优化关键点

优化点1： 常量传播

这个优化的关键点在于对运行过程的“模拟”

需要在本地建立对应的运行栈，与实际汇编代码保持同步进度，无需使用汇编代码的常数计算直接在本地栈内完成

同时为变量增设“变化”，认为其是常量，且其有初始值，即可直接进行值传递

而一旦其出现在 = 左侧，就必须认为其以及成为变量

这是不考虑数据流分析的简单做法

主要问题其实还是在于本地变量的存取

先给出数据结构,该优化与本地的符号表及栈都相关

```
struct identTable {
    string name = "error!";
    int ident_kind = 0; // 0 is a / 1 is a[] / 2 is a[][] / -1 is bool
    int dimension_one = 0;
    int dimension_two = 0;
    int value = 0;
    bool init = false;
    bool changeable = true;
    bool isGlobal = false;
    vector<int> arrayValue;
    string reg = "NULL!";
    int fg_offset = -1;
} ErrorIdent;
struct funcTable {
    string name = "error!";
    int return_kind = 0;    // 0 is int / 1 is void
    int params_num = 0;
    vector<identTable> params_list; // in fact, wo only need keep the ident_kind is true
    bool effective = true;
} ErrorFunc;
struct st{
    int kind = 0;   // 0 is value 1 is string
    int value = 0;
    string name = "error!";
//    bool key = true;
    int offset = -1;
    string reg = "NULL!";
    int fg_offset = -1;
    bool melt = false;
    bool isAddress = false;
} ErrorSt;
```

对于遇到的变量，需要先查询符号表，如果其可以认为是常量，则将 值 存入栈中

否则将其存储信息存入栈中

需要注意的是，其本质上仍然是变量，在值改变时，需要及时修改其属性

常量传播的基础完成并不复杂，主要是在Cond、数组、参数传递中的常量传播

对于这些情况，需要先把问题分化，这些问题的起源都在于对一个变量的取值，例如

```
a < 1
// 1直接进行传播，我们现在需要比较一个变量和1
// 取该变量的值
// 比较函数

b[a] = 1
// 1 直接进行传播
// 取a的值
// 计算b[a]

(a,1)
// 取a的值
// 参数传递
```

可以看出采取合适的方式完成对a的取值后，实质上要处理的仅仅是对固定格式变量的处理，就容易很多

而对a的取值核心思想就在于，如果认为它是常量，则直接当作常量带入

我们在后续的处理中要充分考虑到这一点

```
a = 5;
if(a < 5)

    //完全等价于 a = 5; if(0)
    //不需要考虑原有逻辑，用本地运算出的结果代替即可
```

优化点2： 寄存器分配

寄存器分配的关键在于要明确每个寄存器的使用价值

```
struct reg {
    string name = "NULL!";
    string reg;
    bool used = false;
} ErrorReg;
vector<reg> load;
vector<bool> opLoad;
vector<string> ParamsGet;
// zero at
reg retReg[2]; // v0 v1
reg paramReg[4]; // a0 a1 a2 a3

reg varReg[10]; // t6 - t7 s0 - s7
```

例如在我的分配中，我确认对于 + - * / ！ 等运算至多只需要5个寄存器即可完成，因此仅保留了五个寄存器作为临时结果寄存器，其余寄存器全部用作变量寄存器。

并且对于确定不生效的寄存器，要及时释放，例如在块结束时，块内新定义的变量的寄存器就都可以直接释放