УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.04.04 Программная инженерия

Дисциплина «Системное программное обеспечение»

Лабораторная работа №3

Вариант 12

Студент

Данилов П. Ю.

P4114

Преподаватель

Кореньков Ю. Д.

Санкт-Петербург, 2025 г.

Цель

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.

Задачи

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

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

Описание работы

Текстовый файл с исходным текстом разбираемой программы. Пример содержимого файла:

```
def sum(arg1 (of int), arg2 (of int))
   arg1 + arg2;
end

def sub(arg1 (of int), arg2 (of int))
   arg1 - arg2;
end

def mul(arg1 (of int), arg2 (of int))
   arg1 * arg2;
end

def div(arg1 (of int), arg2 (of int))
   arg1 / arg2;
end

def read_num()
   i = 1;
   in = 0;
   res = 0;
   while (true) {
       in = stdin();
       if (in = 13) then {
           stdin();
           break;
       }
       res = res * 10 + (in - 40);
       i = i + 1;
   } end
   res;
end

def number_length(num (of int))
   rest = 0;
```

```
    count = 1;
    rest = num / 10;
    while (rest != 0) {
        rest = rest / 10;
        count = count + 1;
    } end
    count;
end

def print_num(num (of int))
    rest = 0;
    count = 0;
    shift = 0;
    rest = num;
    count = number_length(num) - 1;
    shift = 1;
    while (count != 0) {
        shift = shift * 10;
        count = count - 1;
    } end
    while (shift != 0) {
        stdout(rest / shift + 48);
        rest = rest % shift;
    } end
    stdout(10);
    stdout(13);
    0;
end

def main()
    a = 0;
    b = 0;
    op = 0;
    res = 0;
    while (true) {
        a = read_num();
        op = stdin();
        stdin();
        stdin();
        if op == '+' then {
            res = sum(a, b);
        }
        if op == '-' then {
            res = sub(a, b);
        }
        if op == '*' then {
            res = mul(a, b);
        }
        if op == '/' then {
            res = div(a, b);
        }
        print_num(res);
    } end
    sum(a, b);
end
```

