Университет ИТМО

Факультет ПИиКТ

Системное программное обеспечение

Лабораторная работа №3

Работу выполнил:

Абузов Ярослав

Группа:

P4114

Вариант:

Статическая; Двухадресный код; 3: code, constants, data

Преподаватель:

Кореньков Ю. Д.

Санкт-Петербург

2024

Содержание

1 Цели

Реализовать формирование линейного кода в терминах некоторого набора инструкций посредством анализа графа потока управления для набора подпрограмм. Полученный линейный код вывести в мнемонической форме в выходной текстовый файл.

2 Задачи

Подготовка к выполнению по одному из двух сценариев:

1. Составить описание виртуальной машины с набором инструкций и моделью памяти по варианту

a. Изучить нотацию для записи определений целевых архитектур

b. Составить описание ВМ в соответствии с вариантом

i. Описание набор регистров и банков памяти

ii. Описать набор инструкций: для каждой инструкции задать структуру операционного кода, содержащего описание операндов и набор операций, изменяющих состояние ВМ

1. Описать инструкции перемещения данных и загрузки констант

2. Описать инструкции арифметических и логических операций

3. Описать инструкции условной и безусловной передачи управления

4. Описать инструкции ввода-вывода с использованием скрытого регистра в качестве порта ввода-вывода

iii. Описать набор мнемоник, соответствующих инструкциям ВМ

c. Подготовить скрипт для запуска ассемблированного листинга с использованием описания ВМ:

i. Написать тестовый листинг с использованием подготовленных мнемоник инструкций

ii. Задействовать транслятор листинга в бинарный модуль по описанию ВМ

iii. Запустить полученный бинарный модуль на исполнение и получить результат работы

iv. Убедиться в корректности функционирования всех инструкций ВМ

2. Выбрать и изучить прикладную архитектуру системы команд существующей ВМ

a. Для выбранной ВМ:

i. Должен существовать готовый эмулятор (например qemu)

ii. Должен существовать готовый тулчейн (набор инструментов разработчика): компилятор Си, ассемблер и дизассемблер, линковщик, желательно отладчик

b. Согласовать выбор ВМ с преподавателем

c. Изучить модель памяти и набор инструкций ВМ

d. Научиться использовать тулчейн (собирать и запускать программы из листинга)

e. Подготовить скрипт для запуска ассемблированного листинга с использованием эмулятора

i. Написать тестовый листинг с использованием инструкций ВМ

ii. Задействовать ассемблер и компоновщик из тулчейна

iii. Запустить бинарный модуль на исполнение и получить результат его работы

Порядок выполнения:

1. Описать структуры данных, необходимые для представления информации об элементах образа программы (последовательностях инструкций и данных), расположенных в памяти

a. Для каждой инструкции – имя мнемоники и набор операндов в терминах данной ВМ

b. Для элемента данных – соответствующее литеральное значение или размер экземпляра типа данных в байтах

2. Реализовать модуль, формирующий образ программы в линейном коде для данного набора подпрограмм

a. Программный интерфейс модуля принимает на вход структуру данных, содержащую графы потока управления и информацию о локальных переменных и сигнатурах для набора подпрограмм, разработанную в задании 2 (п. 1.a, п. 2.b)

b. В результате работы порождается структура данных, разработанная в п. 1, содержащая описание образа программы в памяти: набор именованных элементов данных и набор именованных фрагментов линейного кода, представляющих собой алгоритмы подпрограмм

c. Для каждой подпрограммы посредством обхода узлов графа потока управления в порядке топологической сортировки (начиная с узла, являющегося первым базовым блоком алгоритма подпрограммы), сформировать набор именованных групп инструкций, включая пролог и эпилог подпрограммы (формирующие и разрушающие локальное состояние подпрограммы)

d. Для каждого базового блока в составе графа потока управления сформировать группу инструкций, соответствующих операциям в составе дерева операций

e. Использовать имена групп инструкций для формирования инструкций перехода между блоками инструкций, соответствующих узлам графа потока управления, в соответствии с дугами в нём

