## Разработка в структурно логической схемы МП

В схеме микропроцессора должны быть включены все устройства, необходимы для приема из памяти, хранения, и выполнение команд заданными согласно варианта режима адресации. В соответствии с требованием полноты микропроцессора, обычно включают следующие устройства.

ОП64

ШД

PC

АЛУ

Адресная шина

Адресная шина должна иметь разрядность, достаточную для передачи адреса заданной длины.

Шина данных. Разрядность шины данных обычно совпадают с длиной байт. В случаи, когда все команды имеют одинаковую длину, она совпадает с основным словом.

Программный счетчик – содержит адрес текущей выполняемой программы. Автоматически увеличивает длину команд, при переходе на следующую.

Регистр команд (IR) – предназначен для приема из памяти и хранении кода команд.

Дешифратор –определяет что это за команда.

Блок управления и синхронизация – выполняет управлением микропроцессором.

DAK – регистр данных и адреса.

Роны – регистры общего назначения.

## Режимы адресации

Адрес исполнительный (адрес ячейки памяти или регистра) с которым работает команда, можно указывать различными способами, руководствуюсь следующими соображениями.

Исполнение команд с более коротким адресом.

Обеспечение простого доступа к возможно большему объему памяти.

Возможность изменения содержимого адресной части без изменения команды.

Обеспечение более быстрой адресации.

Использования более простого метода адресации во избегания ошибок.

Прямая адресация ADD B – в прямой адресации адрес исполнительный является частью команды. Длина команды зависит от адреса.

Прямая регистровая – место адреса исполнительного хранится номер регистра.

Косвенная адресация – адрес исполнительный находится по адресу, указанному в команде.

Косвенно регистровая – адрес исполнительный находится по адресу регистра.

Непосредственная адресация – обрабатываемые данные являются частью команды.

Относительная – Адрес исполнительный формируется путем сложения программного счетчика и смещением, которое находится в команде.

Индексная адресация – адрес исполнительный образуется путем сложения индексного регистра с адресом, который находится в команде.

Базовая адресация – используется для организации перемещения программы в памяти. Для этого пишется псевдо команда, которая закрепляет один из регистров в качестве базового. В этот регистр записывается адрес загрузки и все исполнительные адреса записываются.

Адресация с авто увеличением или авто уменьшением – применяется для организации циклов. Перед выполнением команды или после, содержимое регистров, в котором находился операнд увеличивается или уменьшается на единицу.

Стековая – адресом данных является содержимое указателя стеков.

Система команд микро процессора.

Команды микропроцессора в памяти занимают от 1 байта до максимальной длены.

1 байт команды содержит код операции. В некоторых код операций может размещаться и во втором байте. Требование к системе команд это

функциональная полнота – система команд должна предоставлять максимально удобные средства для программирования, по составлениям алгоритма, в условиях ограниченных ресурсов.

Минимальность – количество длина команд должна быть минимальной.

Система команд микропроцессора обычно содержит следующие операции:

Операции пересылки обеспечиваю передачу операнда источника к операнду приемника, без содержательного их преобразователя.

Арифметические операции – сложение вычитание, умножение, деление.

Логические – и, или, не.

Сдвиг.

Условные переходы.

Изменение содержимого регистра кода условий (флажки).

Команды вода, вывода и останова.

Проектирование системы команд значительной степени зависит от заданных режимах адресаций.

При выборе формата команд следует учитывать следующие особенности:

1 В коротких командах часто используется переменная длина кода операций. Это позволяет плотнее упаковывать информацию и сократить число форматов команд и их длину.

3 байта

1 байт

1 байт

3 байта

Максимальное количество команд 256.

В команда с индексной адресацией нужно обеспечить возможно большую величину смещения, не превышающую оперативную память

|  |  |
| --- | --- |
|  |  |
| RI 0 | 00 |
| RI 1 | 01 |
| RI 2 | 10 |

Команда с непосредственной адресацией – длина непосредственного операнда минимум должна быть 1 байт.

Команды должны быть унифицированы место положения первого и второго операнда. Первый операнд источник, второй приемник.

Некоторые команды могут только составлять код операций.

Смещение.

## Описания языка Ассемблера (28\_09\_07)

Ассемблер – машинно-ориентированный язык, операторы которые транслируются один к одному, либо один к нулю (псевдо операции, передают информацию загрузчику, линковщику). Ассемблер наиболее эффективно используют ресурсы ЭВМ. В общим виде операторы ассемблера выглядят так: метка, код операции, [операнды], [комментарий], или [признак комментария].