Пример исходного текста программы на разбираемом языке

Функция, осуществляющая формирование линейного кода для вершин дерева операций :

```
void tryPrintOperationTreeNode(TreeNode *operationTree, FILE *listingFile,
Array *valuePlaceAssociations,
                               int *argumentNumber) {
   char *operationType = operationTree->type;
   if (!strcmp(operationType, "ARG")) {
       (*argumentNumber)++;
       addArgumentPlace(
               valuePlaceAssociations,
               operationTree->children[1]->value,
               operationTree->children[0]->value
       );
   } else if (!strcmp(operationType, "AS")) {
       addValuePlace(
               valuePlaceAssociations,
               operationTree->children[1]->value,
               operationTree->children[0]->value
       );
   } else if (!strcmp(operationType, "CONST")) {
       if (!strcmp(operationTree->children[0]->value, "int")) {
           fprintlnWithArg("LD R0,", operationTree->children[1]->value,
listingFile);
           fprintln("PUSH R0", listingFile);
       } else if (!strcmp(operationTree->children[0]->value, "char")) {
           char value[1];
           sprintf(value, "%d", (int)
operationTree->children[1]->value[0]);
           fprintlnWithArg("LD R0,", value, listingFile);
           fprintln("PUSH R0", listingFile);
       } else if (!strcmp(operationTree->children[0]->value, "bool")) {
           if (!strcmp(operationTree->children[1]->value, "true")) {
               fprintlnWithArg("LD R0,", "1", listingFile);
               fprintln("PUSH R0", listingFile);
           } else {
               fprintlnWithArg("LD R0,", "0", listingFile);
               fprintln("PUSH R0", listingFile);
           }
       } else {
           fprintln("EXCEPTION", listingFile);
       }
   } else if (!strcmp(operationType, "ASSIGN")) {
       tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
       ValuePlaceAssociation *valuePlace =
findValuePlace(valuePlaceAssociations, operationTree->children[0]->value);
       if (valuePlace == NULL) {
           valuePlace = addValuePlace(valuePlaceAssociations,
operationTree->children[0]->value, operationTree->children[1]->type);
       }
       char valuePlaceShift[1000];
       sprintf(valuePlaceShift, "%d", valuePlace->shiftPosition);
       fprintln("POP R0", listingFile);
       fprintlnWithArg("ST_BP R0,", valuePlaceShift, listingFile);
   } else if (!strcmp(operationType, "READ")) {
       ValuePlaceAssociation *valuePlace =
findValuePlace(valuePlaceAssociations, operationTree->children[0]->value);
       if (valuePlace == NULL) {
           char exceptionMessage[1000];
           sprintf(exceptionMessage, "value place not found by name %s",
operationTree->children[0]->value);
```

```
            printException(exceptionMessage);
            return;
        }
        char valuePlaceShift[1000];
        sprintf(valuePlaceShift, "%d", valuePlace->shiftPosition);
        fprintlnWithArg("LD_BP R0,", valuePlaceShift, listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "EQITY")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("EQ R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "NEQ")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("NEQ R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "PLUS")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("ADD R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "MINUS")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("SUB R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "MUL")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("MUL R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "DIV")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("DIV R0, R1", listingFile);
```

```
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "PERCENT")) {
        tryPrintOperationTreeNode(operationTree->children[0], listingFile,
valuePlaceAssociations, argumentNumber);
        tryPrintOperationTreeNode(operationTree->children[1], listingFile,
valuePlaceAssociations, argumentNumber);
        fprintln("POP R1", listingFile);
        fprintln("POP R0", listingFile);
        fprintln("REM R0, R1", listingFile);
        fprintln("PUSH R0", listingFile);
    } else if (!strcmp(operationType, "CALL")) {
        if (!strcmp(operationTree->children[0]->value, "stdin")) {
            fprintln("LD_IN R0", listingFile);
            fprintln("PUSH R0", listingFile);
        } else if (!strcmp(operationTree->children[0]->value, "stdout")) {
            tryPrintOperationTreeNode(
                    operationTree->children[1],
                    listingFile,
                    valuePlaceAssociations,
                    argumentNumber
            );
            fprintln("POP R0", listingFile);
            fprintln("ST_OUT R0", listingFile);
        } else {
            for (int i = 1; i < operationTree->childrenQty; ++i) {
                tryPrintOperationTreeNode(operationTree->children[i],
listingFile, valuePlaceAssociations,
                                          argumentNumber);
            }
            fprintlnWithArg("CALL", operationTree->children[0]->value,
listingFile);
        }
    } else {
        fprintln("EXCEPTION", listingFile);
    }
}
```

