**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

**РАЗРАБОТКА КОМПИЛЯТОРА**

Пояснительная записка

На 16 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
| Исполнитель |  | студент гр. ИCБ-118 А.С. Соболев |

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72713406)

[**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72713407)

[1.1 Основные требования 4](#_Toc72713408)

[1.2 Лексический анализатор 5](#_Toc72713409)

[1.3 Синтаксический анализатор 6](#_Toc72713410)

[1.4 Построение таблицы символов и генерация промежуточного представления 10](#_Toc72713411)

[1.5 Трансляция в целевой код 11](#_Toc72713412)

[**2** **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ** 12](#_Toc72713413)

# **Аннотация**

В данной пояснительной записке приведено описание и примеры работы компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Python с использованием библиотеки ply. Основная функция компилятора – трансляция модуля входного языка в эквивалентный модуль на языке ассемблера MIPS

Разработка компилятора подмножества процедурного языка в ассемблер состоит из следующих стадий:

* Разработка лексера
* Разработка парсера
* Разработка таблицы символов
* Генерация промежуточного кода
* Трансляция в целевой код

# **1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

Требования к операторам:

1. Оператор присваивания.
2. Арифметические операторы (\*, / , +, -, >, <, =).
3. Логические операторы (И, ИЛИ, НЕ).
4. Условный оператор (ЕСЛИ).
5. Оператор цикла (while).
6. Базовый вывод (строковой литерал, переменная).
7. Типы (целочисленный, вещественный).

Требования к выходному языку:

1. В ассемблере.

## Лексический анализатор

Лексический анализатор преобразует входной поток символов в поток токенов.

Лексический анализатор реализован при помощи библиотеки ply.lex.

Сам исходный язык представляет собой видоизменённое подмножество языка Pascal: вместо операторных скобок используются “start” и “stop”, объявление переменных происходит в блоке “var”.

Основная часть языка выполнена в виде зарезервированных слов: and, start, div, do, stop, for, func, if, mod, not, or и прочих.

Арифметика представлена в виде массива литералов в который так же включены точка, запятая, точка с запятой, двоеточие и скобки.

Так же в лексере содержатся правила определяющие комментарии в тексте программы, числа с плавающей точкой, строковые литералы, переход на новую строку и правило выбрасывающее ошибку при вводе некорректных символов.

## Синтаксический анализатор

На вход синтаксический анализатор получает набор токенов из лексического анализатора. В самом парсере описаны правила их соединения -грамматика.

Грамматика языка, основная часть модуля:

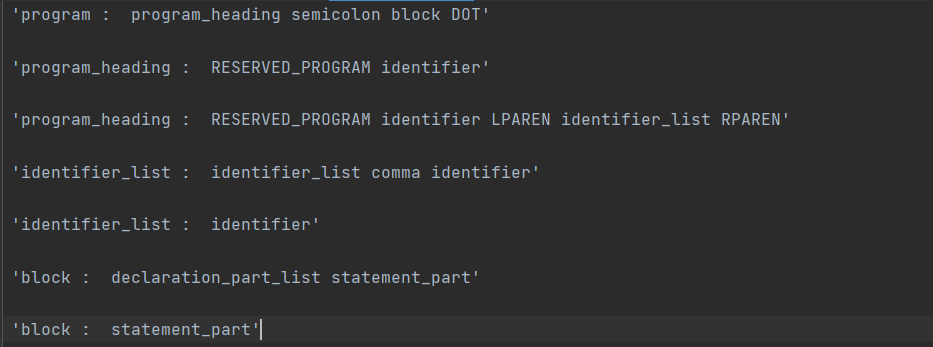


Рис. 1

Рассмотрим примеры использования правил выполнения операций:

Присваивание:

Для присваивания доступны числа, строковые литералы, объявленные переменные типов int, float и string.