Существуют различные типы ассемблера:

Резидентный ассемблер – программа, которая хранится в ПЗУ и к ней имеется постоянная доступ. Кросс ассемблер – выполняется на ЭВМ общего назначения, и написана на языке программирования этого ЭВМ. Достоинства – разработка программы предшествует технической реализации. Используется мощная программное обеспечение ЭМВ. Выразительность языка и диагностика, обеспечивается преимущества машины выразительности языка и диагностики.

Существую абсолютные и перемещаемые ассемблеры. В абсолютном ассемблере адрес загрузки определяется псевдокомандой. Адрес загрузки известен во время ассемблирования, поэтому при построении адресных константы команд выполняется в абсолютных адресах. В перемещаемом ассемблере возможность определить адрес загрузки происходит не вовремя ассемблирования а во время загрузки программы в память. Для этого относительный адреса адресных констант команд увеличивается на адрес загрузки.

Однопроходные и двухпроходные – формированию объектной программы за один просмотр препятствует ссылки в перед, в командах передачи управления. Ассемблер не может сразу определить адрес перехода. Существует два варианта построения однопросмотровых ассемблеров:

Неопределенная метка записывается в таблицу меток. По мере нахождения меток и определения адреса этой метки, в зарезервированные места записываются адрес. Ссылок на одну и ту же метку может быть много.

Ассемблер строит дополнительную таблицу переходов. Все команды обращения в перед осуществляют передачу косвенно, через эту таблицу.

Достоинством однопросмотрового ассемблера является выигрыш в скорости ассемблирования. Тем не менее большинство ассемблеров современных ЭВМ двухпросмотровые. Целью первого просмотра является определения адреса для каждой команды в программе и формирования символических имен или меток. Целью второго прохода является генерация машины команды констант, обработка псевдо команд а так же распечатка листинга.

## Основные функции Ассемблера

Синтаксический анализ исходного текста программы.

Распределения памяти (назначения относительных адресов) для команд, констант, переменных и литералов.

Трансляции исходной программы в объектный код.

Формирования информации для загрузчика, линковщика.

Формирования листинга программы.

Основные базы данных для формирования просмотра.

Файл исходного текста

Значение программного счетчика.

Таблица машинных операций. Длина команды. Формат команды.

Таблица псевдоопераций.

Таблица имен (меток).

Таблица литералов.

Таблица внешних имен.

Таблица входов.

Лекция (5\_10\_07)

Машино зависимые части ассемблера:

Форматы данных и форматы команд

Режимы адресации или способы адресации

Перемещение.

Машино не зависимые характеристики ассемблера. Синтаксис языка довольно прост. При написании команд используются приемы которые облегчают написание программы.

Косвенная адресация – для реализации косвенной адресации используется прием, который называется префикс.

Непосредственного операнда

Для литералов – позволяет отказаться от использования отдельного предложения для определения константы и соответственно его имя. Таблица литералов записывается после тела программы.

Средства определение имен и задание значения.

Выражение. В большинстве ассемблеров на ряду с одиночными терминами разрешают использовать выражения. Каждое такое выражение вычисляется ассемблером во время трансляции, а за тем полученное значение используется в виде адреса или непосредственного операнда.

Программные блоки – многие ассемблеры предоставляют гибкие средства обработки исходных программ. Они позволяют располагать сгенерированные машинные командные данные в порядке отличного от исходного. Создавать независимые части управляющие секции. Каждая из этих частей сохраняет свою индивидуальность и обрабатывается загрузчиком отдельно.

Объектная программа

В заключительной фазе своей работе, ассемблер должен записать полученный объектный код на некоторое устройства вывода. После этого объектная программа должна помещается в оперативную память. Для реализации этого используются 3 основные процесса:

Загрузка – обеспечивает размещение в оперативной памяти для исполнения. Это системная программа и традиционная система.

Перемещение – позволяет модифицировать объектную программу так, что она может загружаться с адреса, отличного от первоначального.

Связывание – обеспечивает объединение двух и более раздельно транслированных объектных программ и предоставляет информацию необходимую для разрешения ссылок между ними. Для того чтобы программа была перемещаемая и мы могли разрешить все внешние ссылки, в объектной программе должна быть заложена информация.

Формат объектной программы.

Запись заголовок.

- // - 2-7 – имя программы

- // - 8-13 – начала адреса программы

- // - 14-19 – длина программы в байтах

Тело. Признак Т.

- // - 2-7 – Начальный адрес в данной записи.

