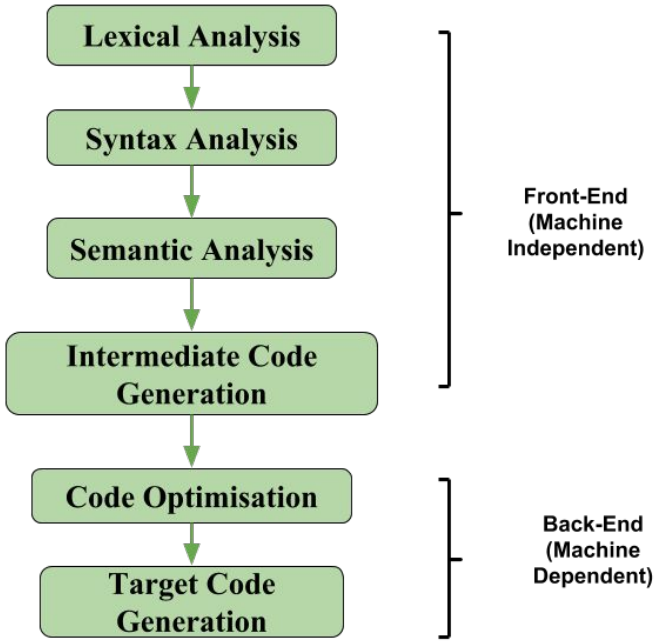
Three-address code
Static Single Assignment

Three-address code

- вид промежуточного представления кода (IR)
- подготавливает код к дальнейшей оптимизации
- используется оптимизирующими компиляторами
- по нему компилятор определяет порядок операций

- использует не более трех адресов и одного оператора для представления выражения
- результат каждой инструкции сохраняется во временную переменную, сгенерированную компилятором



Общий вид

```
a = b op c
```

- a, b и c - имена, константы или временные файлы, сгенерированные компилятором
- op - оператор

Пример 1

$$a * - (b + c)$$

Пример 2

```
for(i = 1; i<=10; i++)
 {
  a[i] = x * 5;
 }
```

Представления

- четвёрки (Quadruple)
- тройки (Triples)
- косвенные тройки (Indirect Triples)

Quadruple

- структура с 4-мя полями

op: оператор

arg1: 1-й операнд

arg2: 2-й операнд

result: результат выражения

Пример Quadruple

$$a = b * - c + b * - c.$$

```
t1 = uminus c
t2 = b * t1
t3 = uminus c
t4 = b * t3
t5 = t2 + t4
a = t5
```

#	Op	Arg1	Arg2	Result
(0)	uminus	c		t1
(1)	*	t1	b	t2
(2)	uminus	c		t3
(3)	*	t3	b	t4
(4)	+	t2	t4	t5
(5)	=	t5		a

Особенности

+ легко изменить код для глобальной оптимизации
+ быстрый доступ к значению временных переменных через таблицу символов

- содержит много временных объектов
- частое создание временных переменных ухудшает производительность и увеличивает затраты по памяти

Triple

- не создает временную переменную для хранения результата операции
- вместо этого ссылается на значение нужной тройки
- структура с 3-мя полями

op: оператор

arg1: 1-й операнд

arg2: 2-й операнд

Пример Triple

$$a = b * - c + b * - c.$$

```
t1 = uminus c
t2 = b * t1
t3 = uminus c
t4 = b * t3
t5 = t2 + t4
a = t5
```

#	Op	Arg1	Arg2
(0)	uminus	c	
(1)	*	(0)	b
(2)	uminus	c	
(3)	*	(2)	b
(4)	+	(1)	(3)
(5)	=	a	(4)

Особенности

- неявные временные переменные, их трудно изменять в коде
- сложно оптимизировать, потому что оптимизация включает в себя перемещение промежуточного кода:
  - когда тройка перемещается, любая другая тройка, ссылающаяся на нее, также должна быть обновлена
  - с помощью указателя можно получить прямой доступ к записи в таблице символов

! эквивалентно и представимо в виде (Directed Acyclic Graph) DAG

Indirect Triple

- использует таблицу указателей на вычисления троек, которые выполняются отдельно и сохраняются
- структура с 3-мя полями и дополнительной таблицей с 1-м полем

statement: ссылка на тройку

Пример Indirect Triple

$$a = b * - c + b * - c.$$

```
t1 = uminus c
t2 = b * t1
t3 = uminus c
t4 = b * t3
t5 = t2 + t4
a = t5
```

#	Op	Arg1	Arg2
(14)	uminus	c	
(15)	*	(14)	b
(16)	uminus	c	
(17)	*	(16)	b
(18)	+	(15)	(17)
(19)	=	a	(18)

#	Statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

Особенности

+ полезность аналогична Quadruple, но требует меньше места
+ за счёт таблицы неявные временные переменные легче изменить в коде

Static Single Assignment

- вид промежуточного представления кода
- упрощает свойства переменных и улучшает другие алгоритмы оптимизации
- используется в JIT-компиляторе Oracle HotSpot

- каждой переменной присваивается значение ровно один раз
- каждая переменная должна быть определена перед использованием

Преобразование

- любой код можно преобразовать в форму SSA
- необходимо заменить целевую переменную каждого сегмента кода новой переменной с учетом всех вхождений целевой переменной
- для этого новая переменная именуется как целевая с индексом текущего вхождения, на котором происходит замена

Пример SSA

```
x = y - z
s = x + s
x = s + p
s = z * q
s = x * s
```

- x, y, z, s, p, q - объявленные целевые переменные
- ? - новые переменные

Применение в оптимизациях

- Constant Propagation - преобразование вычислений из runtime в compiler time:
  - a = 3 * 4 + 5; -> a = 17;
- Value Range Propagation - предварительное вычисление потенциальных диапазонов значений, что позволяет создавать предсказания ветвей
- Sparse Conditional Constant Propagation - проверка диапазонов некоторых значений, позволяющая предсказать наиболее вероятную ветвь
- Partial Redundancy Elimination - удаление дублирующих расчетов, ранее выполненных в некоторых ветках программы

Применение в оптимизациях

- Strength Reduction - замена дорогих операций менее дорогими, но эквивалентными:
  - b = a * 2; -> b = a << 1;
  - b = a / 2; -> b = a >> 1;
- Global Value Numbering - замена повторяющихся вычислений результатом
- Dead Code Elimination - удаление кода, который никак не повлияет на результаты
- Register Allocation - использование ограниченного числа регистров для вычислений

TAC | SSA

- можно использовать промежуточное представление кода в TAC и SSA одновременно:
  - как в примерах

- SSA эквивалентен Continuation-passing style (CPS)
  - оптимизация программ в CPS проще, чем на исходном языке, поэтому компиляторам легче генерировать машинный код
- уточнением TAC является A-нормальная форма CFG (ANF)
  - оказалось, что ANF достигает тех же преимуществ, что и CPS, с помощью лишь одного преобразования на уровне исходного кода