УДК 519.682.1: 681.142.2

В.М. Илюшко, А.П. Собчак, К.В. Ходарев, Н.Д. Смирнов

Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ»

ПРОБЛЕМЫ СОЗДАНИЯ АППАРАТНОГО КОМПИЛЯТОРА ЭЛЕМЕНТАРНЫХ МАТЕМАТИЧЕСКИХ ФУНКЦИЙ

В данной статье затрагиваются проблемы создания аппаратного компилятора. Описано само понятие компилятора, история его создания, структура, логические части. Представлено краткое описание элементной базы, среды разработки для реализации аппаратной модели компилятора.

компилятор, транслятор, конечный автомат, информационные таблицы, ПЛИС

Введение

В настоящее время в отечественной и зарубежной практике наиболее узким местом в развитии и использовании вычислительной техники является проблема компиляции, которая решается, как правило, посредством построения программных средств, что приводит к значительным затратам ресурсов и низкому быстродействию процесса компиляции. До настоящего времени аппаратные компиляторы не применялись. Основным тормозом их развития является теория формальных грамматик, в рамках которой они не могли быть созданы в силу ориентации грамматик на программную реализацию.

1. Формулирование проблемы

Компилятором называется системная программа, выполняющая преобразование программы, написанной на одном алгоритмическом языке, в программу на языке, близком к машинному, и в определенном смысле эквивалентную первой. В случае аппаратной реализации компилятором является узел, выполняющий преобразование исходных данных в целевую исполняемую программу, сохраняя при этом эквивалентность, т.е. струк-

тура, аналогичная программному компилятору. Создание аппаратного компилятора позволит повысить быстродействие в определенных приложениях, например, компиляция исходных данных однотипных задач позволит снизить массо-габаритные и стоимостные показатели конечного изделия. Решение данной проблемы приведет к коренному изменению облика ныне существующих технологических и алгоритмических средств проектирования систем, значительно повысит эффективность использования вычислительной техники, упростит процессы общения с ЭВМ.

1.1. История создания различных компиляторов

Компиляторы составляют существенную часть программного обеспечения ЭВМ. Это связано с тем, что языки высокого уровня стали основным средством разработки программ. Только очень незначительная часть программного обеспечения, требующая особой эффективности, программируется с помощью ассемблеров. В настоящее время распространено довольно много языков программирования. Наряду с традиционными языками, такими, как Фортран, широкое распространение получили так называемые "универсальные языки" (Паскаль, Си, Модула-2, Ада и другие), а также некоторые специализированные (например, язык обработки списочных структур Лисп). Кроме того, большое распространение получили языки, связанные с узкими предметными областями, такие, как входные языки пакетов прикладных программ. Для некоторых языков имеется довольно много реализаций. Например, реализаций Паскаля, Модулы-2 или Си для ЭВМ типа IBM/PC на рынке десятки.

С другой стороны, постоянно растущая потребность в новых компиляторах связана с бурным развитием архитектур ЭВМ. Это развитие идет по различным направлениям. Совершенствуются старые архитектуры, как в концептуальном отношении, так и по отдельным, конкретным линиям. Это можно проиллюстрировать на примере микропроцессора Intel-80X86. Последовательные версии этого микропроцессора 8086, 80186, 80286, 80386, 80486, 80586 отличаются не только техническими характеристиками, но и, что более важно, новыми возможностями и, значит, изменением (расширением) системы команд. Естественно, это требует новых компиля-

торов (или модификации старых). То же можно сказать о микропроцессорах Motorola 68010, 68020, 68030, 68040. В рамках традиционных последовательных машин возникает большое число различных направлений архитектур. Примерами могут служить архитектуры CISC, RISC. Такие ведущие фирмы как Intel, Motorola, Sun, DEC начинают переходить на выпуск машин с RISC-архитектурами. Естественно, для каждой новой системы команд требуется полный набор новых компиляторов с распространенных языков. Наконец, бурно развиваются различные параллельные архитектуры. Среди них отметим векторные, многопроцессорные, с широким командным словом (вариантом которых являются суперскалярные ЭВМ). Естественно, для каждой из машин создаются новые компиляторы для многих языков программирования. Здесь необходимо также отметить, что новые архитектуры требуют разработки совершенно новых подходов к созданию компиляторов, так что наряду с собственно разработкой компиляторов ведется и большая научная работа по созданию новых методов трансляции [1].