- // - 8-9 – длина текущей записи

- // - 10-69 – объектный код

Запись - конец

- // - 2-7 – адрес первой исполняемой команды

Если большинство команд ассемблера использую относительную и непосредственную адресацию, то для реализации перемещения программ модификации требует отдельные программы ассемблера. Запись в модификатор может быть использованным как для перемещения так и для связок. Он состоит из:

Признак М

- // - 2-7 – начальный адрес модифицированного программного объекта

- // - 8-9 – длина модифицированного поля в полубайтах

- // - 10 – признак модификации

- // - 11 - 16 – внешнее имя

Модификатор выделяет все адреса, которые надо пересчитать с учетом адреса загрузки. Однако такая схема годится не для всех машин, если большинство команд используют прямую адресацию и фиксированный командный формат, следовательно должны модифицироваться для перемещения что потребует большого увеличения объема памяти объектной программы.

## Лекция (12\_10\_07)

2 вариант – использование маски. С каждым словом программы объектного кода, связывается в разряд перемещения. Эти разряды образую вместе маску, которая записывается сразу после длины тела программы в каждой записи.

3 способ организации перемещения. Во многих ЭВМ перемещения организуется аппаратным образом. Для этого используется базовая адресация – базовый адрес + смещение.

4 вариант. Любой адрес высчитывается относительно.

Управляющая секция – часть программы, после ассемблирования сохраняет свою индивидуальность, и может перемещаться и загружаться независимо от других. Так как эти части логически связанны и не могут существовать отдельно, должен быть реализован механизм связывания и объединения. Чаще всего управляющей секцией выступает процедура функция. Команды одной управляющей секции должны иметь возможность ссылаться на команды и на области данных, расположенных в других секциях. Ссылки нельзя обрабатывать обычным образом, поскольку УС загружается и перемещается независимо дуг от друга и ассемблер ничего не известно. Для каждой внешней ссылке ассемблер генерирует информацию, которая дает возможность выполнить требуемое связывание программы. Имена определенные в одной управляющей секции могут непосредственно использоваться в другой секции. Для того чтобы загрузчик смог их обработать они должны быть описаны как внешние имена.

Запись определения – Столбец 1 D

- // - 2-7 – идентификатор внешнего имени, определенной данной управляющей секцией.

- // - 8-13 – относительный адрес имени

Запись ссылка – столбец 1 R

- // - 2-7 – идентификатор внешнего имени

Пример иллюстрирующий управляющие секции и связывающие программы.

Copy start 0

Варианты построения загрузчиков –

Абсолютный загрузчик – записывает объектную программу в оперативную память и передает управления на адрес ее исполнения.

Связывающий загрузчик – важной информацией для связывающего загрузчика являются объектные файлы, которые связаны друг с другом при помощи внешних ссылок. Связывание не может быть выполнено пока не будут назначены адреса для всех требуемых имен. Поэтому связывающий загрузчик выполняет два просмотра входного просмотра. Во время первого назначаются адреса всех внешних ссылок. Во время второго выполняется загрузка перемещения связывание.

## Лекция (19\_10\_07)

Загрузчики, которые используют редактор связи.

Редактор связи используется для языков высокого уровня.

Динамические загрузчики

Связывание откладывается до момента исполнения программы. В этом случаи загрузка и связывание программ осуществляется тогда, когда происходит первое обращение. Используется в интерактивных диалоговых системах.

Раскручивающий загрузчик.

Применяется метод рекурсии, когда одна программа выполняет свою функцию и загружает следующую.

Макропроцессор

По мере программирования на языке ассемблер, некоторые части программ приходится повторять и переписывать снова и снова. Одним из приемов записи часто используемых групп предложений используемого языка, является применения макро определений и макро инструкций. Программист записывает компактный вариант своей программы, а макропроцессор формирует окончательный вариант текста. Процесс замены макро команд соответствующими группами предложений осуществляемым макро процессором называется макро расширением, или макро генерацией.

Основные функции макропроцесса.

Основной функцией – замена одних групп предложений или символов на другие. Вопросы трансляции их в коды машины не имеют непосредственного отношения к процессу макро генерации. Поэтому механизм работы макропроцессора практически не связан со структурой машин. Любой процессор макро команд должен решать следующие 4 основные задачи.

Распознавание макро определения. Задача усложняется если в нутрии могут быть использованы другие макро определения.

Запоминания макро определения.

Распознать макро вызовы, представлены в формате мнемонического кода операции. Это предполагает, что имена команд обрабатываются как один из типов кода операций.

