report-PA4

在这次作业中，我实现了死代码消除、常量传播和复写传播。

死代码消除

死代码消除可以消除无用的语句，减少指令条数。经过常量传播和复写传播之后的代码容易出现很多死代码，这样就可以把它们消除掉，减少指令条数。例如下面这段代码中的 `a=b` 在复写传播后就会变成死代码，并被优化掉。（这里用 decaf 代码举了个例子，实际是对 TAC 优化的）

```
a = b;
Print(a);
// 之后没有用到 a
```

（基于活跃变量分析的）死代码消除只需要删除所有没有副作用且赋值目标不再活跃的语句。框架中已经实现了活跃变量分析，因此只需要几行代码就可以判断赋值目标是否活跃并删除了。

常量传播

在 PA3 生成出的 TAC 代码中，new 一个数组和给变量赋初值带来一些操作数都是常数的指令，这时候常量传播（包含常量折叠）可以帮助我们在编译期间确定哪些变量会是常量，并配合死代码消除减少无用的计算。例如下面这段代码就可以在编译期间计算出 `c` 的值，并删除 `a` 和 `b` 的赋值语句：

```
a = 1;
b = 2;
c = a + b;
```

实现方法参考了文档。先做一遍数据流分析找到每个变量的每次定值是否是常量，并针对每种指令进行处理。例如如果一个二元运算指令的两个操作数都是常量，我们就可以在编译期间确定运算结果。进一步的，如果一个二元运算指令的其中一个操作数是常量，有时候也可以帮助我们简化代码。

复写传播

在实际的代码中会有很多复写操作（以及给变量赋初值也会带来很多无用的复写语句），这时候复写传播可以配合死代码消除来消除冗余的复写语句。

实现方法参考了文档。先做一遍数据流分析找到每个变量的每次引用能替换成什么变量，然后对于每条指令的每个变量引用都进行替换。

性能测试结果

由于正常人写的代码在经过正常的编译器编译后不太可能产生死代码，因此我额外构造了一个测例（dead code elimination test）用于对比。

可以看出，这三种优化方法都有很好的效果。同时加入了这三种优化能在公开测例中减少约 5% 的指令。

program	without optimize	dead code elimination	constant propagation + dead code elimination	copy propagation + dead code elimination	copy propagation + constant propagation + dead code elimination
basic	41	41	38	37	37
fibonacci	3426	3426	3425	3425	3424
math	139	139	135	139	135
queue	2536	2536	2532	2475	2471
stack	733	733	730	729	726
mandelbrot	3893085	3893085	3782375	3815338	3704632
rbtree	2439827	2439827	2439816	2356805	2356794
sort	560987	560987	559980	558593	557586
dead code elimination test	2700009	700009	70008	700009	700008