1.2. Фазовая структура компилятора

Концептуально компилятор работает пофазно, причем в процессе каждой фазы происходит преобразование исходной программы из одного представления в другое [2]. Типичное разбиение компилятора на фазы показано на рис. 1.

Одной из важных функций компилятора является запись используемых в исходной программе идентификаторов и сбор информации о различных атрибутах каждого идентификатора. Эти атрибуты представляют сведения об отведенной идентификатору памяти, его типе, области видимости (где в программе он может применяться). При использовании имен процедур атрибуты говорят о количестве и типе их аргументов, методе передачи каждого аргумента (например, о ссылке) и типе возвращаемого значения, если таковое имеется.

Таблица символов представляет собой структуру данных, содержащую записи о каждом идентификаторе с полями для его атрибутов. Данная

структура позволяет быстро найти информацию о любом идентификаторе и внести необходимые изменения.

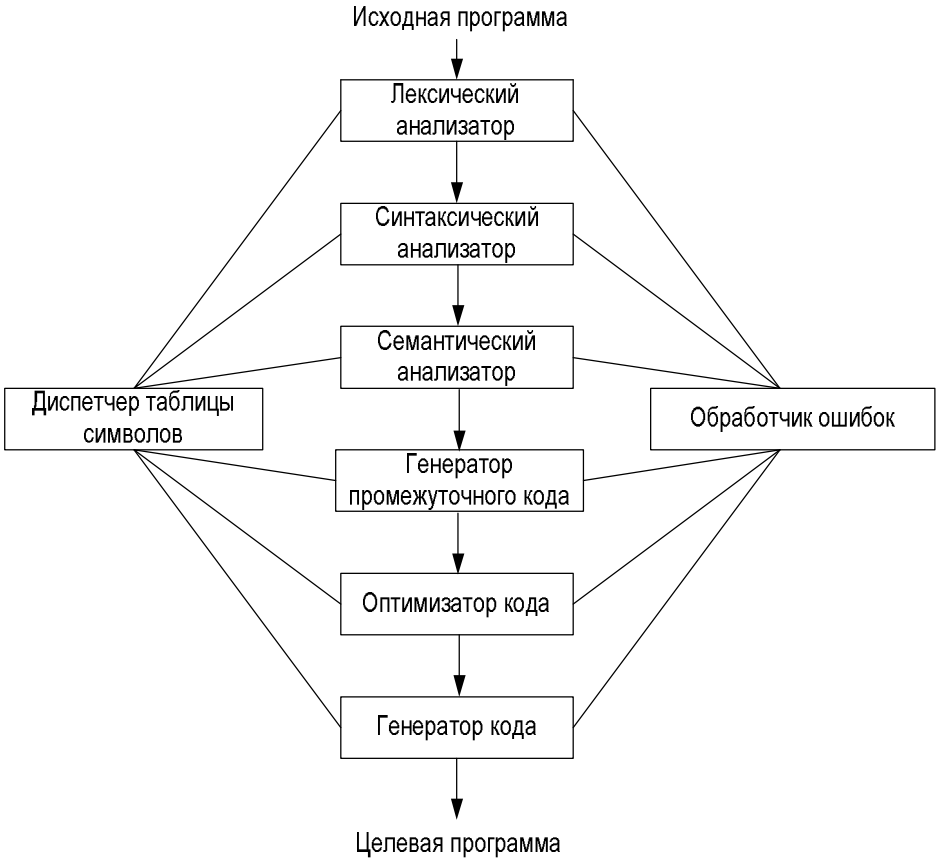

Рис. 1. Фазы компилятора

Конечно, те или иные фазы транслятора могут либо отсутствовать совсем, либо объединяться. В простейшем случае однопроходного транслятора нет явной фазы генерации промежуточного представления и оптимизации, остальные фазы объединены в одну, причем нет и явно построенного синтаксического дерева.

