

Университет ИТМО
Факультет программной инженерии и компьютерной техники

Разработка компиляторов
Проектная работа
Разработка компилятора и интерпретатора

Выполнил
и студенты:

Суворова Елизавета Михайловна Р3323,

Подольский Вячеслав Ильич Р3320

Преподаватель: Лаздин Артур Вячеславович

Санкт-Петербург

2025

ОГЛАВЛЕНИЕ

| | |
|--------------------------|----|
| Задание | 2 |
| Ход работы | 3 |
| Описание языка | 3 |
| Структура проекта | 4 |
| Фрагмент кода грамматики | 4 |
| Примеры работы | 7 |
| Обработка ошибок | 16 |

Задание

Необходимо разработать компилятор для учебного языка программирования.

Вход компилятора: программа написанная на учебном языке программирования.

Выход компилятора: текст программы на языке ассемблера.

Предлагается использовать эмулятор RISC-V процессора (ссылка на README в репозитории эмулятора <https://github.com/asurkis/risc-emulator>). По согласованию с преподавателем язык целевой машины может быть изменен.

Характеристики учебного языка программирования (императивный тьюринг полный язык), поддерживающий следующий набор операторов и выражений:

Операторы

1. Определения переменных. Поддерживаемые типы данных: целые числа со знаком, строки (можно ограничиться набором ASCII символов из первой половины таблицы коды от 32 до 127 и управляющими символами переноса строки и пр.).
2. Присваивание (может быть заменено выражением в случае определения оператора-выражения).
3. Ветвление (if) с факультативным else.
4. Цикл while.
5. Блок (составной оператор).
6. Вывода (типа print).

Выражения

1. Арифметические (минимум + * - /).
 2. Логические not, and, or.
 3. Для чего поддерживать операции (operator) арифметические, логические сравнения.
- Допускается использовать ключевые слова для определения операторов и операций отличные от общепринятых. Полный список операторов, операций и ключевых слов должен быть приведен в отчете.

Для разработанного учебного языка программирования необходимо:

Определить лексический состав языка и описать правила определения лексем для генератора лексических анализаторов flex. Учесть обработку комментариев: однострочных и многострочных.

Определить разработанный язык в терминах КС грамматики в формате входного файла для GNU Bison.

Для каждого правила грамматики (в рамках синтаксически управляемой трансляции) определить семантические действия, определяющие правила построения AST.

Разработать функцию обхода AST для его визуализации. Для построения AST использовать примеры корректных программ (небольших) для учебного языка.

На этом завершается первый этап, его результаты можно обсудить со мной, если возникнут проблемы.

Второй этап предполагает генерацию кода в процессе обхода AST.

Можно построить промежуточное представление программы в виде трехадресного кода, если вам это будет удобно. Трехадресный код не является целью проекта.

Цель второго этапа - получение ассемблерной программы эквивалентной тексту входной программы.

Проект можно выполнять в парах. Полученная на защите оценка ставится обоим участникам. Выбор языка реализации оставляю за вами.

По итогам будет нужно оформить отчет с описанием лексического и синтаксического состава, демонстрация работы компилятора для корректных и ошибочных программ. (вопросы обработки ошибок обсудим).

Ход работы

Исходный код программы: <https://github.com/DemonM1x/compiler-dev>

Описание языка

Название языка: Our

Язык программирования - императивный тьюринг полный язык со статической типизацией. Синтаксис языка похож на JS.

Обязательное объявление `score` переменной.

`ever` (`var`) - ключевое слово для объявления глобальной области видимости переменной.

`lim` (`let`) - ключевое слово для объявления локальной области видимости на уровне блока.

Язык поддерживает следующий набор операторов и выражений:

Операторы

1. Типы данных: целые числа со знаком, строки;
2. Присваивание (`=`);
3. Ветвление (`if`) с факультативным `else`;
4. Цикл `while`, `round` (`for`) с четким определением диапазона и шага;
5. Блок (составной оператор);

6. Вывод (“print”).

Выражения

1. Арифметические (“+”, “-”, “*”, “/”, “%”);
2. Логические and, or;
3. Работа со строками присваивание и конкатенация

Язык допускает конкатенацию строк при выполнении операции присваивания и запрещает конкатенацию во время вывода, и конкатенацию строки и числа.

Целые числа изменяемы. Строки являются immutable. Строки представляются указателем на начало. Длина строки неизменяема. Максимальная длина строки - 255.

Компиляция происходит в кастомный ассемблер для последующей эмуляции исполнения на RISC-машине.

Поддерживаются однострочные и многострочные комментарии.

