 个寄存器或常量。同样的 obd 表示目的寄存器所属的块，而 od 表示目的寄存器所对应的寄存器或常量序号。

规定

65534 号寄存器为符号寄存器，用于记录每次逻辑运算的结果。

65535 号寄存器为指令寄存器，用于记录当前指令的索引，应此在每次函数调用或跳转时它将被置为正确的值，同样的在函数返回时它也将被置为正确的值。

指令说明

```
enum OpCode
{
    Mov       = 0,   // R[a] = k || R[a] = R[b]
    Less      = 1,   // R[a] < k   || k < R[b]   || R[a] < R[b]
    More      = 2,   // R[a] > k   || k > R[b]   || R[a] > R[b]
    LessEqual = 3,   // R[a] <= k || k <= R[b] || R[a] <= R[b]
    MoreEqual = 4,   // R[a] >= k || k >= R[b] || R[a] >= R[b]
    Equal     = 5,   // R[a] == k || k == R[b] || R[a] == R[b]
    Add       = 6,   // R[a] = R[b] + k   || R[a] = k + R[c]   || R[a] = R[b] + R[c]
    Sub       = 7,   // R[a] = R[b] - k   || R[a] = k - R[c]   || R[a] = R[b] - R[c]
    Mul       = 8,   // R[a] = R[b] * k   || R[a] = k * R[c]   || R[a] = R[b] * R[c]
    Div       = 9,   // R[a] = R[b] / k   || R[a] = k / R[c]   || R[a] = R[b] / R[c]
    Mod       = 10, // R[a] = R[b] % k   || R[a] = k % R[c]   || R[a] = R[b] % R[c]
    LogicAnd  = 11, // R[a] = R[b] && k || R[a] = k && R[c] || R[a] = R[b] && R[c]
    LogicOr   = 12, // R[a] = R[b] || k || R[a] = k || R[c] || R[a] = R[b] || R[c]
    Not       = 13, // R[a] = !R[b]
    BitAnd    = 14, // R[a] = R[b] & k   || R[a] = k & R[c]   || R[a] = R[b] & R[c]
    BitOr     = 15, // R[a] = R[b] | k   || R[a] = k | R[c]   || R[a] = R[b] | R[c]
    BitXor    = 16, // R[a] = R[b] ^ k   || R[a] = k ^ R[c]   || R[a] = R[b] ^ R[c]
    Inc       = 17, // R[a] = R[a] + 1
    Dec       = 18, // R[a] = R[a] - 1
    Pos       = 19, // R[a] = +R[a]
    Neg       = 20, // R[a] = -R[a]
    Jmp       = 21, // jmp k
    Call      = 22, // call k1, k2
    Ret       = 23   // ret ([a] || [k])
};
```

从以上枚举类型可以看出，QLanguage 拥有 23 种指令，下面将分别说明这些指令的具体含义。

Mov

这条指令可用于将指定的常数或寄存器输出给另外一个寄存器，应此它仅会用到源寄存器（或常数）1 和目的寄存器。

Less

这条指令可用于比较两个寄存器或常数的值，比较完成后它将比较结果放到 65534 号寄存器中。

More

这条指令可用于比较两个寄存器或常数的值，比较完成后它将比较结果放到 65534 号寄存器中。

LessEqual

这条指令可用于比较两个寄存器或常数的值，比较完成后它将比较结果放到 65534 号寄存器中。

MoreEqual

这条指令可用于比较两个寄存器或常数的值，比较完成后它将比较结果放到 65534 号寄存器中。

Equal

这条指令可用于比较两个寄存器或常数的值，比较完成后它将比较结果放到 65534 号寄存器中。

Add

Sub

Mul

Div

Mod

LogicAnd

LogicOr

Not

BitAnd

BitOr

BitXor

Inc

Dec

Pos

Neg

Jmp

这条指令可用于实现函数内的跳转，应此要跳转到这个函数外是不允许的。

Call

Ret

尾声