Это классическое разбиение компилятора на фазы. Так работает большинство современных компиляторов, реализующих те или иные языки программирования высокого уровня. Все, что способен выполнять компилятор, построенный по такой схеме – это преобразовать исходную про-

грамму в целевую для ее последующего исполнения конкретным процессором. Он подобен конечному автомату, имеющему конечное множество состояний Q, конечное множество допустимых входных символов T и функцию переходов, отображающую множество $Q \times T$ во множество подмножеств множества Q и определяющую поведение управляющего устройства [3].

Для увеличения количества корректно преобразуемых им лексем, в его структуру включаются информационные таблицы. При анализе программы из описаний, заголовков процедур, заголовков циклов и т.д. извлекается информация и сохраняется для последующего использования. Эта информация обнаруживается в отдельных точках программы и организуется так, чтобы к ней можно было обратиться из любой части компилятора.

В каждом компиляторе в той или иной форме используется таблица символов (иногда ее называют списком идентификаторов или таблицей имен). Это таблица идентификаторов, встречающихся в программе, вместе с их атрибутами. К атрибутам относятся: тип идентификатора, его адрес в объектной программе или любая другая информация о нем, которая может понадобиться при генерации объектной программы или при составлении и интерпретации внутреннего представления программы.

Информационные таблицы, по сути, представляют собой расширение понятия таблицы символов. В информационные таблицы компилятор может заносить не только информацию о встречающихся в программе идентификаторах, но и правила, сформированные им в ходе выполнения процесса компиляции с целью их последующей проверки и применения.

Конечно, для начала процесса обучения компилятору необходимо иметь в распоряжении некоторый набор начальных правил, на основании которых он затем может формировать новые. Создание этих правил, а также минимизация их количества является важнейшей (а также самой сложной) задачей при построении такого рода систем.

Текст результирующей программы должен быть доступен для анализа в каком-либо его блоке, например для анализа ошибок компиляции и добавления нового правила в информационные таблицы или изменения существующих (но не тех, на которых основывается его обучение).

1.3. Схема работы компилятора

Сначала исходная программа разлагается на составные части, затем из них строятся части эквивалентной объектной программы. Для этого на этапе анализа компилятор строит несколько таблиц, которые используются затем как при анализе, так и при синтезе. На рис. 2 весь процесс показан более подробно. Пунктирные стрелки изображают информационные потоки, тогда как сплошные стрелки указывают порядок работы программ.

