|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** | | | | |
|  | | | | |
| БГТУ.СМК-Ф-4.2-К5-01 | | | | |
| Факультет | |  | И |  | Информационные и управляющие системы | |
|  | |  | шифр |  | Наименование | |
| Кафедра | |  | И9 |  | Систем управления и компьютерных технологий | |
|  | |  | шифр |  | Наименование | |
| Дисциплина | |  | «Системное программное обеспечение» | | | |

КУРСОВАЯ РАБОТА

на тему

Создание компилятора с подмножества языка

|  |
| --- |
|  |
| высокого уровня FALSE на язык ассемблер i386 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | | И975 |
| Лопушанский А.М. | | | | | | |
| Фамилия И.О. | | | | | | |
| **Преподаватель** | | | | | | |
| Афанасьев М.П. | |  |  | | | |
| Фамилия И.О. Подпись | | | | | | |
| Оценка |  | | | |  | |
| « » |  | | | | 2019 г. | |

САНКТ-ПЕТЕРБУРГ

2019

СОДЕРЖАНИЕ

Введение 3

1 Лексический анализатор 5

2 Синтаксический анализатор 9

3 Генератор кода ??

4 Тестовые примеры ??

Заключение ??

Список использованных источников ??

ВВЕДЕНИЕ

В рамках курсовой работы необходимо разработать программу, транслирующую код с языка высокого уровня на эквивалентный код на языке ассемблера. Для данной работы было выбрано подмножество языка высокого уровня FALSE. По завершению работы необходимо проверить работоспособность тестовых примеров, используя написанную программу.

В выбранное подмножество языка входят математические, логические операции, операции со стеком, условный оператор, цикл с предусловием.

В результате предварительного анализа тестовых примеров, было принято решение расширить подмножество, добавив в него тип данных с плавающей точкой, массивы.

Компилятор должен состоять из лексического анализатора, синтаксического анализатора, генератора кода.

За время разработки компилятора должны быть выполнены следующие задачи:

1. Разработка лексического анализатора
   1. Составление описания лексики языка, составление лексических диаграмм.
   2. Описание методов обработки входных данных лексического анализатора. Описание дискретного конечного автомата.
   3. Описание конструкций данных, реализуемых в лексическом анализаторе, методы их обработки.
2. Разработка синтаксического анализатора
   1. Описание грамматики языка, ее классификация. Составление синтаксических диаграмм.
   2. Описание метода обработки потока лексем, проверки допустимости для выбранной грамматики, генерации дерева разбора.
   3. Описание реализации МП автомата и генерации дерева разбора, используемых для этого конструкций данных и методов их обработки.
3. Разработка генератора кода
   1. Описание выходного языка i386 и его конструкций.
   2. Описание методов генерации кода по дереву разбора.
   3. Описание конструкций данных и методов их обработки при генерации кода.
4. Проверка работы компилятора на тестовых примерах.

Итого должно быть решено 10 задач, в результате чего будет готов компилятор, а цель работы выполнена.

1 ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР

* 1. Описание лексики языка, лексические диаграммы.

В алфавит подмножества языка FALSE входят следующие символы:

1. Буквы - символы [a-z] и [A-Z].
2. Цифры - [0-9].
3. Служебные символы: «+», «-», «\*», «/», «:», «~», «=», «&», «|», «$», «%», «\», «;», «@», «?», «#», «(», «)», «[», «]», «{», «}», «**½**», «!», «’», «\_».
4. Разделители: табуляция, пробел, перенос строки.

В языке FALSE нет зарезервированных слов, поэтому имя переменной может содержать в себе любое сочетание букв и цифр, но не начинаться на цифру.

Тип данных специальным словом объявлять не нужно. Целое число отличается от числа с плавающей запятой наличием символа «½».

В данном языке каждый символ, кроме скобок, является терминальным.

Описание языка в виде РБНФ:

УпрСимвол = “+”|“-”|“\*”|“/”|“:”|“~”|“=”|“&”|“|”|“$”|“%”|“\”|“;”|“@”|“?”|“#”  
|“(”|“)”|“[”|“]”|“!”|“`”|”\_”.

Цифра = “1”|”2”|”3”|”4”|“5”|”6”|”7”|”8”|“9”|”0”.

ЦелоеЧисло = Цифра{Цифра}.

ЧислоСПлавЗап = [ЦелоеЧисло]½[ЦелоеЧисло].

Буква = “a”|”b”|”c”|”d”|“e”|”f”|”g”|”h”|“i”|”j”|“k”|”l”|”m”|”n”|“o”|”p”|”q”|”r”|“s”|  
”t“|”u”|”v”|”w”|”x”|“y”|”z”|“A”|”B”|”C”|”D”|“E”|”F”|”G”|”H”|“I”|”J”|“K”|”L”|”M”|”N”|“O”|”P”|”Q”|”R”|“S”|”T“|”U”|”V”|”W”|”X”|“Y”|”Z”.