С полной грамматикой языка можно ознакомиться в файле: parser.y

Структура проекта

lexer/lexer.l - Лексер языка, сгенерированный с помощью flex.

parser/parser.y - парсер языка, сгенерированный с помощью bison.

ast/ast.c - построение AST с семантикой языка.

ast/ast_visualizer.c - вывод AST в заданный поток.

compiler/risc_generator.c - генерация ассемблерного кода на основе AST.

error_handler.c - обработка ошибок на этапе компиляции кода.

main.c - основной файл с генерацией AST и ассемблерного кода.

examples/ - директория с тестовыми программами.

output/ - директория с ассемблерными кодами программ.

Фрагмент кода грамматики

Lexer

```

36  %%
37  | Ctrl+L to chat, Ctrl+K to generate
38  "if"      {update_column(); return IF; }
39  "else"    {update_column(); return ELSE; }
40  "while"   {update_column(); return WHILE; }
41  "round"   {update_column(); return ROUND; }
42  "in"      {update_column(); return IN; }
43  "range"   {update_column(); return RANGE; }
44  "evere"   {update_column(); return EVERE; }
45  "lim"     {update_column(); return LIM; }
46  "int"     {update_column(); return INT; }
47  "string"  {update_column(); return STRING; }
48  "and"     {update_column(); return AND; }
49  "or"      {update_column(); return OR; }
50  "print"   {update_column(); return PRINT; }

```

Рисунок № 1 - Операторы языка

```

57  "=="      { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
58  "!="      { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
59  "<="      { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
60  ">="      { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
61  "<"       { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
62  ">"       { update_column(); yyval.sval = strdup(yytext); return COMPARE; }
63
64  "+"       { update_column(); return '+'; }
65  "-"       { update_column(); return '-'; }
66  "*"       { update_column(); return '*'; }
67  "/"       { update_column(); return '/'; }
68  "%"       { update_column(); return '%'; }
69  "="      { update_column(); return '='; }
70  "."       { update_column(); return '.'; }
71
72  "("       { update_column(); return '('; }
73  ")"       { update_column(); return ')'; }
74  "{"       { update_column(); return '{'; }
75  "}"       { update_column(); return '}'; }
76  ";"       { update_column(); return ';'; }
77  ","       { update_column(); return ','; }

```

Рисунок № 2 - Операторы языка

Parser

```

61  %right '='
62  %left '+' '-'
63  %left '*' '/' '%'
64  %left '.'
65

```

Рисунок № 3 - Порядок выполнения операций

```

87  operation
88      : declaration ';'      { $$ = $1; }
89      | assignment ';'      { $$ = $1; }
90      | print_opr ';'      { $$ = $1; }
91      | if_opr              { $$ = $1; }
92      | while_opr           { $$ = $1; }
93      | round_opr           { $$ = $1; }
94      | block_stmt          { $$ = $1; }
95      ;
96

```

Рисунок № 4 - Операции

```

133  assignment
134      : IDENTIFIER '=' expr
135      {
136          $$ = create_assignment_node($1, $3);
137          free($1);
138      }
139      ;
140
141  print_opr
142      : PRINT '(' expr ')'
143      {
144          $$ = create_print_node($3);
145      }
146      ;
147
148  if_opr
149      : IF '(' expr ')' operation
150      {
151          $$ = create_if_node($3, $5, NULL);
152      }
153      | IF '(' expr ')' operation ELSE operation
154      {
155          $$ = create_if_node($3, $5, $7);
156      }
157      ;
158

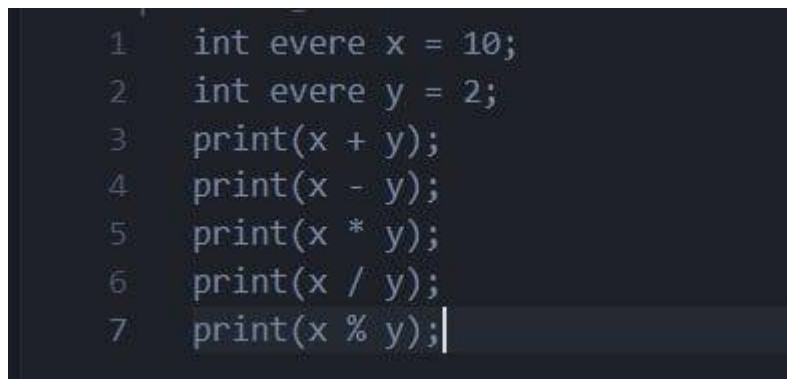
```

Рисунок № 5 - Синтаксис выражений

Примеры работы

С полным перечнем примеров можно ознакомиться в директории `examples/` и ASM в директории `output/`

Пример № 1:



```
1  int evere x = 10;
2  int evere y = 2;
3  print(x + y);
4  print(x - y);
5  print(x * y);
6  print(x / y);
7  print(x % y);
```

Рисунок № 6 - Арифметические операции

#AST:

PROGRAM

VAR_DECL: x (type: int, global: yes)