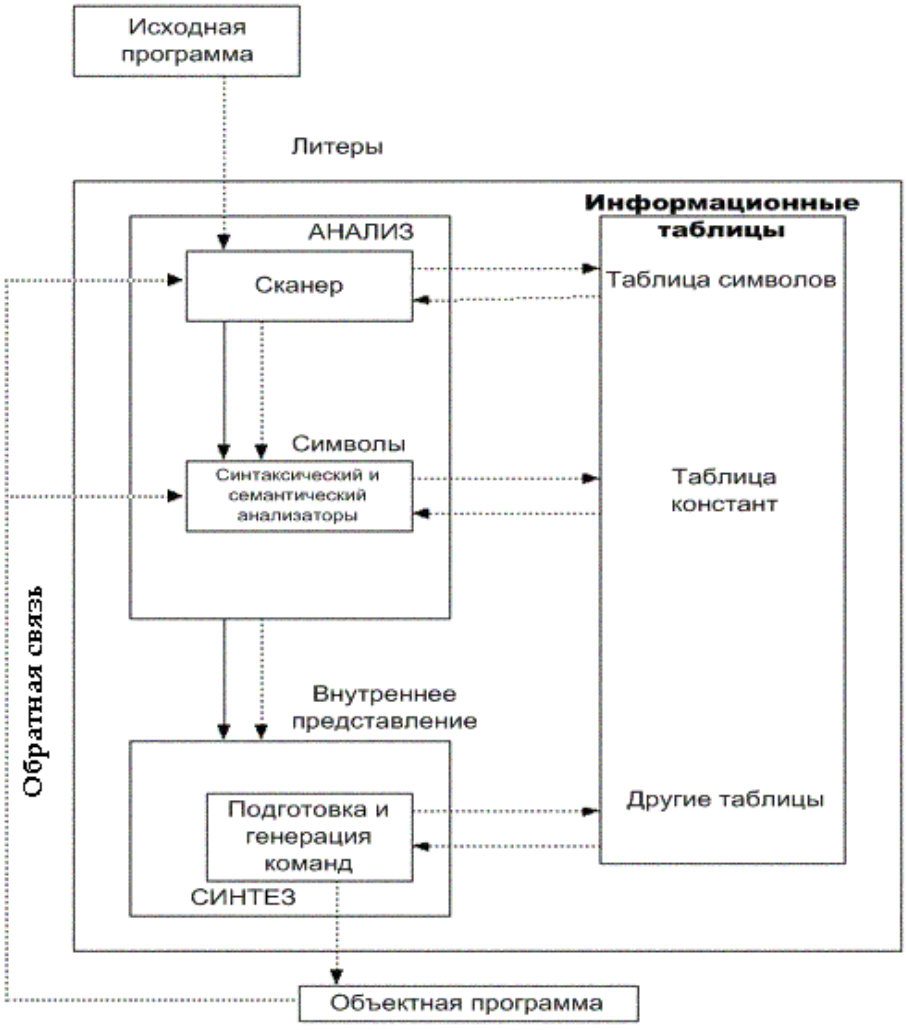

Рис. 2. Логические части компилятора

Сканер. Сканер – самая простая часть компилятора, также называемая лексическим анализатором. Сканер просматривает литеры исходной программы

Символы передаются затем на обработку синтаксическому анализатору. Сканеру можно поручить задачу удаления комментариев, простую макрогенерацию, занесение идентификаторов в таблицу символов и другую простую работу, которая не требует анализа исходной программы.

Синтаксический и семантический анализаторы. Анализаторы выполняют действительно сложную работу по расчленению исходной программы на составные части, формированию ее внутреннего представления и занесению информации в таблицу символов и другие таблицы. При этом также выполняется полный синтаксический и семантический контроль программы.

Обычный анализатор представляет собой синтаксически управляемую программу. В действительности стремятся отделить синтаксис от семантики настолько, насколько это возможно. Когда синтаксический анализатор узнает конструкцию исходного языка, он вызывает соответствующую семантическую процедуру или семантическую программу, которая контролирует данную конструкцию с точки зрения семантики и затем запоминает информацию о ней во внутреннем представлении программы и, если надо, в таблице символов.

Внутреннее представление исходной программы в значительной степени зависит от его дальнейшего использования. Это может быть дерево, содержащее синтаксис исходной программы. Часто используется еще одна форма — список тетрад (оператор, результат, операнд, операнд) в порядке их выполнения. Например, присваивание "$a = b + c \times d$" будет представлено так:

$\times$, $T1$, c, d
$+$, $T2$, b, $T1$
$=$, a, $T2$,

где $T1$ и $T2$ – временные переменные, образованные соответствующими семантическими процедурами. Операндами в приведенном примере будут не сами символические имена, а указатели на те элементы в таблице символов, в которых описаны эти операнды.

Подготовка к генерации команд или интерпретации. Перед генерацией команд обычно необходимо некоторым образом обработать и изменить внутреннюю программу. Кроме того, должна быть выделена память под переменные готовой программы. Одним из важных моментов на этом этапе является оптимизация программы с целью уменьшения времени ее работы, оптимального использования памяти и регистров целевой машины.

Генерация команд или интерпретация. По существу, на этом этапе происходит перевод внутреннего представления исходной программы на машинный язык. Это, по-видимому, наиболее кропотливая часть работы при создании компилятора, хотя и наиболее понятная. Предположим, что внутреннее представление имеет вид тетрад, как это описано выше, и мы генерируем команды для каждой тетрады по порядку на языке ассемблера для процессора семейства Intelx86. Для приведенных выше тетрад можно сгенерировать следующие команды:

mov *AX*, *c* ; загрузить содержимое *C* в регистр AX
mul *d* ; результат умножения в паре *DX*:*AX*
add *AX*, *b* ; прибавить к результату содержимое *B*
mov *a*, *AX* ; запомнить результат в *a*.