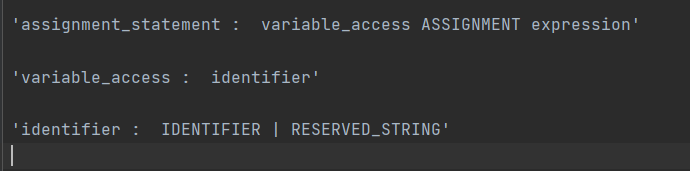


Рис. 2

Арифметика:

Так как большинство операторов представлены в лексере в виде литералов, то и в правилах указаны так же литералы. В иных случаях указаны токены от зарезервированных слов.

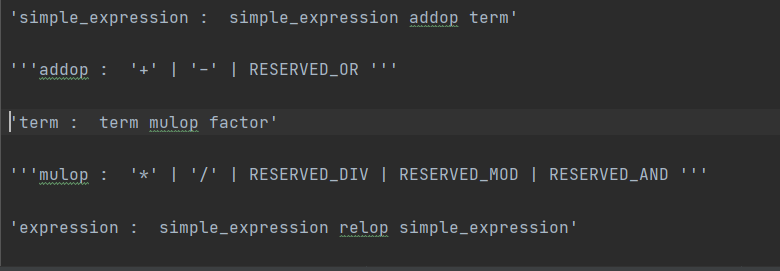


Рис. 3

Условный оператор:

Блок обязательно начинается с зарезервированного слова *if,* далее идет само условие, зарезервированное слово *then* отделяющее оператор перехода и тело цикла



Рис. 4

Функции:

Функция определяется после использования зарезервированного слова *func*. Далее определяются переменные, двоеточие и тип выходных данных.

Вызов функции происходит путем ввода *func* и идентификатора функции, объявленного ранее.

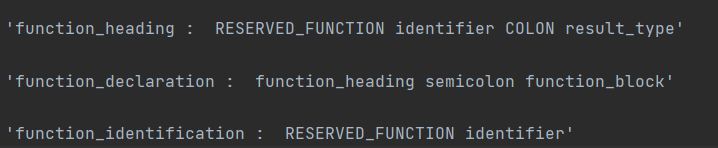


Рис. 5

Пример дерева для программы:

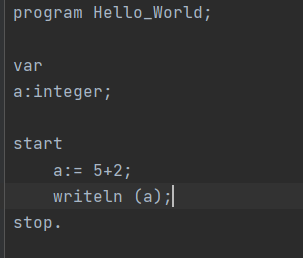


Рис. 6

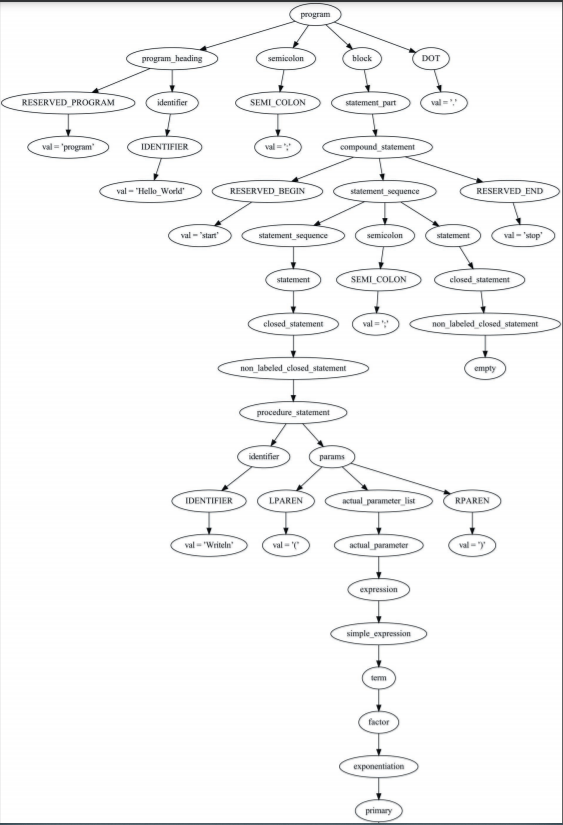


Рис. 7

## Построение таблицы символов и генерация промежуточного представления

Таблица символов получает на вход от парсера информацию об объявленных переменных, функциях, их типах, уровнях вложенности и значениях.