3. Доработать тестовую программу, разработанную в задании 2 для демонстрации работоспособности созданного модуля

a. Добавить поддержку аргумента командной строки для имени выходного файла, вывод информации о графах потока управления сделать опциональных

b. Использовать модуль, разработанный в п. 2 для формирования образа программы на основе информации, собранной в результате работы модуля, созданного в задании 2 (п. 2.b)

c. Для сформированного образа программы в линейном коде вывести в выходной файл ассемблерный листинг, содержащий мнемоническое представление инструкций и данных, как они описаны в структурах данных (п. 1), построенных разработанным модулем (пп. 2.c-e)

d. Проверить корректность решения посредством сборки сгенерированного листинга и запуска полученного бинарного модуля на эмуляторе ВМ (см. подготовка п. 1.c или п. 2.e)

4. Результаты тестирования представить в виде отчета, в который включить:

a. В части 3 привести описание разработанных структур данных

b. В части 4 описать программный интерфейс и особенности реализации разработанного модуля

c. В части 5 привести примеры исходных текстов, соответствующие ассемблерные листинги и примера вывода запущенных тестовых программ

3 Детали реализации

Для запуска собственного ASM была разработана и описана собственная ВМ, которая поддерживает следующие команды:

```
#define commandHLT(buffer) ({plain0(buffer, "HLT");})
#define commandNOP(buffer) ({plain0(buffer, "NOP");})
#define commandRET(buffer) ({plain0(buffer, "RET");})

#define commandADD(buffer, size, reg1, reg2) ({size2(buffer, "ADD", size, reg1,
reg2);})
#define commandSUB(buffer, size, reg1, reg2) ({size2(buffer, "SUB", size, reg1,
reg2);})
#define commandMUL(buffer, size, reg1, reg2) ({size2(buffer, "MUL", size, reg1,
reg2);})
#define commandDIV(buffer, size, reg1, reg2) ({size2(buffer, "DIV", size, reg1,
reg2);})
#define commandMOD(buffer, size, reg1, reg2) ({size2(buffer, "MOD", size, reg1,
reg2);})

#define commandST(buffer, size, reg1, reg2) ({size2(buffer, "ST", size, reg1,
reg2);})
#define commandSToffset(buffer, size, reg1, reg2, offset) ({sizeOffset3(buffer, "ST",
size, reg1, reg2, offset);})
#define commandLD(buffer, size, reg1, reg2) ({size2(buffer, "LD", size, reg1,
reg2);})
#define commandLDoffset(buffer, size, reg1, reg2, offset) ({sizeOffset3(buffer, "LD",
size, reg1, reg2, offset);})
#define commandLDCoffset(buffer, size, reg1, reg2, offset) ({sizeOffset3(buffer,
"LDC", size, reg1, reg2, offset);})
#define commandLDI32(buffer, reg1, value32) ({plain2(buffer, "LDI32", reg1,
value32);})
#define commandLDC64(buffer, reg1, constAddr) ({plain2(buffer, "LDC64", reg1,
constAddr);})

#define commandPUSH(buffer, reg1) ({plain1(buffer, "PUSH", reg1);})
#define commandPOP(buffer, reg1) ({plain1(buffer, "POP", reg1);})

#define commandMOV(buffer, toReg, fromReg) ({plain2(buffer, "MOV", toReg, fromReg);})