Здесь имена *a*, *b*, *c*, *d* фактически являются адресами слов в памяти, содержащими фактические значения переменных *a*, *b*, *c*, *d*. В интерпретаторе эта часть компилятора заменяется программой, которая выполняет (или интерпретирует) внутреннее представление исходной программы.

На рис. 2 показаны логические связи между отдельными частями компилятора. Все четыре логически последовательных процесса: сканирование, анализ, подготовку к генерации и генерацию команд (или интерпретацию) можно организовать в реально работающем компиляторе разными способами в зависимости от количества проходов и возможностей языка, на котором написан сам компилятор.

2. Реализация компилятора

Физически компилятор может быть реализован в виде программы (программный компилятор), работающей под управлением какого-либо процессора, либо в виде отдельного электронного устройства, реализующего

все вышеописанные этапы компиляции. Современная элементная база позволяет реализовать все основные блоки на одной микросхеме, например, на программируемых логических интегральных схемах (ПЛИС), в частности, фирмы Altera.

Для аппаратной реализации научных проектов применяются языки описания аппаратуры VHDL и AHDL.

Язык AHDL разработан фирмой Altera и предназначен для описания комбинационных и последовательностных логических устройств, групповых операций, цифровых автоматов (state machine) и таблиц истинности с учетом архитектурных особенностей программируемых логических интегральных схем (ПЛИС) фирмы Altera. Он полностью интегрируется с системой автоматизированного проектирования ПЛИС MAX+PLUS II. Файлы описания аппаратуры, написанные на языке AHDL, имеют расширение *.TDF (Text design file), а файлы на языке VHDL имеют расширение *.VHD [4].

Примером аппаратного компилятора может служить модель компилятора элементарных математических функций. Структура компилятора в аппаратной реализации не имеет существенных различий от структуры программного компилятора. Различия появляются во внутренней структуре каждого из блоков в соответствии со спецификой языков AHDL и VHDL, а также принимая во внимание то, что мы имеем дело с реальными сигналами, а не с потоком битов.

Алгоритм действия лексического анализатора основан на сравнении эталонных значений со значениями компилируемой строки. Устройство содержит два вида памяти: ПЗУ и ОЗУ.

В ПЗУ заносятся эталонные значения, т.е. значения, из которых может состоять компилируемая строка. В ОЗУ формируется эта строка, которую можно изменять.

Синтаксический анализатор выполняет работу по расчленению исходной программы на составные части, формированию ее внутреннего представления и занесению информации в таблицу символов и другие таблицы. К примеру, для числовой последовательности его работа основана на том, что после числа в математической последовательности могут следовать либо знак, либо закрывающая скобка; после знака – открывающая

скобка либо число; после открывающей скобки – только число и после закрывающей скобки может следовать только знак действия.

Генератор целевого кода генерирует целевой код, который, как правило, является машинным кодом конкретного процессора либо семейства, используя для этого данные, предоставляемые ему генератором промежуточного кода [5, 6].

Заключение

В данной статье рассмотрены известные вопросы, касающиеся программных компиляторов, их структуры, истории, а также целесообразность и проблемы создания аппаратных компиляторов. Залогом их успешной реализации служат достижения в технологии проектирования БИС с высокой степенью интеграции.

Литература

1. Донован Дж. Системное программирование. – М.: Мир, 1975. – 540 с.

2. Альфред А., Равви С., Джеффри У. Компиляторы. Принципы, технологии, инструменты. – К., 2001. – 767 с.

3. Жихарев В.Я., Илюшко В.М., Чумаченко И.В. Проектирование электронных компиляторов. – Х.: Факт, 1999. – 86 с.

4. Соловьев В.В. Проектирование цифровых систем на основе ПЛИС // К.: Горячая линия – Телеком, 2001. – 265 с.

5. Шалыто А.А. Методы аппаратной и программной реализации алгоритмов. – С.-Пб.: Наука, 2000. –749 с.

6. Собчак А.П., Марченко А.Н., Ходарев К.В. Реализация лексического анализатора на языке аппаратного описания. – Х.: Нац. аэрокосм. ун-т «ХАИ», 2004. – 122 с.

Поступила в редакцию 6.04.2005