INIT:

LITERAL: 10 (type: int)

VAR_DECL: y (type: int, global: yes)

INIT:

LITERAL: 2 (type: int)

PRINT

EXPRESSION:

BIN_OP: +

LEFT:

ID: x

RIGHT:

ID: y

PRINT

EXPRESSION:

BIN_OP: -

LEFT:

ID: x

RIGHT:

ID: y

PRINT

EXPRESSION:

BIN_OP: *

LEFT:

ID: x

RIGHT:
 ID: y
 PRINT
 EXPRESSION:
 BIN_OP: /
 LEFT:
 ID: x
 RIGHT:
 ID: y
 PRINT
 EXPRESSION:
 BIN_OP: %
 LEFT:
 ID: x
 RIGHT:
 ID: y

The screenshot shows the risc-emulator interface. On the left, there are control buttons: Reload, Step once, Run, Reload & Run, Stop, and Clear output. The main area displays assembly code with comments. On the right, there are two tables: one for the instruction stream and another for the register file.

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|------------|-------------------|--------------------------------|
| 0030 | 0005 8713 | 362259 | addi x14, x11, 0 | x14 := x11 + 0 |
| 0031 | 0006 5263 | 414307 | bge x12, x0, 2 | if x12 >= x0 then pc := pc + 2 |
| 0032 | 0010 0693 | 1050259 | addi x13, x0, 1 | x13 := 0 + 1 |
| 0033 | 40C0 0633 | 1086326323 | sub x12, x0, x12 | x12 := 0 - x12 |
| 0034 | 02A6 47B3 | 44451763 | div x15, x12, x10 | x15 := x12 / x10 |
| 0035 | 02A6 6833 | 44460083 | rem x16, x12, x10 | x16 := x12 % x10 |
| 0036 | 0308 0F93 | 50859923 | addi x31, x16, 48 | x31 := x16 + 48 |
| 0037 | 01F7 2023 | 32972835 | sw x14, 0, x31 | [x14] := x31 |
| 0038 | FFF7 0713 | -588013 | addi x14, x14, -1 | x14 := x14 + (-1) |
| 0039 | 0007 8613 | 493075 | addi x12, x15, 0 | x12 := x15 + 0 |
| 003A | FE06 19E3 | -33154589 | bne x12, x0, -7 | if x12 != x0 then pc := pc - 7 |
| 003B | 0006 8263 | 426595 | beq x13, x0, 2 | if x13 == x0 then pc := pc + 2 |
| 003C | 02D0 0F93 | 47189907 | addi x31, x0, 45 | x31 := 0 + 45 |
| 003D | 004F 8073 | 5210227 | ewrite x31 | WRITE x31 |
| 003E | 0017 0713 | 1509139 | addi x14, x14, 1 | x14 := x14 + 1 |
| 003F | 0007 2F83 | 470915 | lw x31, x14, 0 | x31 := [x14] |

| Address | Hex | Decimal | Command | Explanation |
|---------|--------|----------|---------|-------------|
| pc | 00A2 | | | |
| x0 | 0 x8 | 0 x16 | 0 x24 | 0 |
| x1 | 0 x9 | 0 x17 | 0 x25 | 0 |
| x2 | 10 x10 | 10 x18 | 0 x26 | 0 |
| x3 | 10 x11 | 1023 x19 | 0 x27 | 0 |
| x4 | 2 x12 | 0 x20 | 0 x28 | 0 |
| x5 | 0 x13 | 0 x21 | 0 x29 | 0 |
| x6 | 0 x14 | 1023 x22 | 0 x30 | 0 |
| x7 | 0 x15 | 0 x23 | 0 x31 | 48 |

Рисунок № 7 - Результат работы

Пример № 2:

```

1 string evere hello = "Hello";
2 string evere helloWorld = hello . " world!";
3 print(hello);
4 print(helloWorld);

```

Рисунок № 8 - Вывод строки и конкатенация строк

#AST:
 PROGRAM


```

VAR_DECL: hello (type: string, global: yes)
INIT:
    LITERAL: ""Hello"" (type: string)
VAR_DECL: helloWorld (type: string, global: yes)
INIT:
    BIN_OP: .
    LEFT:
        ID: hello
    RIGHT:
        LITERAL: "" world!"" (type: string)
PRINT
    EXPRESSION:
        ID: hello
PRINT
    EXPRESSION:
        ID: helloWorld