Выполнить расширения макро команд, постановку, и подстановку фактических параметров.

После обработки макро определений расширенный файл может быть использован в качестве входного файла для ассемблера.

## Варианты построения макропроцессоров

Однопросмотровый

Для того что бы реализовать однопросмотровый, макро определения должны находится до макро команд. На первом этапе происходит распознавание макро процесса и запоминания. Второй проход это уже макро генерация.

Таблица фактических параметров.

Макро процессоры общего назначения.

Они не зависят от конкретного языка программирования.

Достоинства: могут быть использованы для различных языков. Минимум времени для изучения.

Раскручивающие макропроцессоры.

Допускает ложность, и требует столько проходов, сколько необходимо для того чтобы получить исходный текст.

Процесс компиляции.

Языки высокого уровня появились в конце 50 начало 60 годов, такие как Фортран, Алгол. Позже Паскаль, Си. Программа которая транслирует любую программу написанную на языке высокого уровня в эквивалентную программу на другом языке (обычно это машинный код) называют компилятором. Два компилятора, реализующие один и тот же язык, на одной и той же машине, могут отличатся хотя бы потому, что разработчики преследовали различные цели при их создании.

Цель:

1. получения эффективного объектного кода программы.

2. Минимизация времени компилирования программы.

3. Разработка небольших объектных программ.

4. Разработка компилятора минимального размера.

5. Создания компилятора обладающий широкими возможностями обнаружения и исправления ошибки.

6. Обеспечение надежности компилятора.

Существует и другой подход, который называется интерпретация, представителем является Basic. Интерпретация состоит в том, что вместо трансляции в машинный код и последующее выполнение программа сначала по операторно транслируется в промежуточный язык, а затем транслируется и выполняется каждый оператор промежуточного языка.

Передать сообщение об ошибках пользователю, часто бывает легче в терминах исходной программы.

Версия программы на языке не редко оказывается компактней, чем машинный код, выдаваемый компилятор.

Изменение части программы не требует перекомпиляции всей программы.

Недостатки:

Медленность работы.

Общая схема обработки языков высокого уровня

Скелет исходная программа

Предпроцессор

макропроцессор

компилятор

Целевая ассемблер

программа

Редактор связей

транслятор

Перемещаемый машинный код

Абсолютный машинный код

Общая структура компилятора

Исходная программа ЯВУ

Лексический

анализатор

Стандартная форма

Таблицы символов

Синтаксический

анализатор

Семантический анализатор

Оптимизация кода

генерация кода и распределения памяти

## Лекция (1\_11\_07)

Лексический анализатор – представляет собой первую фазу компилятора. Его основная задача состоит в чтении новых символов и выдачи последовательности лексем. Замена объектов переменной длины символами фиксированной длины. С теоретической точки зрения лексический анализатор не является обязательной частью компилятора. Все его функции могут выполнятся на этапе синтаксического разбора, поскольку полностью регламентированы синтаксисом входного языка. Однако существует несколько причин по которым состав практически всех компиляторов включает лексический анализ.

Применения лексического анализатора упрощает работу с текстом программы, за счет удаления комментариев и пробелов, то есть символов не несущих смысловой нагрузки.

Увеличения эффективности компилятора. Поскольку на чтение исходной программы и разбор ее на лексемы тратится много времени, специальные технологии утилизации и обработки лексем могут существенно повысить производительность компилятора.

Увеличения переносимости компилятора.

Лексема – минимальная единица языка. Исходная программа на ЯВУ представляет собой цепочку образуя последовательность всех листов программы. Язык паскаль использует алфавит, который использует подмножество наборов символа таблицы кодов.

Специальные символы.

Управляющие символы. Некоторые символы расширенного набора кодов включают буквы и кириллицы. Они используются в строках, но не применяются для построения лексем. Среди допустимых для всех языков символов разделителей, благодаря предложения исходной программы разделяются на отдельные слова. В паскале можно выделить несколько категорий лексем:

## Минимальная значимая единица текста программ

Идентификаторы – последовательность букв и цифр. Стандартными, пред определенными идентификаторами в языке паскаль является: имена встроенных процедур и функций.

Метки – числовые и символьные и отделяются двоеточием.

Числа. Число различают целое - десятичное, вещественное,

Строки – последовательность символов из расширенного набора кодов, заключенная в кавычках.

Синтаксический анализатор

Проверяет является ли программа грамматически правильной, иначе говоря удовлетворяет ли она законам языка программирования, на котором она написана. Синтаксис языка затрагивает только форму языка. Если предложения удовлетворяет нормальным правилам, оно не зависимо от его значения рассматривается как синтаксически правильное.