#define commandMOVT(buffer, size, toReg, fromReg) ({size2(buffer, "MOVT", size,
toReg, fromReg);})
#define commandMOVZX(buffer, size, toReg, fromReg) ({size2(buffer, "MOVZX", size,
toReg, fromReg);})

#define commandMEMCPYC(buffer, toPtr, fromPtr, size) ({plain3(buffer, "MEMCPYC",
toPtr, fromPtr, size);})

#define commandCBW(buffer, reg1, reg2) ({plain2(buffer, "CBW", reg1, reg2);})
#define commandCBD(buffer, reg1, reg2) ({plain2(buffer, "CBD", reg1, reg2);})
#define commandCBQ(buffer, reg1, reg2) ({plain2(buffer, "CBQ", reg1, reg2);})
#define commandCWD(buffer, reg1, reg2) ({plain2(buffer, "CWD", reg1, reg2);})
#define commandCWQ(buffer, reg1, reg2) ({plain2(buffer, "CWQ", reg1, reg2);})
#define commandCDQ(buffer, reg1, reg2) ({plain2(buffer, "CDQ", reg1, reg2);})

#define commandNEG(buffer, size, reg1) ({size1(buffer, "NEG", size, reg1);})
#define commandNOT(buffer, size, reg1) ({size1(buffer, "NOT", size, reg1);})
#define commandAND(buffer, size, reg1, reg2) ({size2(buffer, "AND", size, reg1,
reg2);})
```

```
#define commandOR(buffer, size, reg1, reg2) ({size2(buffer, "OR", size, reg1,
reg2);})

#define commandEQ(buffer, reg1, reg2) ({plain2(buffer, "EQ", reg1, reg2);})
#define commandNEQ(buffer, reg1, reg2) ({plain2(buffer, "NEQ", reg1, reg2);})
#define commandGR(buffer, reg1, reg2) ({plain2(buffer, "GR", reg1, reg2);})
#define commandLE(buffer, reg1, reg2) ({plain2(buffer, "LE", reg1, reg2);})
#define commandGREQ(buffer, reg1, reg2) ({plain2(buffer, "GREQ", reg1, reg2);})
#define commandLEEQ(buffer, reg1, reg2) ({plain2(buffer, "LEEQ", reg1, reg2);})

#define commandJMP(buffer, label) ({plain1(buffer, "JMP", label);})
#define commandJZ(buffer, label) ({plain1(buffer, "JZ", label);})
#define commandJNZ(buffer, label) ({plain1(buffer, "JNZ", label);})
#define commandJEQ(buffer, reg1, reg2, label) ({plain3(buffer, "JEQ", reg1, reg2,
label);})
#define commandJNEQ(buffer, reg1, reg2, label) ({plain3(buffer, "JNEQ", reg1, reg2,
label);})
#define commandJGR(buffer, reg1, reg2, label) ({plain3(buffer, "JGR", reg1, reg2,
label);})
#define commandJLE(buffer, reg1, reg2, label) ({plain3(buffer, "JLE", reg1, reg2,
label);})
#define commandJGREQ(buffer, reg1, reg2, label) ({plain3(buffer, "JGREQ", reg1, reg2,
label);})
#define commandJLEEQ(buffer, reg1, reg2, label) ({plain3(buffer, "JLEEQ", reg1, reg2,
label);})

#define commandCALL(buffer, label) ({plain1(buffer, "CALL", label);})
#define commandENTER(buffer, size) ({plain1(buffer, "ENTER", size);})
#define commandLEAVE(buffer, size) ({plain1(buffer, "LEAVE", size);})
```

ВМ содержит 8 регистров общего назначения, регистр IP, SP, BP, два буферных регистра, регистр адреса, регистр возврата, регистры IO, регистр указателя аллокатора, регистр флагов.

ВМ поддерживает в нужных командах работу с данными разных размеров (8, 16, 32, 64 бита). ВМ содержит три банка памяти – для данных, для констант, для кода. Все команды занимают 64 бита.