```

Reload

Step once

Run

Reload & Run

Stop

```

# Initialize hello (address 1000)
li x2, 1001 # start address
li x1, 72 # 'H'
sw x2, 0, x1 # save char
addi x2, x2, 1 # next position
li x1, 101 # 'e'
sw x2, 0, x1 # save char
addi x2, x2, 1 # next position
li x1, 108 # 'l'
sw x2, 0, x1 # save char
addi x2, x2, 1 # next position
li x1, 108 # 'l'
sw x2, 0, x1 # save char
addi x2, x2, 1 # next position
li x1, 111 # 'o'
sw x2, 0, x1 # save char
addi x2, x2, 1 # next position
li x1, 0 # null terminator
sw x2, 0, x1
li x1, 1001
li x2, 1000 # Load address of hello
sw x2, 0, x1 # Store value to variable
# Initialize helloWorld (address 1018)
li x2, 1000 # Load address of hello
lw x31, x2, 0 # Load variable value
add x5, x31, x0
li x2, 1019 # start address
li x1, 32 # ' '

```

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|------------|-------------------|----------------|
| 0030 | 3F80 0313 | 1068499731 | addi x6, x0, 1019 | x6 := 0 + 1019 |
| 0031 | 3E80 0113 | 1048576275 | addi x2, x0, 1000 | x2 := 0 + 1000 |
| 0032 | 0001 2F83 | 77699 | lw x31, x2, 0 | x31 := [x2] |
| 0033 | 000F 82B3 | 1016499 | add x5, x31, x0 | x5 := x31 + 0 |
| 0034 | 40E0 0113 | 1088422163 | addi x2, x0, 1038 | x2 := 0 + 1038 |
| 0035 | 0200 0093 | 33554579 | addi x1, x0, 32 | x1 := 0 + 32 |
| 0036 | 0011 2023 | 1122339 | sw x2, 0, x1 | [x2] := x1 |
| 0037 | 0011 0113 | 1114387 | addi x2, x2, 1 | x2 := x2 + 1 |
| 0038 | 0770 0093 | 124780691 | addi x1, x0, 119 | x1 := 0 + 119 |
| 0039 | 0011 2023 | 1122339 | sw x2, 0, x1 | [x2] := x1 |
| 003A | 0011 0113 | 1114387 | addi x2, x2, 1 | x2 := x2 + 1 |
| 003B | 06F0 0093 | 116392083 | addi x1, x0, 111 | x1 := 0 + 111 |
| 003C | 0011 2023 | 1122339 | sw x2, 0, x1 | [x2] := x1 |
| 003D | 0011 0113 | 1114387 | addi x2, x2, 1 | x2 := x2 + 1 |
| 003E | 0720 0093 | 119537811 | addi x1, x0, 114 | x1 := 0 + 114 |
| 003F | 0011 2023 | 1122339 | sw x2, 0, x1 | [x2] := x1 |

Address Hex Decimal Command Explanation

<<

<

>

>>

Program input

Clear output

Hello
Hello world!

Рисунок № 9 - Результат работы

Пример № 3:

```

1  int lim x = 10;
2  int lim y = 3;
3  int lim d = 4;
4  int lim g = x / 3;
5  print(x * y / d - g + 5);|

```

Рисунок № 10 - Выполнение нескольких математических операций в одном выражении

#AST:

PROGRAM

VAR_DECL: x (type: int, global: no)

INIT:

LITERAL: 10 (type: int)

VAR_DECL: y (type: int, global: no)

INIT:

LITERAL: 3 (type: int)

VAR_DECL: d (type: int, global: no)

INIT:

LITERAL: 4 (type: int)

VAR_DECL: g (type: int, global: no)

INIT:

BIN_OP: /

LEFT:

ID: x

RIGHT:

LITERAL: 3 (type: int)

PRINT

EXPRESSION:

BIN_OP: +

LEFT:

BIN_OP: -

LEFT:

BIN_OP: /

LEFT:

BIN_OP: *

LEFT:

ID: x

RIGHT:

ID: y

RIGHT:

ID: d

RIGHT:

ID: g

RIGHT:

LITERAL: 5 (type: int)

Reload

Step once

Run

Reload & Run

Stop

```

# Initialize x (address 1000)
li x1, 10
li x2, 1000 # Load address of x
sw x2, 0, x1 # Store value to variable
# Initialize y (address 1001)
li x1, 3
li x2, 1001 # Load address of y
sw x2, 0, x1 # Store value to variable
# Initialize d (address 1002)
li x1, 4
li x2, 1002 # Load address of d
sw x2, 0, x1 # Store value to variable
# Initialize g (address 1003)
li x2, 1000 # Load address of x
lw x31, x2, 0 # Load variable value
add x3, x31, x0
li x4, 3
div x1, x3, x4
li x2, 1003 # Load address of g
sw x2, 0, x1 # Store value to variable
li x2, 1000 # Load address of x
lw x31, x2, 0 # Load variable value
add x1, x31, x0
li x2, 1001 # Load address of y
lw x31, x2, 0 # Load variable value
add x2, x31, x0
mul x3, x1, x2
li x2, 1002 # Load address of d