Формальное определения языков программирования

Под формальным определением языка программирования мы понимает полное описание синтаксиса и семантики. Желательно иметь такое описание.

Сведения о языке содержится в учебниках и руководствах. Часто эти описания не однозначные и не освещают всех тонкостей.

Формальное описание надо для разработчиков компилятора. Синтаксическое определение может задаваться формальными или не формальными способами.

Метаязык (металингвистические символы)

Формальное описания языка.

Метаязык

Использую синтаксические диаграммы

Скобочные конструкции

С помощью множеств

Метаязык

Описание любого формального языка описывается на МЕТО языке. Он может описывать синтез, либо семантику (смысл конструкций), либо все вместе. Для языков программирования наиболее распространенным МЕТО языком для описание синтеза служит нормальная форма БНФ. Перечислим основные понятия и конструкции этого языка:

Терминальный символ – символ, состоящей только из букв алфавита описываемого языка. Одна или несколько букв.

Не терминальный символ – сформулированная на русском или другом языке понятие описываемого языка программирования. Металингвистические переменные. Для того чтобы раскрыть понятие языка обозначаемыми не терминальными символами, используются правила подстановки. U и u – произвольные конечные последовательные цепочки терминальных символов. Знак:: = есть по определению или представляет собой. При описании языков программирования U - это один не терминальный символ. u – любая последовательность терминальных или не терминальных символов раскрывающая сущность не терминального символа с лева.

Символическое имя >:: =<буква> | <символическое имя> <бц>

<бц>:: =<буква> <цифра>

По одном из правил определяющие наиболее общих понятий языка строится первым и называется начальным символом языка.

Классификация языков по Хомскому

В основе этой классификации лежит форма левой и правой части правил подстановки. Языки делятся на 4 класса:

0

1

2

3

При чем каждый класс большего номера, является подмножеством каждого класса с меньшим номером.

Класс 0. (Грамматика с фразовой структурой). Не накладывается ни каких ограничений на правила подстановки. Правило имеет вид приведенный выше, где U – произвольная не пустая последовательность терминальных и не терминальных символов. Класс 0 является наиболее мощным языки этого класса могут служить моделью естественных языков.

Класс 1 Контекстно-зависимая грамматика. U1 – нетерминальный символ. X Y u – произвольная цепочка терминальных и нетерминальных символов. Смысл этого правила состоит в том, что замена U на u осуществляется только в контексте X Y. Если длину обозначить, то видна что левая часть всегда меньше чем правая.

Класс 2 Контекстно-свободные грамматики. U ровно один не терминал. Грамматика класса 2 обычно используется для описания синтаксиса языков программирования.

Класс 3 Регулярной грамматикой. U – один не терминал. t – один терминал. n – один не терминал. Грамматика три может использоваться для описания символа простых языков. Используется для сборки лексем. Если б хотя бы одно правило подстановки относится к более высокому классу чем остальные, то и вся грамматика относится к этому классу. Для описания синтаксиса формального языка достаточно задать грамматику с помощью 4 объектов.

S→aS

S→a

S→b

S→bY

Y→bY

Y→b

S →ξ

G3=({a,b},{S,Y},P3,S)

Две грамматики генерирующие один и тот же язык называются эквивалентные грамматики.

Каждая строка, которую можно вывести из начального символа называется сентенциальный символ.

Синтаксические диаграммы.

Другой распространенный способ описания синтаксиса языка является графическим изображениям форм Бекуса Наура. Не терминальные символы записываются на диаграмме прямоугольниках, а терминальные в кружках или овалах.

Пример определения символического имени.

0

1

Синтаксический анализ языков программирования.

После того, как на этапе лексического анализа, программа разбивается на ее основные элементы, следующая фаза компилятора должна распознать структуру выражения, состоящая из этих элементов и интерпретировать их. Синтаксический анализ, представляет собой задачу противоположную задачи порождения (вывода). Задача разбора формулируется следующим образом: определить соответствует ли данная конструкция некоторого языка, грамматике этого языка. Является ли данная конструкция правильным предложением языка, то есть не содержит синтаксических ошибок. Различают два типа разбора. Левосторонний и правосторонний.

Левосторонний разбор – на каждом этапе вывода, начиная с первого начального символа языка замещается с помощью одного из порождающих правил грамматики самый левый не терминальный символ в сентенциальной форме.