Компилятор совершает следующий pipeline:

1) Построение AST с выводом ошибок
2) Построение CFG и OT (+ сбор информации о символах) с выводом ошибок
3) Проверка типизации и вывод ошибок (типизация строгая, как в Java)
4) Расчёт необходимого количества регистров (или необходимость пролива в стек)

5) Распределение регистров, задание смещений адресам, подготовка образа памяти
6) Генерация ASM листинга

Этапы достаточно большие подробности в коде. Из реализованного в том числе:

1) Система внутренних функций
2) Линейный аллокатор для массивов
3) Работа со строками
4) Некоторые cast'ы
5) Передача любого количества аргументов
6) Работоспособная рекурсия
7) Корректный IO

Из того, что еще нужно реализовать:

1) Все виды кастов (просто прописать внутри нужную комбинацию команд, обвязка готова)
2) Многомерные массивы (проблема известна, связана с интерпретацией lvalue в индексации)
3) Починить кое-что в условных переходах

4 Примеры

```
int main(string[] args) {

    printNumber(123);
    __writeChar('\n');

    char[] arr;
    __alloc("arr", 4);

    arr[1] = '1';
    arr[2] = 'h';

    __writeChar(arr[1]);
    __writeChar('\n');
    __writeChar(arr[2]);
    __writeChar('\n');
```

```
    char[] arr2 = retArr();

    __writeChar(arr2[6]);
    __writeChar('\n');
    __writeChar(arr2[5]);
    __writeChar('\n');

    recursive(0);

    string s = "Hello, World!";
    char c = s[0];

    int i = 0;
    int l = strlength(s);
    while (i != l) {
        __writeChar(s[i]);
        i = i + 1;
    }

    __writeChar('\n');

    int ret = test(1, 2, 3, test(1, 1, 1, test(0, 0, 0, test(0, 0, 0, 1))));
    if (ret == 10) {
        i = 0;
        l = 4;
        while (i != l) {
            __writeChar("YES"[i]);
            i = i + 1;
        }
    } else if (ret == 9) {
        i = 0;
        l = 3;
        while (i != l) {
            __writeChar("NO"[i]);
            i = i + 1;
        }
    }

    __writeChar('\n'); // \n

    i = 0;
    l = 3;
    while (i != l) {
        __writeChar(retString()[i]);
        i = i + 1;
    }

    __writeChar('\n'); // \n

    char in = __readChar();
    __writeChar(in);
    __writeChar('\n');
    __writeChar('a');
    __writeChar('\n');

    0;
}

int test(int a, int b, int c, int d) {
    int r = a + b + c + d;
```

```
    r;
}

char[] retArr() {
    char[] arr2;
    __alloc("arr2", 8);

    arr2[6] = '2';
    arr2[5] = 'k';
    arr2;
}

string retString() {
    "lol";
}

int recursive(int i) {
    if (i < 8) {
        __writeChar('l');
        __writeChar('\n');
        recursive(i + 1);
    }
    0;
}
```

