Тема 6. Генерация промежуточного кода

Фаза генерации промежуточного кода предназначена для преобразования исходной программы в промежуточное представление, которое можно рассматривать как программу для некоторой виртуальной машины. Промежуточный код должен относительно просто транслироваться в целевой код. Использование промежуточного представления имеет определенные преимущества [4]:

а) позволяет не учитывать особенности целевого языка, которые значительно усложняют трансляцию;

б) упрощается перенос на другую целевую машину, для чего потребуется только реализовать преобразование промежуточного кода в язык новой машины;

в) к промежуточному представлению можно применить машинно-независимую оптимизацию.

Наиболее распространенными формами представления промежуточной программы являются динамические структуры, представляющие ориентированный граф (в частности, синтаксическое дерево), трехадресный код (в виде троек или четверок), префиксная и постфиксная запись, байт-код Java. Достаточно подробное изложение этих форм промежуточных представлений с иллюстрациями на примерах можно найти в работах [1; 2; 3]. Ограничимся рассмотрением трехадресного кода.

6.1. Трехадресный код

Трехадресный код является линеаризованным представлением синтаксического дерева и представляет собой последовательность команд вида $x := y\ op\ z$, где x, y и z – имена, константы (кроме x) или временные переменные, генерируемые компилятором; op – некоторая операция, например, арифметическая операция или операция для работы с логическими значениями. Команды могут иметь символьные метки, представляющие индекс трехадресной команды в массиве, содержащем промежуточный код. Замена меток индексами может быть выполнена в процессе отдельного прохода либо с использованием метода обратных поправок (см. подразд. 6.5).

Элементы x, y и z для удобства представляются именами или константами. При реализации это обычно указатели на соответствующие записи таблицы символов.

Выбор множества команд является важной задачей при создании промежуточного представления. Оно должно быть достаточно богатым, чтобы позволить реализовать все операции исходного языка. Небольшое множество команд легче реализуется, однако может привести к генерации длинных последовательностей команд промежуточного представления.

Пример множества основных трехадресных команд:

1) команда присваивания вида $x := y\ op\ z$, где op – бинарная арифметическая или логическая операция;

2) команда присваивания вида $x := op\ y$, где op – унарная арифметическая или логическая операция (включая и операции преобразования типов);

3) команда копирования вида $x := y$;

4) индексированные присваивания типа $x := y[i]$ и $x[i] := y$.

5) безусловный переход **goto** L, после этой команды будет выполнена команда с меткой L;

6) условный переход типа **if** $x\ relop\ y$ **goto** L, где $relop$ – операция отношения; если отношение $x\ relop\ y$ истинно, следующей выполняется команда с меткой L, в противном случае выполняется следующая за условным переходом команда;

7) условные переходы вида **if** *x* **goto** *L* и **ifFalse** *x* **goto** *L*, в которых следующей выполняется команда с меткой *L*, если значение *x* соответственно истинно или ложно, в противном случае выполняется следующая за условным переходом команда.

Данное множество команд можно дополнить командами вызова подпрограмм и передачи им параметров, командами возврата из подпрограмм и передачи возвращаемых значений, командами работы с адресами и указателями и другими командами в зависимости от возможностей исходного языка. Примеры таких команд можно посмотреть в работах [1; 3].

Рассмотрим фрагмент программы на языке Паскаль (каждый элемент массива занимает 4 единицы памяти, диапазон индексов массива 0..100)

```
s:=0; i:=0;
while i < 100 do
   s:=s+a[i];
   i:=i+1
end
```

В результате трансляции этого фрагмента может сформироваться трехадресный код (последовательность формируемых команд полностью зависит от семантических правил, связанных с продукциями грамматики):

```
50: s:=0
51: i:=0
52: if i >= 100 goto 60
53: t1:=s
54: t2:=i*4
55: t3:=a[t2]
56: s:=t1+t3
57: t4:=i+1
58: i:=t4
59: goto 52
60:
```

В полученном фрагменте кода отсчет номеров позиций команд начинается с 50. Команда t_2:=i*4 требуется для обеспечения прямого доступа к элементу массива (каждый из элементов занимает 4 единицы памяти).

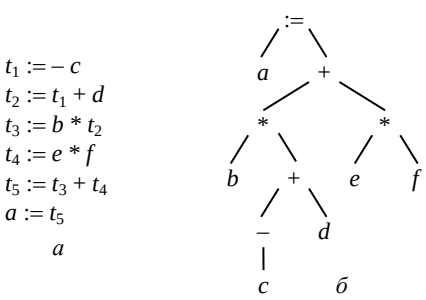
Основными структурами данных для представления трехадресных команд являются четверки (тетрады), тройки (триады) и косвенные тройки.