Если сравнить два вывода, то можно выделить в правостороннем обратный порядок порождающих правил. Так как правосторонний разбор обычно ассоциируется с приведением предложения к начальному символу, а не с генерацией изначального символа.

Синтаксическое дерево разбора

Вывод можно описать и в терминах построения дерева разбора. Дерево представляет собой иерархическую структуру, корень дерева – начальный символ грамматики, узлы промежуточные обычно не терминальные символы, а все остальные узлы не терминальные символы. В большинстве случаев лево и правосторонний разбор и синтаксическое дерево является уникальным. Однако, существуют грамматики, которые имеют более одного дерево разбора, такие грамматики называются не однозначными. Установить неоднозначность является не разрешимой задачей. Не существует алгоритма, который принял бы любую грамматику в качестве входа, и определил однозначна она или нет. Методы разбора могут быть детерминированные и не детерминированные, в зависимости от того, возможен возврат или нет. Не детерминированные методы весьма дорогие с точки зрения памяти и времени., общий перебор.

## Лекция 23.11.07

Базовые методы синтаксического анализа.

Вариант построения синтаксического анализа.

Нисходящий разбор - синтаксическое дерево строится от корня к листьям, его отличительная черта является целенаправленность, так как отправляясь от нетерминального символа языка, мы стремимся найти такую подстановку, которая бы привела к части цепочки терминальных символов. Достигается это путем направленного перебора различных вариантов. В списке правил подстановке отыскивается правило, которые в левой части содержат не терминальные символы, а в правой части символы терминальные анализируемого предложения. Если такое правило есть, то дерево не рассчитывается и правило повторяется. Если правило не найдено, то возвращаемся на один или несколько шагов назад, пытаясь изменить выбор сделанный ранее. Процесс разбора заканчивается в одном из двух случае.

Построенное дерево, все листья которого являются терминальными символами и при чтении с лево на право образуют анализируемое предложение. В этом случаи результат положительный, синтаксически рассматриваемое предложение соответствует грамматике языка.

Распознаватель переработал все возможные варианты, но так и не пришел к дереву, значит анализируемое предложение не принадлежит данному языку или содержит ошибку.

«константа»:: = «КФТ» | «знак» «КФТ»

Шаг 1 константа

Шаг 2 КФТ

Восходящий разбор – дерево строится от листьев к корню, то есть алгоритм отправляется от заданной строки, пытается применить правило подстановки с лева на право и все это привести к начальному символу грамматики. Часть строки, которую можно привести к нетерминальному символу называют фразой. Если приведение осуществляется приведением одного правило подстановки, фраза называется непосредственно приводимой. Самая левая непосредственная фраза называется основой. Алгоритм разбора заключается в следующем: в исходном предложении отыскивается основа и приводится к нетерминальному символу. Эта операция применяется до тех пор, пока не получим единый символ и он должен быть начальным символом грамматики. Либо в цепочке не может быть найдена фраза, в этом случаи делается возврат на один или несколько шагов, выбирается другая основа, если все возможные варианты перебраны, а корень дерева так и не построен, делается вывод об наличии ошибки. Восходящий разбор представляет собой перебор вариантов, но они не целенаправленны.

Нисходящие и восходящие методы требуют большого количества перебора. Поэтому требую только детерминированные методы.

Метод рекурсивного спуска – хорошо известный легко реализуемый и детерминированный метод разбора с верху в низ. С его помощью на основании соответствующей грамматике, можно быстро написать синтаксический анализатор. Основное преимущества – скорость создания анализатора. Другое преимущество заключается в соответствии между грамматикой и анализатором, благодаря тому что увеличивается вероятность того, что анализатор правильный. Основной недостаток - медленность, много вызовов. Вручную грамматику изменим, в ведем два нетерминальных символа. По грамматике пишем программу синтаксического анализатора. Lex – функция, которая выделяет лексему.

## Лекция 30.11.07

Ll(1) – грамматика

Контекстно-свободные грамматики традиционно служат основой создания синтаксических анализаторов. Для того чтобы построить де терминированный анализатор работающий по принципу сверху в низ используется Ll(1) грамматика. Первая l означает, что исходная строка разбивается с лево на право, вторая буква – левосторонний разбор, а цифра означает, что варианты порождающих правил выбирается с помощью одного предварительного просматриваемого символа.

Определим S-грамматику.

Правая часть порождающего правила начинается с терминала.

В тех случаях, когда в левой части более одного одинаковых не терминала, то соответствующие правые части начинаются с разных терминалов.