Программа выше работает с IO, массивами, содержит рекурсию, вызывает функции в качестве аргументов и т.д.

```
    [section codeM]

  JMP main
recursive:

.BB0:
  ENTER 0
.BB1:
  LD d R0, [BP, 16]
  LDI32 R1, 8
  LE R0, R1
  JZ .BB3
  JNZ .BB2

.BB2:
  LDI32 R0, 0
  MOV RT, R0
  JMP .BB4

.BB3:
  LDI32 R0, 0x6C
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
```

```
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, 16]
  LDI32 R1, 1
  ADD d R0, R1
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  CALL recursive
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  JMP .BB2

.BB4:
  LEAVE 0
  RET

retString:

.BB0:
```

```
  ENTER 0
.BB1:
  LDI32 R0, retString_const0
  MOV RT, R0
  JMP .BB2

.BB2:
  LEAVE 0
  RET

retArr:

.BB0:
  ENTER 1
.BB1:
  LDI32 AR, retArr_const1
  LDC64 R1, AR
  LDI32 R0, retArr_const0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R1
  PUSH R0
  POP R0
  POP R1
  ADD q ALR, R1
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD b R0, [BP, -8]
  LDI32 R1, 6
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDI32 R1, 0x32
  ST q R1, R0
  LD b R0, [BP, -8]
  LDI32 R1, 5
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDI32 R1, 0x6B
  ST q R1, R0
  LD b R0, [BP, -8]
  MOV RT, R0
  JMP .BB2

.BB2:
  LEAVE 1
  RET
```

```
test:

.BB0:
  ENTER 1
.BB1:
  LD d R0, [BP, 16]
  LD d R1, [BP, 24]
  ADD d R0, R1
  LD d R1, [BP, 32]
  ADD d R0, R1
  LD d R1, [BP, 40]
  ADD d R0, R1
  ST d R0, [BP, -8]
  LD d R0, [BP, -8]
  MOV RT, R0
  JMP .BB2

.BB2:
  LEAVE 1
  RET

main:

.BB0:
  ENTER 8
.BB1:
  LDI32 R0, 123
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  CALL printNumber
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
```

```
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 AR, main_const3
LDC64 R1, AR
LDI32 R0, main_const2
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R1
PUSH R0
POP R0
POP R1
ADD q ALR, R1
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LD b R0, [BP, -24]
LDI32 R1, 1
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LDI32 R1, 0x31
ST q R1, R0
LD b R0, [BP, -24]
LDI32 R1, 2
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LDI32 R1, 0x68
ST q R1, R0
LD b R0, [BP, -24]
LDI32 R1, 1
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LD q R0, R0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
```

```
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0x0A
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LD b R0, [BP, -24]
LDI32 R1, 2
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LD q R0, R0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0x0A
MOV BR1, SP
PUSH R0
```

```
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
CALL retArr
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
ST b RT, [BP, -32]
LD b R0, [BP, -32]
LDI32 R1, 6
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LD q R0, R0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
```

```
POP R1
POP R0
LDI32 R0, 0x0A
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LD b R0, [BP, -32]
LDI32 R1, 5
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LD q R0, R0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0x0A
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
```

```
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
CALL recursive
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, main_const4
ST q R0, [BP, -40]
LD q R0, [BP, -40]
LDI32 R1, 0
LDI32 BR2, 8
MUL q R1, BR2
ADD q R0, R1
LDC64 R0, R0
ST b R0, [BP, -48]
LDI32 R0, 0
ST d R0, [BP, -56]
LD q R0, [BP, -40]
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