Четверка представляет собой запись с полями *op*, *arg*1, *arg*2 и *result* (рис. 8, *в*). Поле *op* содержит внутренний код операции. Поля *arg*1, *arg*2 и *result* обычно содержат указатели на соответствующие записи таблицы символов. Поэтому временные имена при их создании должны быть внесены в общую таблицу символов или в отдельную таблицу временных имен. Трехадресная команда $x := y + z$ представляется размещением + в *op*, y в *arg*1, z в *arg*2 и x в *result*. Унарные команды наподобие $x := -y$ или $x := y$ не используют *arg*2. Условные и безусловные переходы помещают в *result* целевую метку.

Тройка состоит из трех полей: *op*, *arg*1, *arg*2 (рис. 8, *г*). Они позволяют избежать вставки временных имен в таблицу символов, поскольку можно ссылаться на временное значение по номеру команды, которая вычисляет это значение. Поля *arg*1 и *arg*2 представляют собой либо указатели в таблицу символов (для определенных программистом имен или констант), либо указатели на тройки (для временных значений). Тернарные операции наподобие $x[i] := y$ требуют двух троек, например, можно поместить x и i в одну тройку, а y – в другую. Аналогично $x := y[i]$ можно реализовать, рассматривая эту операцию так, как если бы это были две команды, $t := y[i]$ и $x := t$, где t – временная переменная, сгенерированная компилятором. Здесь временная переменная t в действительности в тройке не появляется, поскольку обращения к временным значениям выполняются с использованием их позиций в последовательности троек.

Косвенные тройки состоят не из последовательности самих троек, а из списка указателей на тройки (рис. 8, *д*). Например, можно использовать массив *команды* для перечисления указателей на тройки в требуемом порядке.

Пример трехадресного кода и его представления для присваивания $a := b * (- c + d) + e * f$ приведен на рис. 8.



а *б*

	op	*arg*1	*arg*2	*result*
0	–	*c*		t_1
1	+	t_1	*d*	t_2
2	*	*b*	t_2	t_3
3	*	*e*	*f*	t_4
4	+	t_3	t_4	t_5
5	:=	t_5		*a*

в

	op	*arg*1	*arg*2
0	–	*c*	
1	+	(0)	*d*
2	*	*b*	(1)
3	*	*e*	*f*
4	+	(2)	(3)
5	:=	*a*	(4)

г

команды	
10	(0)
11	(1)
12	(2)
13	(3)
14	(4)
15	(5)

	op	*arg*1	*arg*2
0	–	*c*	
1	+	(0)	*d*
2	*	*b*	(1)
3	*	*e*	*f*
4	+	(2)	(3)
5	:=	*a*	(4)

д

Рис. 8. Трехадресный код и его представления
для оператора присваивания $a := b * (- c + d) + e * f$:
а – трехадресный код; *б* – синтаксическое дерево;
в – четверки; *г* – тройки; *д* – косвенные тройки

Рассмотренные представления трехадресных команд имеют свои достоинства и недостатки. При использовании четверок трехадресные команды, определяющие или использующие временные переменные, могут непосредственно обращаться к памяти для этих переменных с помощью таблицы символов. Другое преимущество четверок проявляется в оптимизирующем компиляторе, когда в процессе оптимизации приходится удалять или перемещать команды. При перемещении команды, вычисляющей *x*, команда, использующая это значение, не требует внесения каких-либо изменений. В случае же использования троек перемещение команды, определяющей временное значение, требует изменения всех ссылок на эту команду в полях *arg*1 и *arg*2. Эта проблема затрудняет использование троек в оптимизирующих компиляторах. При использовании косвенных троек такой проблемы не возникает – перемещение команд осуществляется простым переупорядочением списка команд, а сами тройки остаются неизменными.

Создание временных переменных при формировании трехадресного кода (например, при использовании четверок) обычно делается с помощью специальной функции, назовем ее для определенности *NewTemp*, которая создает новую временную переменную и возвращает указатель на соответствующую запись в таблице (в зависимости от реализации это может быть общая таблица символов или отдельная таблица временных переменных). Для оптимизирующего компилятора при каждом вызове *NewTemp* полезно создавать новую переменную, т.е. в трехадресном коде все присваивания выполняются для переменных с различными именами (так называемый принцип *статических единственных присваиваний* [1]). Для экономии памяти за счет уменьшения числа временных переменных можно использовать метод повторного их использования, учитывающий их время жизни [2], хотя он и имеет ряд ограничений и создает определенные трудности для оптимизации кода.

В следующих разделах, чтобы излишне не усложнять связанные с продукциями семантические правила, будем рассматривать только правила, связанные с генерацией промежуточного кода. При этом следует помнить, что наряду с правилами для трансляции в промежуточный код в СУО могут быть и правила для проверки типов (см. разд. 5).