Идентификатор = Буква{Буква}.

Комментарий = “{“{? Любой символ ASCII ?}”}”.

1.2 Описание методов обработки входных данных лексического анализатора. Описание дискретного конечного автомата.

Задача лексического анализатора – разбить входной поток на лексемы, исключить из текста исходной программы комментарии, незначащие пробелы, символы табуляции и переносы строк, составить вектор лексем, идентификаторов и констант.

При анализе выделяются лексемы следующих типов:

1. Идентификаторы – пользовательские имена объектов программы.
2. Константы – указанные в программе числовые значения.
3. Знаки операций – символы, обозначающие математические, логические операции, операции со стеком.
4. Специальные символы – квадратные и круглые скобки.
5. Разделители – пробелы и символы новой строки.
6. Комментарии – последовательность символов, заключенная в фигурные скобки

На вход лексического анализатора подается поток символов. Символы рассматриваются по одному. Если встречается буква, то происходит считывание идентификатора, пока не встретится иной символ. После этого полученная строка заносится в таблицу идентификаторов. Так же считываются целые числа. В случае, если встречается знак «½», то продолжается считывание следующего целого числа, в противном случае число заносится в таблицу констант.

Если встречается открывающая фигурная скобка, то пропускаются все символы, пока не встретится закрывающая фигурная скобка, или не закончится программа.

Все знаки операций и специальные символы сразу обрабатываются, как отдельные лексемы.

Язык FALSE является автоматным, так как существует конечный автомат, распознающий его. В нашем случае используется конечный автомат, изображенный на рисунке 1.

Данный конечный автомат не содержит ни одного перехода с меткой длиной больше единицы, это возможно, так как язык является автоматным.

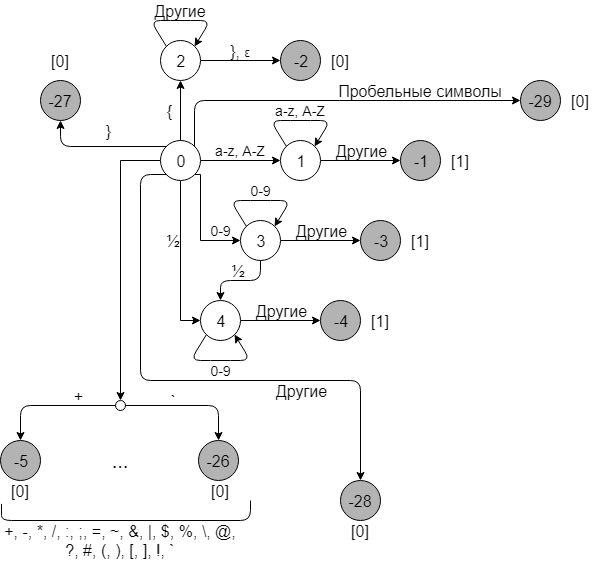


Рисунок 1 – Детерминированный конечный автомат

В квадратных скобках указано число возвращаемых символов обратно в поток.

1.3. Описание конструкций данных, реализуемых в лексическом анализаторе, методы их обработки.

Для реализации лексического анализатора были созданы следующие структуры данных:

Структура, содержащая в пронумерованном виде токены для всех обрабатываемых лексем. Задается перед работой программы.

Вектор лексем – итоговая структура, которая хранит все токены лексем в том порядке, в котором они встречались в программе. Она является итогом работы лексического анализатора.

Лексема – структура, хранит тип лексемы и ее значение.

Вектор идентификаторов – структура, содержащая имена всех встречающихся в программе идентификаторов. Если встречается новый идентификатор, то сначала производится поиск по вектору, не встречался ли он раньше. Если встречался, то в вектор лексем добавляется токен, а вектор идентификаторов не изменяется; если же идентификатор встречается впервые, то перед добавлением токена в вектор лексем, производится добавление имени идентификатора в вектор идентификаторов.

Вектор констант – структура, содержащая все явно указанные в программе числовые константы. Принцип заполнения идентичный вектору идентификаторов.

Программа, которую будет обрабатывать лексический анализатор, считывается полностью из файла в строку. Такой подход имеет смысл, так как ввиду минимализма языка большинство программ на нем имеют сравнительно небольшой размер. Каждый символ обрабатывается конечным автоматом, описанным выше. По итогу работы лексического анализатора получаются три структуры данных, передаваемых синтаксическому анализатору: вектор лексем, вектор констант, вектор идентификаторов.

2 СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР

2.1. Описание грамматики языка, ее классификация. Составление синтаксических диаграмм.

Грамматика языка FALSE является контекстно-свободной, так как каждое правило в ней имеет вид A→α, где A∈N, α∈(N∪Σ)\*. Язык также является контекстно-свободным, так как он порождается контекстно-свободной грамматикой.