Аспекты реализации

Архитектура:`architecture spo {`

```
registers:
   storage R0_STORAGE [16];         // регистр общего назначения 0
   storage R1_STORAGE [16];         // регистр общего назначения 1
   storage IP [16];                 // указатель на инструкцию
   storage SP_STORAGE [16];         // указатель на стек
   storage BP_STORAGE [16];         // указатель на фрейм
   storage IN_PORT [8];             // регистр ввода
   storage OUT_PORT [8];            // регистр вывода
   storage HP [16];                 // указатель на кучу

   view R0 = R0_STORAGE;
   view R1 = R1_STORAGE;
   view IPV = IP;
```

```
   view SP = SP_STORAGE;
   view BP = BP_STORAGE;
   view IN = IN_PORT;
   view OUT = OUT_PORT;

memory:

   range ram [0x0000 .. 0xffff] {
      cell = 8;
      endianess = little-endian;
      granularity = 2;
   }

instructions:

   encode imm16 field = immediate [16];

   encode reg field = register {
      R0 = 0000,
      R1 = 0001,
      IPV = 0100,
      SP = 0101,
      BP = 0110,
      IN = 0111,
      OUT = 1000
   };

   instruction add_register = { 0000 0000, reg as op1, reg as op2, 0000 0000
0000 0000} {
       op1 = op1 + op2;
       IP = IP + 4;
   };

   instruction add_const = { 0000 0001, reg as op1, imm16 as op2, 0000} {
      op1 = op1 + op2;
      IP = IP + 4;
   };

   instruction sub_register = { 0000 0010, reg as op1, reg as op2, 0000 0000
0000 0000} {
      op1 = op1 - op2;
      IP = IP + 4;
   };

   instruction sub_const = { 0000 0011, reg as op1, imm16 as op2, 0000} {
      op1 = op1 - op2;
      IP = IP + 4;
   };

   instruction asl = { 0000 0100, reg as op1, reg as op2, 0000 0000 0000 0000
} {
      op1 = op2 << 1;
      IP = IP + 4;
```

```
   };

   instruction asr = { 0000 0101, reg as op1, reg as op2, 0000 0000 0000 0000
} {
      op1 = op2 >> 1;
      IP = IP + 4;
   };

   instruction mov_register = { 0000 0110, reg as op1, reg as op2, 0000 0000
0000 0000 } {
       op1 = op2;
       IP = IP + 4;
   };

   instruction mov_const = { 0000 0111, reg as op1, imm16 as op2, 0000 } {
       op1 = op2;
       IP = IP + 4;
   };

   instruction invert = { 0000 1000, reg as op1, reg as op2, 0000 0000 0000
0000 } {
       op1 = !op2;
       IP = IP + 4;
   };

   instruction negative = { 0000 1001, reg as op1, reg as op2, 0000 0000 0000
0000 } {
       op1 = -op2;
       IP = IP + 4;
   };

   instruction and_register = { 0000 1010, reg as op1, reg as op2, 0000 0000
0000 0000} {
      op1 = op1 && op2;
      IP = IP + 4;
   };

   instruction and_const = { 0000 1011, reg as op1, imm16 as op2, 0000} {
      op1 = op1 & op2;
      IP = IP + 4;
   };

   instruction or_register = { 0000 1100, reg as op1, reg as op2, 0000 0000
0000 0000} {
       op1 = op1 | op2;
       IP = IP + 4;
   };

   instruction or_const = { 0000 1101, reg as op1, imm16 as op2, 0000} {
       op1 = op1 | op2;
       IP = IP + 4;
   };
```

```
   instruction div_register = { 0000 1110, reg as op1, reg as op2, 0000 0000
0000 0000} {
       op1 = op1 / op2;
       IP = IP + 4;
   };

   instruction div_const = { 0000 1111, reg as op1, imm16 as op2, 0000} {
       op1 = op1 / op2;
       IP = IP + 4;
   };
   instruction mul_register = { 0001 0000, reg as op1, reg as op2, 0000 0000
0000 0000} {
       op1 = op1 * op2;
       IP = IP + 4;
   };

   instruction mul_const = { 0001 0001, reg as op1, imm16 as op2, 0000} {
       op1 = op1 * op2;
       IP = IP + 4;
   };

   instruction rem_register = { 0001 0010, reg as op1, reg as op2, 0000 0000
0000 0000} {
       op1 = op1 % op2;
       IP = IP + 4;
   };

   instruction rem_const = { 0001 0011, reg as op1, imm16 as op2, 0000} {
       op1 = op1 % op2;
       IP = IP + 4;
   };

   instruction jump = { 0001 0100, imm16 as op1, 0000 0000} {
       IP = op1;
   };

   instruction jumpeq = { 0001 0101, reg as op1, imm16 as op2, 0000} {
       if op1 == 0x0 then
       {
           IP = op2;
       }
       else
       {
           IP = IP + 4;
       }
   };
   instruction jumpgt = { 0001 0110, reg as op1, imm16 as op2, 0000} {
      if (op1 >> 15 == 0x0) && (op1 != 0x0) then
      {
          IP = op2;
      }
      else
      {
```

```
        IP = IP + 4;
    }
};
instruction jumpge = { 0001 0111, reg as op1, imm16 as op2, 0000} {
    if (op1 >> 15 == 0x0) then
    {
        IP = op2;
    }
    else
    {
        IP = IP + 4;
    }
};
instruction jumplt = { 0001 1000, reg as op1, imm16 as op2, 0000} {
    if op1 >> 15 == 0x1 then // op1 < 0
        IP = op2;
    else
        IP = IP + 4;
};