CALL strlength
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
ST d RT, [BP, -64]
```

```
  JMP .BB2

.BB2:
  LD d R0, [BP, -56]
  LD d R1, [BP, -64]
  NEQ R0, R1
  JZ .BB4
  JNZ .BB3

.BB3:
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R3, 1
  LDI32 R2, 0
  LDI32 R1, 0
  LDI32 R0, 0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R3
  PUSH R2
  PUSH R1
  PUSH R0
  CALL test
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R2, 0
  LDI32 R1, 0
  LDI32 R0, 0
  MOV BR1, SP
```

```
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH RT
PUSH R2
PUSH R1
PUSH R0
CALL test
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R2, 1
LDI32 R1, 1
LDI32 R0, 1
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH RT
PUSH R2
PUSH R1
PUSH R0
CALL test
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R2, 3
LDI32 R1, 2
LDI32 R0, 1
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH RT
PUSH R2
PUSH R1
```

```
  PUSH R0
  CALL test
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  ST d RT, [BP, -16]
  JMP .BB5

.BB4:
  LD q R0, [BP, -40]
  LD d R1, [BP, -56]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDC64 R0, R0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -56]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -56]
  JMP .BB2

.BB5:
  LD d R0, [BP, -16]
  LDI32 R1, 10
  EQ R0, R1
  JZ .BB7
  JNZ .BB8

.BB6:
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
```

```
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R0, 0
  ST d R0, [BP, -56]
  LDI32 R0, 3
  ST d R0, [BP, -64]
  JMP .BB18

.BB7:
  LDI32 R0, 0
  ST d R0, [BP, -56]
  LDI32 R0, 4
  ST d R0, [BP, -64]
  JMP .BB9

.BB8:
  JMP .BB12

.BB9:
  LD d R0, [BP, -56]
  LD d R1, [BP, -64]
  NEQ R0, R1
  JZ .BB11
  JNZ .BB10

.BB10:
  JMP .BB6

.BB11:
  LDI32 R0, main_const1
  LD d R1, [BP, -56]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDC64 R0, R0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
```

```
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -56]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -56]
  JMP .BB9

.BB12:
  LD d R0, [BP, -16]
  LDI32 R1, 9
  EQ R0, R1
  JZ .BB14
  JNZ .BB13

.BB13:
  JMP .BB6

.BB14:
  LDI32 R0, 0
  ST d R0, [BP, -56]
  LDI32 R0, 3
  ST d R0, [BP, -64]
  JMP .BB15

.BB15:
  LD d R0, [BP, -56]
  LD d R1, [BP, -64]
  NEQ R0, R1
  JZ .BB17
  JNZ .BB16

.BB16:
  JMP .BB13

.BB17:
  LDI32 R0, main_const0
  LD d R1, [BP, -56]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDC64 R0, R0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
```

```
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -56]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -56]
  JMP .BB15

.BB18:
  LD d R0, [BP, -56]
  LD d R1, [BP, -64]
  NEQ R0, R1
  JZ .BB20
  JNZ .BB19

.BB19:
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  MOV RT, IN
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  ST b RT, [BP, -8]
  LD b R0, [BP, -8]
  MOV BR1, SP
```

```
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0x0A
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
LDI32 R0, 0x61
MOV BR1, SP
PUSH R0
PUSH R1
PUSH R2
PUSH R3
PUSH R4
PUSH R5
PUSH R6
PUSH R7
PUSH R0
POP R0
MOV OUT, R0
POP R7
POP R6
POP R5
POP R4
POP R3
POP R2
POP R1
POP R0
```

```
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R0, 0
  MOV RT, R0
  JMP .BB21

.BB20:
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  CALL retString
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -56]
  LDI32 BR2, 8
  MUL q R0, BR2
  ADD q RT, R0
  LDC64 RT, RT
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH RT
  POP R0
  MOV OUT, R0