Таблица символов представлена в виде хэш-таблицы. Таблица содержит в себе ключ и шесть значений. Ключом является временная переменная (). Поля содержат в себе следующую информацию: offset - сдвиг в стеке, width – размер, type – тип, variable\_name – имя переменной в программе, id\_path – путь к переменной, name – имя временной переменной.

Полученные данные пересылаются в генератор промежуточного кода.

Промежуточный код представляет собой трёхадресный код типа x:= y op z, который записывается в виде ‘dest(x), src1(y), src2(z), op’.

Поскольку трёхадресный код является представлением более близким к машинному, то и генерировать код ассемблера становится проще.

## Трансляция в целевой код

На данном этапе промежуточный код подается на вход генератору машинного кода. Генератор прогоняет промежуточный код по множеству условий, таких как, например, наличие арифметический инструкций, вызова функций или вывода и при обнаружении таковых генерирует соответствующие инструкции на языке ассемблера MIPS.

Основные функции в генераторе:

* Addops – операции сложения, вычитания, присваивания, логическое И и ИЛИ
* Mulops – операции умножения, деления, целочисленного деления и остатка от деления
* Relops – операции сравнения
* Change\_type – смена типов
* Jumps - переходы
* Output – вывод

# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

В задании на курсовую работу были предъявлены следующие требования к реализуемому компилятору:

Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

Требования к операторам:

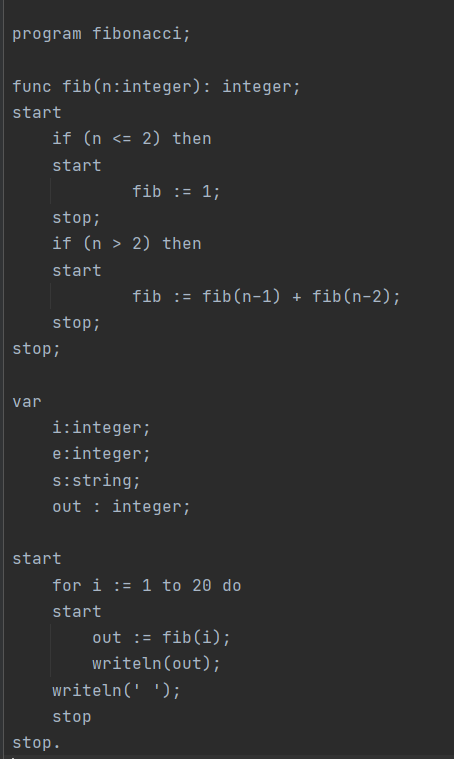
1. Оператор присваивания.
2. Арифметические операторы (\*, / , +, -, >, <, =).
3. Логические операторы (И, ИЛИ, НЕ).
4. Условный оператор (ЕСЛИ).
5. Оператор цикла (while).
6. Базовый вывод (строковой литерал, переменная).
7. Типы (целочисленный, вещественный).

Требования к выходному языку:

1. В ассемблере.

Рассмотрим выполнение указанных выше требований на примерах работы компилятора:

Нахождение последовательности Фибоначчи:

  
Рис. 8

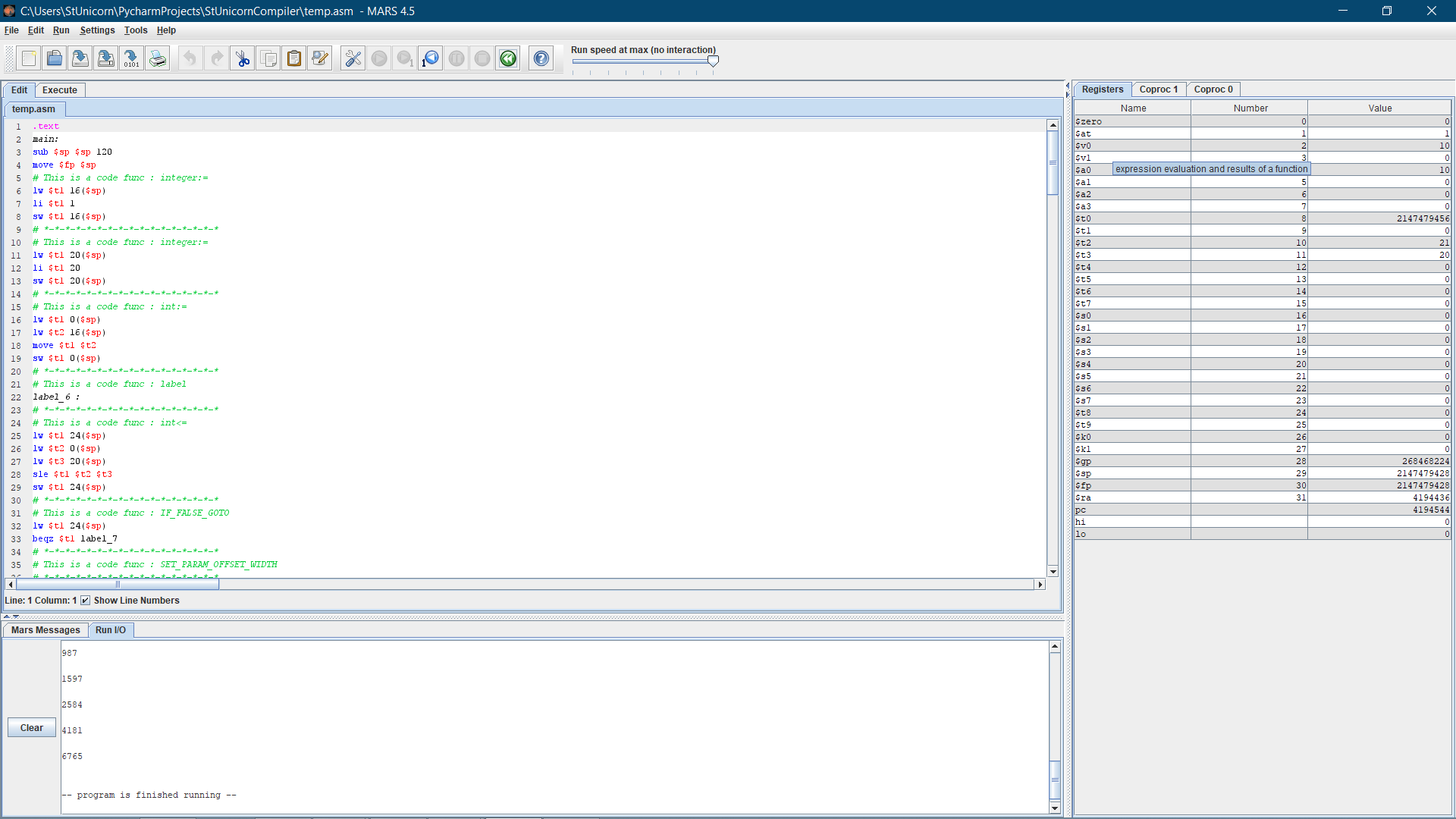


Рис. 9

Факториал:

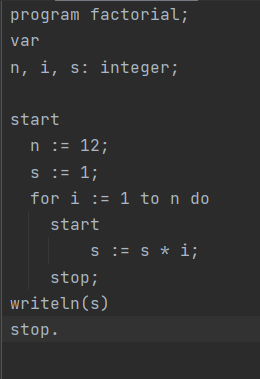


Рис. 10

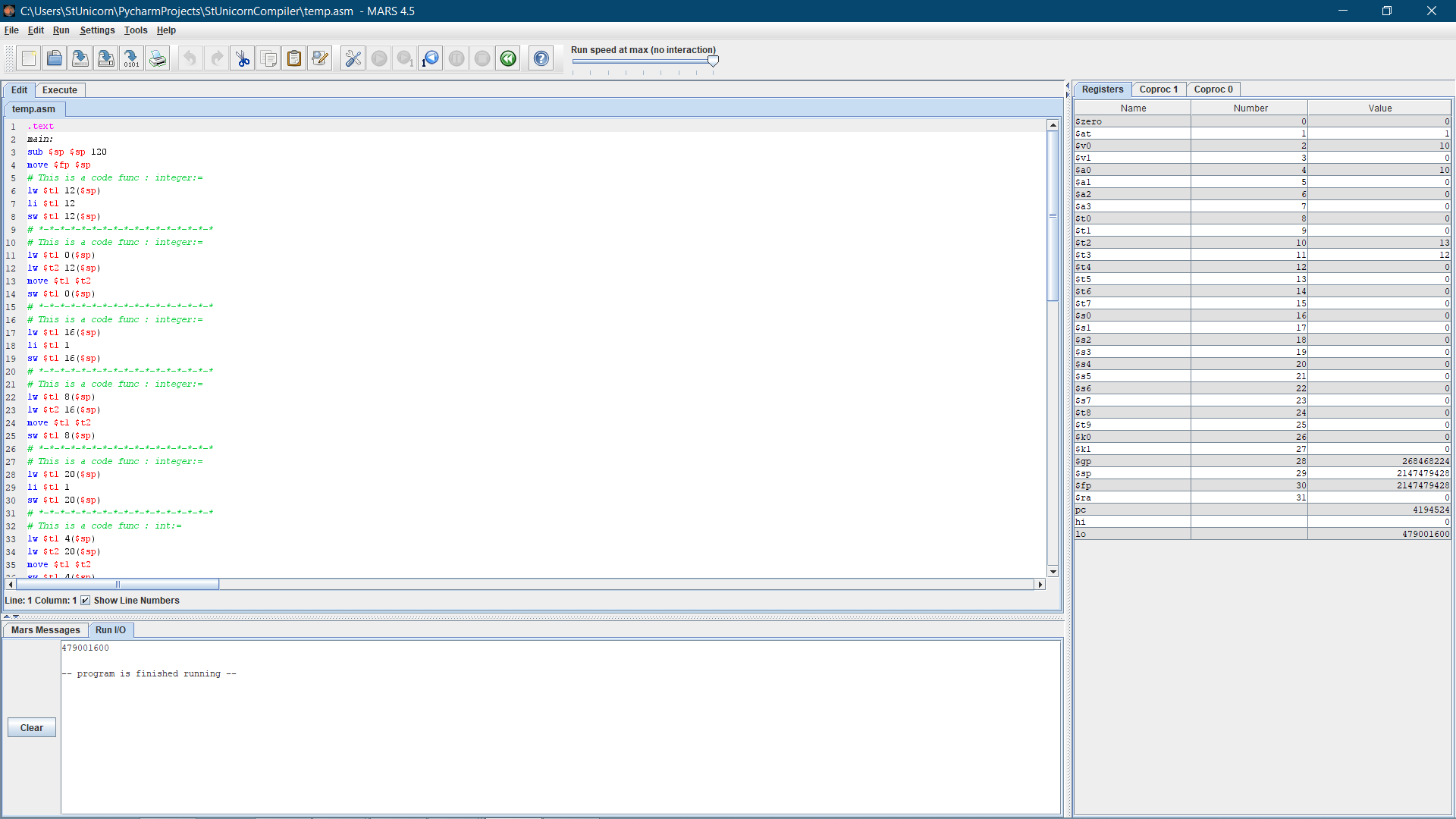


Рис. 11

Возведение в степень:

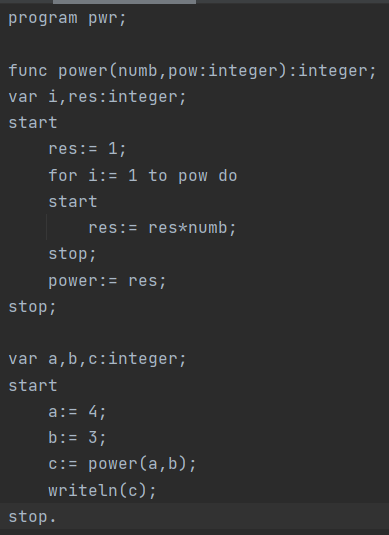


Рис. 12

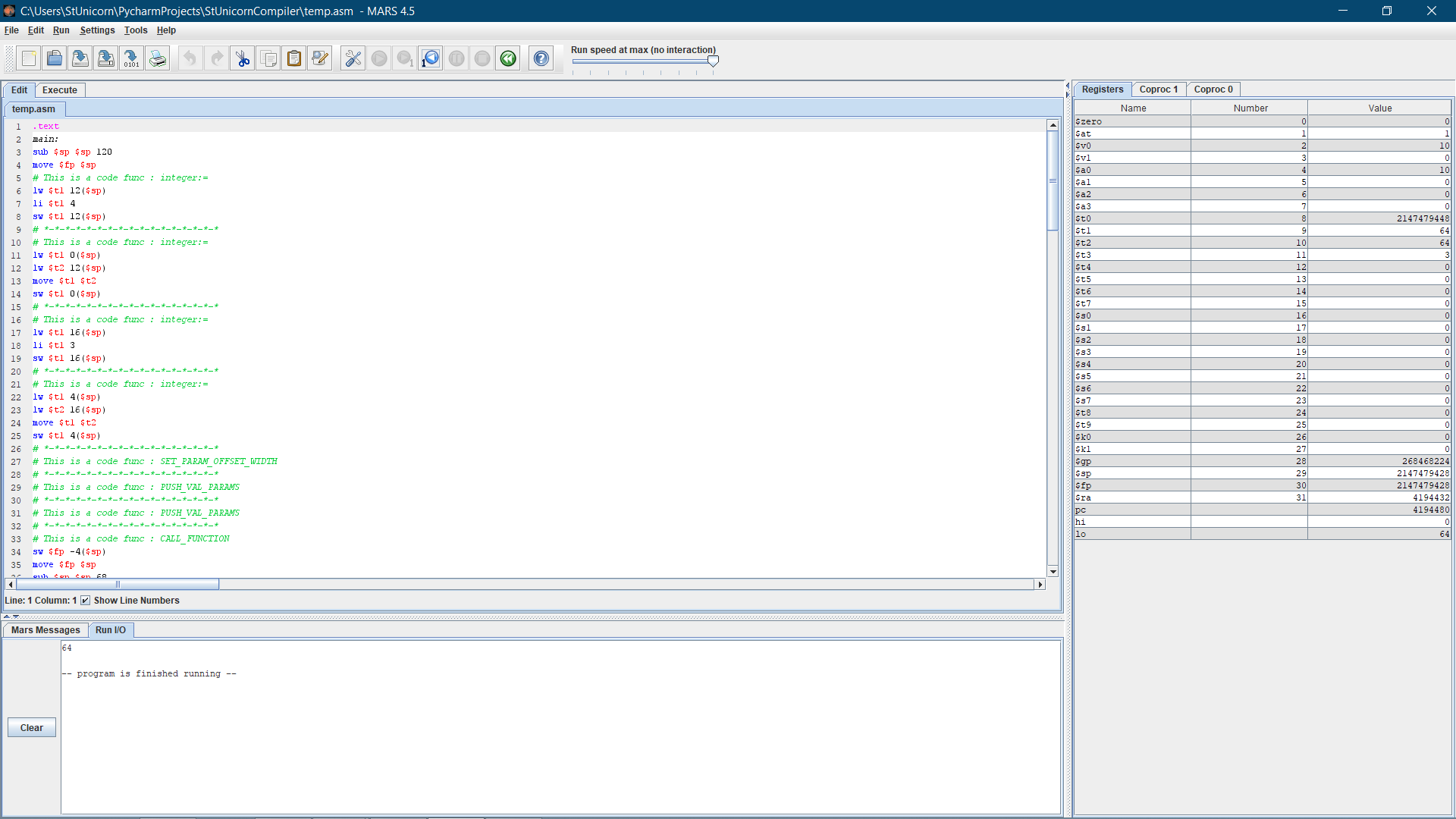


Рис. 13