instruction jumple = { 0001 1001, reg as op1, imm16 as op2, 0000} {
   if (op1 >> 15 == 0x1) || (op1 == 0x0) then // op1 <= 0
   {
       IP = op2;
   }
   else
   {
       IP = IP + 4;
   }
};

encode bank sequence = alternatives {
   d = {0000},
   c = {0001},
   t = {0011}
};

instruction st = { 0001 1010, reg as op1, imm16 as op2, 0000 } {
   ram:1[op2] = op1;
   ram:1[op2+1] = op1>>8;

   IP = IP + 4;
};

instruction ld = { 0001 1011, reg as op1, imm16 as op2, 0000 } {
   op1 = ram:1[op2] + (ram:1[op2+1] << 8);

   IP = IP + 4;
};
instruction push = { 0001 1100, reg as op1, 0000 0000 0000 0000 0000 } {
    SP = SP - 2;
    ram:1[SP] = op1;
    ram:1[SP+1] = op1 >> 8;
   IP = IP + 4;
```

```
  };
  instruction pop = { 0001 1101, reg as op1, 0000 0000 0000 0000 0000 } {
      op1 = ram:1[SP] + (ram:1[SP+1] << 8);
      SP = SP + 2;
     IP = IP + 4;
  };

  instruction call = { 0001 1110, imm16 as op1, 0000 0000 } {
      SP = SP - 2;
      ram:1[SP] = IP;
      ram:1[SP+1] = IP >> 8;
      SP = SP - 2;
      ram:1[SP] = BP;
      ram:1[SP+1] = BP >> 8;
      BP = SP;
      IP = op1;
  };
  instruction ret = { 0001 1111, imm16 as op1, 0000 0000 } {
      if BP != 0 then {
          SP = BP;
          BP = ram:1[SP] + (ram:1[SP+1] << 8);
          SP = SP + 2;
          IP = ram:1[SP] + (ram:1[SP+1] << 8);
          SP = SP + 2;
          SP = SP + op1 * 2;
      }

      IP = IP + 4;
  };

  instruction ld_bp = { 0010 0000, reg as op1, imm16 as op2, 0000 } {
     op1 = ram:1[(BP + (~op2 + 1) * 8) & 0x0FFFF] + (ram:1[(BP + ((~op2 + 1)
+ 1) * 8) & 0x0FFFF]);

     IP = IP + 4;
  };

  instruction st_bp = { 0010 0001, reg as op1, imm16 as op2, 0000 } {
      ram:1[(BP + (~op2 + 1) * 8) & 0x0FFFF] = op1;
      ram:1[(BP + ((~op2 + 1) + 1) * 8) & 0x0FFFF] = op1 >> 8;

      IP = IP + 4;
  };

  instruction ld_in = { 0010 0010, reg as op1, 0000 0000 0000 0000 0000 } {
      op1 = IN;

      IP = IP + 4;
  };

  instruction st_out = { 0010 0011, reg as op1, 0000 0000 0000 0000 0000 } {
      OUT = op1;
```

```
        IP = IP + 4;
    };

    instruction hlt = { 1111 1111 1111 1111 1111 1111 1111 1111 } {
    };

    instruction jumpne = { 0010 0101, reg as op1, imm16 as op2, 0000} {
        if op1 != 0x0 then
        {
            IP = op2;
        }
        else
        {
            IP = IP + 4;
        }
    };

    instruction eq = { 0010 0110, reg as op1, reg as op2, 0000 0000 0000 0000}
{
        op1 = (op1 == op2);
        IP = IP + 4;
    };


    instruction neq = { 0010 0111, reg as op1, reg as op2, 0000 0000 0000
0000} {
        op1 = (op1 != op2);
        IP = IP + 4;
    };

    instruction load_hp = { 0010 1000, imm16 as op1, 0000 0000} {
        HP = op1;
        IP = IP + 4;
    };

mnemonics:

    mnemonic HLT for hlt();

    format plain1 is "{1}";
    format plain2 is "{1}, {2}";

    mnemonic LOAD_HP for load_hp(op1) plain1;

    mnemonic RET for ret(op1) plain1;
    mnemonic ST for st(op1, op2) plain2;
    mnemonic ST_BP for st_bp(op1, op2) plain2;
    mnemonic ST_OUT for st_out(op1) plain1;
    mnemonic LD for ld(op1, op2) plain2;
    mnemonic LD_BP for ld_bp(op1, op2) plain2;
    mnemonic LD_IN for ld_in(op1) plain1;

    mnemonic CALL for call(op1) plain1;
```

```
  mnemonic JUMP for jump(op1) plain1;
  mnemonic PUSH for push(op1) plain1;
  mnemonic POP for pop(op1) plain1;

  mnemonic NEG for negative (op1, op2) plain2;
  mnemonic NOT for invert (op1, op2) plain2;

  mnemonic ADD for add_register (op1, op2) plain2,
           for add_const (op1, op2) plain2;
  mnemonic SUB for sub_register (op1, op2) plain2,
           for sub_const (op1, op2) plain2;
  mnemonic MOV for mov_register (op1, op2) plain2,
           for mov_const (op1, op2) plain2;
  mnemonic AND for and_register (op1, op2) plain2,
           for and_const (op1, op2) plain2;
  mnemonic OR for or_register (op1, op2) plain2,
           for or_const (op1, op2) plain2;
  mnemonic MUL for mul_register (op1, op2) plain2,
           for mul_const (op1, op2) plain2;
  mnemonic DIV for div_register (op1, op2) plain2,
           for div_const (op1, op2) plain2;
  mnemonic REM for rem_register (op1, op2) plain2,
           for rem_const (op1, op2) plain2;
  mnemonic EQ for eq(op1, op2) plain2;
  mnemonic NEQ for neq(op1, op2) plain2;

  mnemonic JE for jumpeq(op1, op2) plain2;
  mnemonic JNE for jumpne(op1, op2) plain2;
  mnemonic JGT for jumpgt(op1, op2) plain2;
  mnemonic JGE for jumpge(op1, op2) plain2;
  mnemonic JLT for jumplt(op1, op2) plain2;
  mnemonic JLE for jumple(op1, op2) plain2;
}
```