```

Program input

9

Clear output

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|-----------|------------------|---------------------------------|
| 0030 | 0006 8263 | 426595 | beq x13, x0, 2 | if x13 == x0 then pc := pc + 2 |
| 0031 | 02D0 0F93 | 47189907 | addi x31, x0, 45 | x31 := 0 + 45 |
| 0032 | 004F 8073 | 5210227 | ewrite x31 | WRITE x31 |
| 0033 | 0017 0713 | 1509139 | addi x14, x14, 1 | x14 := x14 + 1 |
| 0034 | 0007 2F83 | 470915 | lw x31, x14, 0 | x31 := [x14] |
| 0035 | 004F 8073 | 5210227 | ewrite x31 | WRITE x31 |
| 0036 | FE87 1CE3 | -21553949 | bne x14, x11, -4 | if x14 != x11 then pc := pc - 4 |
| 0037 | 00A0 0113 | 10486035 | addi x2, x0, 10 | x2 := 0 + 10 |
| 0038 | 0041 0073 | 4259955 | ewrite x2 | WRITE x2 |
| 0039 | 0010 0073 | 1048691 | ebreak | HALT |
| 003A | 0000 0000 | 0 | | |
| 003B | 0000 0000 | 0 | | |
| 003C | 0000 0000 | 0 | | |
| 003D | 0000 0000 | 0 | | |
| 003E | 0000 0000 | 0 | | |
| 003F | 0000 0000 | 0 | | |

| Address | Hex | Decimal | Command | Explanation |
|---------|--------|----------|---------|-------------|
| pc | 003A | | | |
| x0 | 0 x8 | 0 x16 | 9 x24 | 0 |
| x1 | 9 x9 | 0 x17 | 0 x25 | 0 |
| x2 | 10 x10 | 10 x18 | 0 x26 | 0 |
| x3 | 4 x11 | 1023 x19 | 0 x27 | 0 |
| x4 | 5 x12 | 0 x20 | 0 x28 | 0 |
| x5 | 0 x13 | 0 x21 | 0 x29 | 0 |
| x6 | 0 x14 | 1023 x22 | 0 x30 | 0 |
| x7 | 0 x15 | 0 x23 | 0 x31 | 57 |

Рисунок № 11 - Результат работы

Пример № 4:

```

1  int evere f_1 = 1;
2  int evere f_2 = 1;
3  int evere n = 8;
4  int evere f_res = 1;
5  round i in range(3, n + 1, 1){
6      f_res = f_1 + f_2;
7      f_1 = f_2;
8      f_2 = f_res;
9  }
10 print(f_res);|

```

Рисунок № 12 - Поиск n-го элемента ряда Фибоначчи

#AST:

PROGRAM

VAR_DECL: f_1 (type: int, global: yes)

INIT:

LITERAL: 1 (type: int)

VAR_DECL: f_2 (type: int, global: yes)

INIT:

LITERAL: 1 (type: int)

VAR_DECL: n (type: int, global: yes)

```
INIT:
  LITERAL: 8 (type: int)
VAR_DECL: f_res (type: int, global: yes)
INIT:
  LITERAL: 1 (type: int)
ROUND: i
START:
  LITERAL: 3 (type: int)
END:
  BIN_OP: +
  LEFT:
    ID: n
  RIGHT:
    LITERAL: 1 (type: int)
STEP:
  LITERAL: 1 (type: int)
BODY:
  BLOCK
    ASSIGN: f_res
    VALUE:
      BIN_OP: +
      LEFT:
        ID: f_1
      RIGHT:
        ID: f_2
    ASSIGN: f_1
    VALUE:
      ID: f_2
    ASSIGN: f_2
    VALUE:
      ID: f_res
  PRINT
  EXPRESSION:
    ID: f_res
```

Reload

Step once

Run

Reload & Run

Stop

```

# Initialize f_1 (address 1000)
li x1, 1
li x2, 1000 # Load address of f_1
sw x2, 0, x1 # Store value to variable
# Initialize f_2 (address 1001)
li x1, 1
li x2, 1001 # Load address of f_2
sw x2, 0, x1 # Store value to variable
# Initialize n (address 1002)
li x1, 8
li x2, 1002 # Load address of n
sw x2, 0, x1 # Store value to variable
# Initialize f_res (address 1003)
li x1, 1
li x2, 1003 # Load address of f_res
sw x2, 0, x1 # Store value to variable
# Begin round loop
li x1, 3
add x20, x1, x0 # Save iterator to dedicated register
li x2, 1004 # Load address of i
sw x2, 0, x20 # Store initial value
li x2, 1002 # Load address of n
lw x31, x2, 0 # Load variable value
add x3, x31, x0
li x4, 1
add x1, x3, x4
add x21, x1, x0 # Save end value to dedicated register
li x1, 1