Для того что бы грамматика была, необходимым условием является множеством символам предшественников не должно пересекаться. Грамматику называют Ll(1) если для каждого не терминала появляющегося в левой части более одного раза множества направляющих символов соответствующих правил не пересекаются. Возникает вопрос, все ли грамматики. Существует ли алгоритмы, определяющие свойства. Однако, грамматику, можно преобразовать что бы она стала Ll(1).

Что бы заменить левую рекурсию на правую мы упорядочиваем не терминалы.

Факторизация – во многих ситуациях грамматику не обладающих признаками Ll(1) можно преобразовать в грамматику Ll(1). Процесс факторизации нельзя автоматизировать, распространив его на общий случай.

## Лекция 07.12.07

Ll(1) – грамматика

После нахождения грамматики, можно перейти к построению синтаксического разбора. Этот этап аналогичен рекурсивному спуску, только здесь исключается многочисленные вызовы процедур, благодаря представлению грамматики в табличном виде. Представим грамматику в виде схемы, номера соответствующие элементам будут являться номерами строк в таблице разбора.

В таблицу разборов включают по одному элементу на каждое правило грамматики. И на каждый экземпляр терминала и не терминала правой части правильной грамматики. Таблица состоит из шести столбцов.

1 столбец – направляющие символы (терминал)

2 столбец – поле перехода, обычно дает следующий элемент для обработки (номер строки). Если значение поля возврата, не окажется истинной, то адрес следующего элемента берется из стека. Это соответствует концу правила.

3 столбец – направляющие символы, переход

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Терминал | Переход | Принимать | стек | возврат | ошибка |
| 1 | Begin | 2 | f | f | f  | t |
| 2 | Begin | 3 | t | f | f | t |
| 3 | d | 7 | f | t | f | t |
| 4 | coma | 5 | t | f | f | t |
| 5 | s | 15 | f | t | f | t |
| 6 | end | 0 | t | f | t | t |
| 7 | d | 8 | f | f | f | t |

1 действие - begin считывается и проверяется. Стек пуст, и используется в стек разборах для указания адресов возврата. Переходим на строку 2. Проверяем и принимаем begin.

В таблице каждому шагу разбора соответствует один элемент. В процессе разбора осуществляется:

Считываем и проверяем предварительно просматриваемый символ. С тем, чтобы выяснить не является ли он направляющим для какой либо конкретной правой части порождающего правила. Если этот символ не направляющий, то она проверяется на следующем этапе.

Осуществляется проверка терминала, появляющаяся в правой части порождающего правила.

Проверка не терминала. Она заключается в проверке нахождения предварительно просматриваемого символа, в одном из множеств направляющих символов. Помещения в стек адреса возврата и переходу к первому правилу относящемуся к данному правилу. Если нетерминал появляется в конце правой части, то нет необходимости помещать в стек. Программа содержит цикл процедуры. Тело которое обрабатывает элемент таблицы разбора и определяется следующий элемент для обработки. Если предварительно просматриваемый элемент отсутствует в списке системы и значение поле ошибки окажется ложью, нужно обрабатывать следующий элемент с тем же символом. Ели предварительно просматриваемый символ не содержится в текущем и поле ошибки t, то выдается сообщение о синтаксической ошибке.

Преимущества:

Никогда не требуется преимущества возврата, поскольку этот метод не терминированный.

Имеются хорошие диагностические характеристики, и существует возможность исправления ошибок. Так как синтаксические ошибки распознаются по первому не приемлемому символу, а в таблице разборов есть список возможных символов продолжения.

Таблица разбора меньше чем соответствующие таблицы в других методах, значит скорость выше.

LL1 разбор применяется к широкому классу языков, однако в большинстве случаев требуется ручное преобразование.

LR(1) – снизу в верх, разбираемый детерминированный. К – используется правосторонний разбор, от начального символа.1 - фиксированное число предварительно просматриваемых символов. Первое действие – сдвиг, во время которого считывается и помещается в стек символ, это соответствует продвижению на один пункт вдоль какого либо правила грамматики. Приведение, во время которого множество элементов верхней части стека замещается каким либо не терминалом грамматики.

## Лекция 14.12.07

S - > real IDLIST

IDLIST - >IDLIST

IDLIST - > ID

ID - > a b с d

Стек символов

A ID IDLIST

Real real real

ID

IDLIST

Real

Чтобы построить таблицу разбора необходимо найти все состояния грамматики.