Текстовый файл с описанием графа вызовов введенной программы в формате flowchart. Пример:

```
flowchart TB
node18([Type: operationTreeId, Value: 16]) --> node19([Type: call, Value:
printf])
node344([Type: function, Value: main]) --> node18([Type: operationTreeId,
Value: 16])
node27([Type: operationTreeId, Value: 25]) --> node28([Type: call, Value:
printf])
node344([Type: function, Value: main]) --> node27([Type: operationTreeId,
Value: 25])
node38([Type: operationTreeId, Value: 36]) --> node39([Type: call, Value:
printf])
node344([Type: function, Value: main]) --> node38([Type: operationTreeId,
Value: 36])
node47([Type: operationTreeId, Value: 45]) --> node48([Type: call, Value:
printf])
node344([Type: function, Value: main]) --> node47([Type: operationTreeId,
Value: 45])
node53([Type: operationTreeId, Value: 51]) --> node54([Type: call, Value:
```

```
printf])
node344([Type: function, Value: main]) --> node53([Type: operationTreeId,
Value: 51])
node59([Type: operationTreeId, Value: 57]) --> node60([Type: call, Value:
printf])
...
```

Пример графа вызовов в формате flowchart

Результаты

В результате выполнения работы:

1. Была изучена структура архитектуры различных ВМ.
2. Был реализован модуль формирования линейного кода.
3. Модуль был протестирован посредством реализации программы калькулятора.

Исходный код разработанного решения: https://github.com/47iq/spo

Выводы

В итоге цель работы можно считать успешно выполненной. Модуль был реализован и протестирован. В ходе выполнения работы был изучен материал по рассматриваемым темам.