```

```
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -56]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -56]
  JMP .BB18

.BB21:
  LEAVE 8
  HLT

printNumber:

.BB0:
  ENTER 3
.BB1:
  LD d R0, [BP, 16]
  ST d R0, [BP, -24]
  JMP .BB2

.BB2:
  LD d R0, [BP, -24]
  LDI32 R1, 0
  LE R0, R1
  JZ .BB4
  JNZ .BB3

.BB3:
  LD d R0, [BP, -24]
  LDI32 R1, 0
  EQ R0, R1
  JZ .BB6
  JNZ .BB7

.BB4:
  LDI32 R0, 0x2D
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
```

```
  POP R1
  POP R0
  LD d R0, [BP, -24]
  NEG d R0
  ST d R0, [BP, -24]
  JMP .BB3

.BB5:
  LDI32 R0, 0
  MOV RT, R0
  JMP .BB14

.BB6:
  LDI32 R0, 0x30
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  JMP .BB5

.BB7:
  LDI32 AR, printNumber_const1
  LDC64 R1, AR
  LDI32 R0, printNumber_const0
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R1
  PUSH R0
  POP R0
  POP R1
  ADD q ALR, R1
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
```

```
  POP R0
  LDI32 R0, 0
  ST d R0, [BP, -16]
  JMP .BB8

.BB8:
  LD d R0, [BP, -24]
  LDI32 R1, 0
  GR R0, R1
  JZ .BB10
  JNZ .BB9

.BB9:
  LD d R0, [BP, -16]
  LDI32 R1, 1
  SUB d R0, R1
  ST d R0, [BP, -16]
  JMP .BB11

.BB10:
  LD b R0, [BP, -8]
  LD d R1, [BP, -16]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LD d R1, [BP, -24]
  LDI32 R2, 10
  MOD d R1, R2
  LDI32 R2, 48
  OR q R1, R2
  ST q R1, R0
  LD d R0, [BP, -16]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -16]
  LD d R0, [BP, -24]
  LDI32 R1, 10
  DIV d R0, R1
  ST d R0, [BP, -24]
  JMP .BB8

.BB11:
  LD d R0, [BP, -16]
  LDI32 R1, -1
  NEQ R0, R1
  JZ .BB13
  JNZ .BB12

.BB12:
  JMP .BB5

.BB13:
  LD b R0, [BP, -8]
  LD d R1, [BP, -16]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LD q R0, R0
  MOV BR1, SP
  PUSH R0
  PUSH R1
```

```
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LD d R0, [BP, -16]
  LDI32 R1, 1
  SUB d R0, R1
  ST d R0, [BP, -16]
  JMP .BB11

.BB14:
  LEAVE 3
  RET

println:

.BB0:
  ENTER 0
.BB1:
  LDI32 R0, 0x0A
  MOV BR1, SP
  PUSH R0
  PUSH R1
  PUSH R2
  PUSH R3
  PUSH R4
  PUSH R5
  PUSH R6
  PUSH R7
  PUSH R0
  POP R0
  MOV OUT, R0
  POP R7
  POP R6
  POP R5
  POP R4
  POP R3
  POP R2
  POP R1
  POP R0
  LDI32 R0, 0
  MOV RT, R0
  JMP .BB2

.BB2:
  LEAVE 0
  RET
```

```
strlength:

.BB0:
  ENTER 1
.BB1:
  LDI32 R0, 0
  ST d R0, [BP, -8]
  JMP .BB2

.BB2:
  LDI32 R0, 1
  LDI32 R1, 1
  EQ R0, R1
  JZ .BB4
  JNZ .BB3

.BB3:
  LD d R0, [BP, -8]
  MOV RT, R0
  JMP .BB8

.BB4:
  JMP .BB5

.BB5:
  LD q R0, [BP, 16]
  LD d R1, [BP, -8]
  LDI32 BR2, 8
  MUL q R1, BR2
  ADD q R0, R1
  LDC64 R0, R0
  LDI32 R1, 0x00
  EQ R0, R1
  JZ .BB7
  JNZ .BB6

.BB6:
  LD d R0, [BP, -8]
  LDI32 R1, 1
  ADD d R0, R1
  ST d R0, [BP, -8]
  JMP .BB2

.BB7:
  JMP .BB3

.BB8:
  LEAVE 1
  RET


    [section constantsM]


retString_const0:
  dq 0x6C
  dq 0x6F
  dq 0x6C
  dq 0x00
retArr_const0:
  dq 0x61
```

```
  dq 0x72
  dq 0x72
  dq 0x32
  dq 0x00
retArr_const1:
  dq 8

main_const0:
  dq 0x4E
  dq 0x4F
  dq 0x00
main_const1:
  dq 0x59
  dq 0x45
  dq 0x53
  dq 0x00
main_const2:
  dq 0x61
  dq 0x72
  dq 0x72
  dq 0x00
main_const3:
  dq 4

main_const4:
  dq 0x48
  dq 0x65
  dq 0x6C
  dq 0x6C
  dq 0x6F
  dq 0x2C
  dq 0x20
  dq 0x57
  dq 0x6F
  dq 0x72
  dq 0x6C
  dq 0x64
  dq 0x21
  dq 0x00
printNumber_const0:
  dq 0x62
  dq 0x75
  dq 0x66
  dq 0x66
  dq 0x65
  dq 0x72
  dq 0x00
printNumber_const1:
  dq 10
```

Сгенерированный листинг на ASM.

5 Выводы

В ходе выполнения задания были разработан «полноценный» компилятор для собственного языка со статической типизацией под собственную архитектуру VM со своей системой команд.