Таблица разбора представляет собой матрицу состоящую из столбцов – для каждого терминала и не терминала грамматики + признак окончания, и строк соответствующему каждому состоянию.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Состояние | S | IDLIST | ID | real | , | A B S D |  |
| 1 | HALT |  |  | S2 |  |  |  |
| 2 |  | S5 | S4 |  |  | S3 |  |
| 3 |  |  |  |  | R4 |  | R4 |
| 4 |  |  |  |  | R3 |  | R3 |
| 5 |  |  |  |  | S6 |  | R1 |
| 6 |  |  | S7 |  |  | S3 |  |
| 7 |  |  |  |  | R2 |  | R2 |

Таблица разбора включает элементы 4 типов. Сдвиг S 2 – 2 означает состояние, поместить в стек символов соответствующие столбцу символ. В стек состояния поместить 2 и перейти в состояние 2. Если входной символ терминал, принять его.

R4 – r означает элемент приведение, 4 означает 4 правило вывода. Выполнить приведения. Удалить элемент.

3 элемент – пробел, соответствует ошибке.

Сравнительный анализ методов

Оба метода детерминированы и могут обнаруживать синтаксические ошибки на самом раннем этапе.2 метод применяется к более широкому классу языков и грамматик и не требует преобразования грамматики. Дд1 требует преобразования, и при наличии хорошего преобразователя не вызывает затруднения.

Экспериментальные данные выполнены с помощью анализатора при сравнении максимального и минимального время разбора предложения пришли к мнению что метод LL быстрее на 50%, то есть метод с верху в низ быстрее на 50%.

После синтаксического анализатора, последним шагом процесса компиляции является генерация кода. Как только распознан фрагмент исходного текста программ соответствующий некоторому правилу грамматики, вызывается семантическая подпрограмма, которая не посредственно генерирует код.

Все реально существующие компиляторы, на этапе разбора входных цепочек, проверяет только синтаксис входного языка не учитывая его семантику. Для проверки необходимо иметь информацию о найденных лексических единицах языка.

## Лекция 21.12.07

Генерация кода

Сложные компиляторах могут компилироваться промежуточные формы представления программ, пригодные для последующего анализа, с целью генерации более эффективного объектного кода.

Оператор ввода

Readln

C

IDLIST

<IDLIST>

ID

a

Промежуточные формы

Последовательность четверок

Последовательность троек

Полиз – позволяет представлять любое математическое выражение без скобок

S->EVP

EVP-> TERM

TERM->FACT

FACT->FACT

ID->A|B|C|D

Грамматика четверок

QUAD->OPERAND OPI OPERAND=INT

OP2 OPERAND=INT

OPERAND->INT|ID

INT->DIGIT|DIGIT INT

DIGIT-> 0|1|2|3|4|5|6|7|8|9

OP|+-|\*

ID->a|b|c|d|e

Оптимизация

На основании четверок может осуществляться анализ и модернизация промежуточного кода.

Цель: оптимизация.

Можно исключать некоторые операции запоминания и загрузки.

Эффективно использовать промежуточные формы.

Уменьшается длина программы, уменьшается количество переменных. Существует и Машино независимая оптимизация.

Лекция 28.12.07

Распределение памяти. Структурированные переменные.

Компилятор для хранения структурированных элементов должен выполнить несколько этапов:

Выделить память под массив, для этого он должен знать границы массива.

Заполнить информацию характеризующую структурную переменную, размер, тип массива и указатель на его начала.

Сгенерировать информацию для обращения компонентам структурированной переменной.

Породить описатель структурированной переменной, для тех случаев, когда необходимая информация отсутствует во время компиляции.

Аналогичная информация возникает при обработки записи строк и множеств.

Рекурсивный вызов процедур, в случаи использования статического распределения памяти не работает. Эту проблему решают с помощью динамическое распределение памяти. Каждый вызов приводить к образованию области инициализации. Обычна область инициализации располагается в стеки, и располагается следующей информацией. Содержит все переменные, адрес возврата, хранит адрес следующего и предыдущего вызова. Этот метод называется метод автоматического распределения.

Варианты создания компиляторов.

Скорость работы

Качество кода

Диагностика ошибок

Переносимость

Поддержка

Если важна скорость компиляции, то одна просмотровая схема предпочтительней. Однако не все языки высокого уровня.

Если с компилированные объектные модули используются многократно, или памяти другие ресурсы существенно ограниченны или модули обрабатывают большие массивы данных, то скорость выполнения программы становится более важным фактором.

Компиляторы с использованием промежуточного кода.

Интерпретатор 3

Интерпретатор 2

Интерпретатор 1

Объектная программа

Генераторы, компиляторы.