```

Program Input

21

Clear output

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|------------|--------------------|--------------------------------|
| 0030 | 0141 2023 | 21045283 | sw x2, 0, x20 | [x2] := x20 |
| 0031 | 015A 22B3 | 22684339 | slt x5, x20, x21 | x5 := x20 < x21 |
| 0032 | FC02 96E3 | -66939165 | bne x5, x0, -26 | if x5 != x0 then pc := pc - 26 |
| 0033 | 3EB0 0113 | 1051722003 | addi x2, x0, 1003 | x2 := 0 + 1003 |
| 0034 | 0001 2F83 | 77699 | lw x31, x2, 0 | x31 := [x2] |
| 0035 | 000F 80B3 | 1015987 | add x1, x31, x0 | x1 := x31 + 0 |
| 0036 | 00A0 0513 | 10487059 | addi x10, x0, 10 | x10 := 0 + 10 |
| 0037 | 3FF0 0593 | 1072694675 | addi x11, x0, 1023 | x11 := 0 + 1023 |
| 0038 | 0000 8613 | 34323 | addi x12, x1, 0 | x12 := x1 + 0 |
| 0039 | 0000 0693 | 1683 | addi x13, x0, 0 | x13 := 0 + 0 |
| 003A | 0005 8713 | 362259 | addi x14, x11, 0 | x14 := x11 + 0 |
| 003B | 0006 5263 | 414307 | bge x12, x0, 2 | if x12 >= x0 then pc := pc + 2 |
| 003C | 0010 0693 | 1050259 | addi x13, x0, 1 | x13 := 0 + 1 |
| 003D | 40C0 0633 | 1086326323 | sub x12, x0, x12 | x12 := 0 - x12 |
| 003E | 02A6 47B3 | 44451763 | div x15, x12, x10 | x15 := x12 / x10 |
| 003F | 02A6 6833 | 44460083 | rem x16, x12, x10 | x16 := x12 % x10 |

| Address | Hex | Decimal | Command | Explanation |
|---------|--------|----------|---------|-------------|
| pc | 004F | | | |
| x0 | 0 x8 | 0 x16 | 2 x24 | 0 |
| x1 | 21 x9 | 0 x17 | 0 x25 | 0 |
| x2 | 10 x10 | 10 x18 | 0 x26 | 0 |
| x3 | 8 x11 | 1023 x19 | 0 x27 | 0 |
| x4 | 13 x12 | 0 x20 | 9 x28 | 0 |
| x5 | 0 x13 | 0 x21 | 9 x29 | 0 |
| x6 | 0 x14 | 1023 x22 | 1 x30 | 0 |
| x7 | 0 x15 | 0 x23 | 0 x31 | 49 |

Рисунок № 13 - Результат работы

Пример № 5:

```

1  int evere x = 100;
2  int evere y = 15;
3  while(x > 0 and y > 0){
4      if (x > y){
5          x = x % y;
6      }
7      else {
8          y = y % x;
9      }
10 }
11 if (x > y){
12     print(x);
13 }
14 else {
15     print(y);
16 }

```

Рисунок № 14 - Поиск НОД

#AST:

PROGRAM

VAR_DECL: x (type: int, global: yes)

INIT:

LITERAL: 100 (type: int)

VAR_DECL: y (type: int, global: yes)

INIT:

LITERAL: 15 (type: int)

WHILE

CONDITION:

BIN_OP: >

LEFT:

ID: x

RIGHT:

BIN_OP: and

LEFT:

LITERAL: 0 (type: int)

RIGHT:

BIN_OP: >

LEFT:

ID: y

RIGHT:

LITERAL: 0 (type: int)

BODY:

BLOCK

IF

CONDITION:

BIN_OP: >

LEFT:

ID: x

RIGHT:

ID: y

THEN:

BLOCK

ASSIGN: x

VALUE:

BIN_OP: %

LEFT:

ID: x

RIGHT:

ID: y

ELSE:

BLOCK

ASSIGN: y

VALUE:

BIN_OP: %

LEFT:

ID: y

RIGHT:

```

        ID: x
IF
CONDITION:
    BIN_OP: >
    LEFT:
        ID: x
    RIGHT:
        ID: y
THEN:
BLOCK
PRINT
    EXPRESSION:
        ID: x
ELSE:
BLOCK
PRINT
    EXPRESSION:
        ID: y