Синтаксис FALSE можно описать в виде РБНФ:

S = УпрСимвол|ЦелоеЧисло|ЧислоСПлавЗап|Идентификатор|  
”(“ ЦелоеЧисло”)” Идентификатор|”(“ ЧислоСПлавЗап”)” Идентификатор |”[“S”]”|Ɛ|SS.

2.2 Описание метода обработки потока лексем, проверки допустимости для выбранной грамматики, генерации дерева разбора.

Задача синтаксического анализатора – обработать входной поток лексем и составить по нему дерево разбора.

Так как для любой КС-грамматики G можно построить недетерминированный МП-автомат, допускающий язык L(G).

Построим МП-автомат, допускающий язык FALSE:

L={}.

Пусть M= ({q0 , q1 , q2}, {a, b,c}, {Z0 , a,b}, δ , q0 , Z0 ,{q2}).

Тыры пыры, выходит, что язык детерминированный, т.к. распознается дмп-автоматом, который дмп из-за 1 из 2 правил.

2.3 Описание реализации МП автомата и генерации дерева разбора, используемых для этого конструкций данных и методов их обработки.

Для реализации синтаксического анализатора были созданы следующие структуры данных:

Структура «ветка» - содержит указатели на три дочерние ветки и хранит в себе одну лексему. Нужна для построения дерева разбора. Если дочерних веток нет – указатели хранят нулевую константу.

МП-автомат как-то реализован

Дерево как-то строится

Итогом работы синтаксического анализатора будет построенное дерево – структура ветка, хранящая адрес дочерних веток, которые в свою очередь хранят адреса своих дочерних веток.

3 ГЕНЕРАТОР КОДА

3.1 Описание выходного языка i386 и его конструкций

[Регистр](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0) — это небольшой объем очень быстрой памяти, размещённой на процессоре. Он предназначен для хранения результатов промежуточных вычислений, а также некоторой информации для управления работой процессора. Так как регистры размещены непосредственно на процессоре, доступ к данным, хранящимся в них, намного быстрее доступа к данным в [оперативной памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C).

Регистры общего назначения:

%eax: Accumulator register — аккумулятор, применяется для хранения результатов промежуточных вычислений.

%ebx: Base register — базовый регистр, применяется для хранения адреса (указателя) на некоторый объект в памяти.

%ecx: Counter register — счетчик, его неявно используют некоторые команды для организации циклов.

%edx: Data register — регистр данных, используется для хранения результатов промежуточных вычислений и ввода-вывода.

%esp: Stack pointer register — указатель стека. Содержит адрес вершины стека.

%ebp: Base pointer register — указатель базы кадра стека (англ. stack frame). Предназначен для организации произвольного доступа к данным внутри стека.

%esi: Source index register — индекс источника, в цепочечных операциях содержит указатель на текущий элемент-источник.

%edi: Destination index register — индекс приёмника, в цепочечных операциях содержит указатель на текущий элемент-приёмник.

Стек обычно используется для сохранения адресов возврата и передачи аргументов при вызове процедур (о процедурах в следующей части), также в нём выделяется память для локальных переменных. Кроме того, в стеке можно временно сохранять значения регистров.

Для стека существуют всего две основные операции: добавление элемента на вершину стека (PUSH), извлечение элемента с вершины стека (POP).

Основные используемые команды ассемблера:

Add - cложение, sub – вычитание, cmp – сравнение двух операндов, div – беззнаковое деление, mul - умножение, jmp – безусловная передача управления, mov – пересылка данных.

3.2 Обход дерева разбора. Генерация кода.

Для генерации кода на языке ассемблера необходимо обойти составленное дерево разбора. Для этого используется рекурсивный обратный обход. Каждый раз, когда мы находимся в нетерминальном узле дерева, вызывается функция обхода для левой ветки, затем для правой. В случае, если узел терминальный, то транслируем его в код ассемблера.

Ввиду особенностей языка FALSE, после обхода дерева нет необходимости переводить выражения в обратную польскую запись, т.к. язык изначально предполагает написание в этой форме.

Когда в дереве встречается терминальный символ, то он сразу же транслируется в ассемблерный код. Например, при встрече константы, она сразу же добавляется в стек:

push константа

при встрече оператора «+»:

pop ebx  
pop eax   
add eax,ebx   
push eax

3.3 Описание конструкций данных и методов их обработки при генерации кода.

В генераторе кода используются те же конструкции, что и в синтаксическом анализаторе, так как разбирается то же дерево, что было построено в нем.

При прохождении дерева, каждый отдельный терминал транслируется однозначно в код ассемблера без каких-либо дополнительных операций.

Итогом работы генератора кода является сгенерированный ассемблерный код, эквивалентный изначальному на языке FALSE.

4 ТЕСТОВЫЕ ПРИМЕРЫ