```

Reload

Step once

Run

Reload & Run

Stop

```

# Initialize x (address 1000)
li x1, 100
li x2, 1000 # Load address of x
sw x2, 0, x1 # Store value to variable
# Initialize y (address 1001)
li x1, 15
li x2, 1001 # Load address of y
sw x2, 0, x1 # Store value to variable
# Begin while-loop
_while_0:
li x2, 1000 # Load address of x
lw x31, x2, 0 # Load variable value
add x3, x31, x0
li x1, 0
li x2, 1001 # Load address of y
lw x31, x2, 0 # Load variable value
add x1, x31, x0
li x2, 0
slt x2, x2, x1
# Logical AND
sne x5, x1, x0
sne x6, x2, x0
and x4, x5, x6
slt x1, x4, x3
beq x1, x0, __endwhile_1 # Exit loop if condition is false
# Begin if-statement
li x2, 1000 # Load address of x
lw x31, x2, 0 # Load variable value

```

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|------------|--------------------|--------------------------------|
| 0030 | 3E80 0113 | 1048576275 | addi x2, x0, 1000 | x2 := 0 + 1000 |
| 0031 | 0001 2F83 | 77699 | lw x31, x2, 0 | x31 := [x2] |
| 0032 | 000F 8183 | 1016243 | add x3, x31, x0 | x3 := x31 + 0 |
| 0033 | 3E90 0113 | 1049624851 | addi x2, x0, 1001 | x2 := 0 + 1001 |
| 0034 | 0001 2F83 | 77699 | lw x31, x2, 0 | x31 := [x2] |
| 0035 | 000F 8233 | 1016371 | add x4, x31, x0 | x4 := x31 + 0 |
| 0036 | 0032 2083 | 3285171 | slt x1, x4, x3 | x1 := x4 < x3 |
| 0037 | 0200 8C63 | 33590371 | beq x1, x0, 28 | if x1 == x0 then pc := pc + 28 |
| 0038 | 3E80 0113 | 1048576275 | addi x2, x0, 1000 | x2 := 0 + 1000 |
| 0039 | 0001 2F83 | 77699 | lw x31, x2, 0 | x31 := [x2] |
| 003A | 000F 8083 | 1015987 | add x1, x31, x0 | x1 := x31 + 0 |
| 003B | 00A0 0513 | 10487059 | addi x10, x0, 10 | x10 := 0 + 10 |
| 003C | 3FF0 0593 | 1072694675 | addi x11, x0, 1023 | x11 := 0 + 1023 |
| 003D | 0000 8613 | 34323 | addi x12, x1, 0 | x12 := x1 + 0 |
| 003E | 0000 0693 | 1683 | addi x13, x0, 0 | x13 := 0 + 0 |
| 003F | 0005 8713 | 362259 | addi x14, x11, 0 | x14 := x11 + 0 |

<<

<

>

>>

| Address | Hex | Decimal | Command | Explanation | | | |
|---------|------|---------|---------|-------------|---|-----|----|
| pc | 0070 | | | | | | |
| x0 | 0 | x8 | 0 | x16 | 5 | x24 | 0 |
| x1 | 5 | x9 | 0 | x17 | 0 | x25 | 0 |
| x2 | 10 | x10 | 10 | x18 | 0 | x26 | 0 |
| x3 | 0 | x11 | 1023 | x19 | 0 | x27 | 0 |
| x4 | 5 | x12 | 0 | x20 | 0 | x28 | 0 |
| x5 | 1 | x13 | 0 | x21 | 0 | x29 | 0 |
| x6 | 1 | x14 | 1023 | x22 | 0 | x30 | 0 |
| x7 | 0 | x15 | 0 | x23 | 0 | x31 | 53 |

Clear output

5

Рисунок № 15 - Результат работы

Пример № 6:

```

1  // однострочный комментарий
2  /*
3  многострочный
4  комментарий
5  */
6  print(5 + 3);

```

Рисунок № 16 - Обработка комментариев

#AST:

PROGRAM

PRINT

EXPRESSION:

BIN_OP: +

LEFT:

LITERAL: 5 (type: int)

RIGHT:

LITERAL: 3 (type: int)

Reload Step once Run Reload & Run Stop

```

li x3, 5
li x4, 3
add x1, x3, x4
# Print value
addi x10, x0, 10 # divisor
addi x11, x0, 1023 # buffer end
addi x12, x1, 0 # number to print
addi x13, x0, 0 # is negative flag
addi x14, x11, 0 # buffer position
bge x12, x0, __producer_loop_0 # if positive, skip negation
addi x13, x0, 1 # set negative flag
sub x12, x0, x12 # negate number
__producer_loop_0:
div x15, x12, x10 # divide by 10
rem x16, x12, x10 # get remainder
addi x31, x16, 48 # convert to ASCII
sw x14, 0, x31 # store digit
addi x14, x14, -1 # move buffer position
addi x12, x15, 0 # update number
bne x12, x0, __producer_loop_0 # continue if not zero
beq x13, x0, __after_minus_1 # skip minus if positive
addi x31, x0, 45 # minus sign ASCII
ewrite x31 # print minus
__after_minus_1:
addi x14, x14, 1 # adjust buffer position
lw x31, x14, 0 # load digit
ewrite x31 # print digit
bne x14, x11, __after_minus_1 # continue if not end

```

Program input

Clear output

| Address | Hex | Decimal | Command | Explanation |
|---------|-----------|------------|--------------------|--------------------------------|
| 0000 | 0050 0193 | 5243283 | addi x3, x0, 5 | x3 := 0 + 5 |
| 0001 | 0030 0213 | 3146259 | addi x4, x0, 3 | x4 := 0 + 3 |
| 0002 | 0041 8083 | 4292787 | add x1, x3, x4 | x1 := x3 + x4 |
| 0003 | 00A0 0513 | 10487059 | addi x10, x0, 10 | x10 := 0 + 10 |
| 0004 | 3FF0 0593 | 1072694675 | addi x11, x0, 1023 | x11 := 0 + 1023 |
| 0005 | 0000 8613 | 34323 | addi x12, x1, 0 | x12 := x1 + 0 |
| 0006 | 0000 0693 | 1683 | addi x13, x0, 0 | x13 := 0 + 0 |
| 0007 | 0005 8713 | 362259 | addi x14, x11, 0 | x14 := x11 + 0 |
| 0008 | 0006 5263 | 414307 | bge x12, x0, 2 | if x12 >= x0 then pc := pc + 2 |
| 0009 | 0010 0693 | 1050259 | addi x13, x0, 1 | x13 := 0 + 1 |
| 000A | 40C0 0633 | 1086326323 | sub x12, x0, x12 | x12 := 0 - x12 |
| 000B | 02A6 47B3 | 44451763 | div x15, x12, x10 | x15 := x12 / x10 |
| 000C | 02A6 6833 | 44460083 | rem x16, x12, x10 | x16 := x12 % x10 |
| 000D | 0308 0F93 | 50859923 | addi x31, x16, 48 | x31 := x16 + 48 |
| 000E | 01F7 2023 | 32972835 | sw x14, 0, x31 | [x14] := x31 |
| 000F | FFF7 0713 | -588013 | addi x14, x14, -1 | x14 := x14 + (-1) |

| Address | Hex | Decimal | Command | Explanation |
|---------|--------|---------|---------|-------------|
| pc | 001C | | | |
| x0 | 0 x8 | 0 | x16 | 8 x24 |
| x1 | 8 x9 | 0 | x17 | 0 x25 |
| x2 | 10 x10 | 10 | x18 | 0 x26 |
| x3 | 5 x11 | 1023 | x19 | 0 x27 |
| x4 | 3 x12 | 0 | x20 | 0 x28 |
| x5 | 0 x13 | 0 | x21 | 0 x29 |
| x6 | 0 x14 | 1023 | x22 | 0 x30 |
| x7 | 0 x15 | 0 | x23 | 0 x31 |

Рисунок № 17 - Результат работы

Обработка ошибок

Пример № 1:

```

1  int evere x = 5;
2  if (x > 10){
3  |    int lim y = 10;
4  |  }
5  print(y);|

```


Рисунок № 18 - Область видимости переменных

```
Error: Scope violation in examples/_7.txt:5:10: Cannot access local variable 'y' from global scope
Critical errors found during code generation. Output aborted.
Error generating RISC code
```

Рисунок № 19 - Результат работы

Пример № 2:

```
1  int evere x = 10;
2  print(d);|
```

Рисунок № 20 - Использование не декларированной переменной

```
Error: Undefined variable in examples/_8.txt:2:10: Variable 'd' is not defined
Critical errors found during code generation. Output aborted.
Error generating RISC code
```

Рисунок № 21 - Результат работы

Пример № 3:

```
1  int evere x = 10;
2  print(x / 0);
```

Рисунок № 22 - Деление на ноль

```
Error: Division by zero in examples/_9.txt:2:14: Division by zero detected at compile-time
Critical errors found during code generation. Output aborted.
Error generating RISC code
```

Рисунок № 23 - Результат работы

Пример № 4:

```
1  int evere x = 10;
2  string evere x = "Hello";
```

Рисунок № 24 - Повторное объявление переменной

```
Error: Variable redeclaration in examples/_10.txt:2:26: Variable 'x' is already declared in this scope  
Critical errors found during code generation. Output aborted.  
Error generating RISC code
```

Рисунок № 25 - Результат работы

Пример № 5:

```
1  int evere x = 10;  
2  string evere y = x;  
3  print(y);|
```

Рисунок № 26 - Выполнение операций с различными типами переменных

```
Error: Type mismatch in examples/_11.txt:3:10: Incompatible types: 'string' and 'int' for operation '='  
Critical errors found during code generation. Output aborted.  
Error generating RISC code
```

Рисунок № 27 - Результат работы