Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Системное программное обеспечение

Лабораторная работа 3

Вариант 30

Выполнили:
Кривоносов Егор Дмитриевич

Группа: P4114

Преподаватель:
Кореньков Юрий Дмитриевич

2024 г.

Санкт-Петербург

Оглавление

Задание

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.

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:
    b. Согласовать выбор ВМ с преподавателем
    c. Научиться использовать тулчейн (собирать и запускать программы из листинга)
    d. 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

Задание по варианту

Вариант 30

Вариант	Типизация	Код	Банки памяти
30	Статическая	Регистровый двухадресный	6: код, константы, данные, стек

Описание структур

Для работы с AST были добавлены новые структуры данных. Также был создан модуль preprocess_ast для перехода, который принимает массив AST, а затем преобразует в новые структуры.

```
struct preparedBlock {
    preparedStatement *statements;
    int statementsCount;
    ASTNode *astNode;
};
struct preparedVars {
    preparedVar *vars;
    int count;
    ASTNode *astNode;
};
struct preparedWhile {
    preparedExpression condition;
    preparedBlock block;
    ASTNode *astNode;
};
struct preparedDoWhile {
    preparedExpression condition;
    preparedBlock block;
    ASTNode *astNode;
};
struct preparedAssigment {
    preparedExpression to;
    preparedExpression *rightPart;
    ASTNode *astNode;
};
struct preparedStatement {
    statementType type;
    union {
        preparedBlock block;
```

```
        preparedExpression expression;
        preparedIf *ifp;
        preparedVars vars;
        preparedWhile whilep;
        preparedDoWhile dowhile;
        preparedAssigment assigment;
    };
    ASTNode *astNode;
};


struct preparedFunc {
    char *identifier;
    preparedVars args;
    preparedType returnType;
    preparedBlock body;
    int seen;
    ASTNode *astNode;
};
struct conditionalStatement {
    preparedExpression condition;
    preparedStatement statement;
    ASTNode *astNode;
};
struct preparedIf {
    conditionalStatement statement;
    preparedStatement elseStatement;
    int elseStatementExists;
    ASTNode *astNode;
};
```

Описание разработанных модулей

После модуля preprocess_ast, полученные структуры подаются на вход в semantic_analyser. Здесь, путем обхода графа, производятся проверки типов и проверка существования используемых символов. В частности, для проверки существования символов используется модуль symbolic_table.

```
union ctx {
    preparedFunc *func;
};
enum symbolCategory {
    SYMBOL_CATEGORY_FUNC,
    SYMBOL_CATEGORY_VAR
```

```
};
struct symbol {
    char *identifier;
    preparedType type;
    enum symbolCategory category;
    union ctx ctx;
    char *label;
};
struct symbolicTable {
    symbolicTable *parent;
    symbol *symbols;
    int symbolsCount;
    char *currentFuncId;
    int capacity;
};
```

Если на этом этапе не возникло ошибок, вызывается модуль asm_generator, производит генерацию ассемблер кода. Сгенерированный ассемблер код передается в виртуальную машину (ВМ) вместе с архитектурой разработанной по варианту.

Архитектура

```
architecture spo {

registers:
     storage r0st [16];
     storage r1st [16];
     storage r2st [16];
     storage r3st [16];
     storage ip [16];
     storage spst [16];
     storage fpst [16];

     storage inp [8];
     storage outp [8];

    view inReg = inp;
    view outReg = outp;
     view r0 = r0st;
     view r1 = r1st;
     view r2 = r2st;
     view r3 = r3st;
     view ipv = ip;
```

```
      view sp = spst;
      view fp = fpst;


memory:

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

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

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

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

instructions:

      encode imm16 field = immediate [16];

      encode reg field = register {
            r0 = 0000,
            r1 = 0001,
            r2 = 0010,
            r3 = 0011,
            ipv = 0100,
            sp = 0101,
            fp = 0110,
            inReg = 0111,
            outReg = 1000
      };

      instruction add-reg2reg = { 0000 0000, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
            to = op1 + op2;
            ip = ip + 4;
      };
```

```
     instruction add-imm2reg = { 0000 0001, reg as op1, imm16 as
value, reg as to} {
          to = op1 + value;
          ip = ip + 4;
     };
     instruction sub-reg2reg = { 0000 0010, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
          to = op1 - op2;
          ip = ip + 4;
     };

     instruction sub-imm2reg = { 0000 0011, reg as op1, imm16 as
value, reg as to} {
          to = op1 - value;
          ip = ip + 4;
     };

     instruction asl = { 0000 0100, reg as op1, reg as to, 0000
0000 0000 0000 } {
          to = op1 << 1;
          ip = ip + 4;
     };

     instruction asr = { 0000 0101, reg as op1, reg as to, 0000
0000 0000 0000 } {
          to = op1 << 1;
          ip = ip + 4;
     };
    instruction mov-reg2reg = { 0000 0110, reg as op1, reg as to,
0000 0000 0000 0000 } {
        to = op1;
        ip = ip + 4;
    };

    instruction mov-imm2reg = { 0000 0111, imm16 as value, reg as
to, 0000 } {
        to = value;
        ip = ip + 4;
    };
    instruction invert = { 0000 1000, reg as op1, reg as to, 0000
0000 0000 0000 } {
        to = !op1;
        ip = ip + 4;
    };
    instruction negative = { 0000 1001, reg as op1, reg as to, 0000
0000 0000 0000 } {
        to = -op1;
        ip = ip + 4;
    };
     instruction and-reg2reg = { 0000 1010, reg as op1, reg as op2,
```

```
reg as to, 0000 0000 0000} {
          to = op1 && op2;
          ip = ip + 4;
     };

     instruction and-imm2reg = { 0000 1011, reg as op1, imm16 as
value, reg as to} {
          to = op1 & value;
          ip = ip + 4;
     };
    instruction or-reg2reg = { 0000 1100, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
        to = op1 | op2;
        ip = ip + 4;
    };

    instruction or-imm2reg = { 0000 1101, reg as op1, imm16 as
value, reg as to} {
        to = op1 | value;
        ip = ip + 4;
    };
    instruction div-reg2reg = { 0000 1110, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
        to = op1 / op2;
        ip = ip + 4;
    };

    instruction div-imm2reg = { 0000 1111, reg as op1, imm16 as
value, reg as to} {
        to = op1 / value;
        ip = ip + 4;
    };
    instruction mul-reg2reg = { 0001 0000, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
        to = op1 * op2;
        ip = ip + 4;
    };

    instruction mul-imm2reg = { 0001 0001, reg as op1, imm16 as
value, reg as to} {
        to = op1 * value;
        ip = ip + 4;
    };
    instruction rem-reg2reg = { 0001 0010, reg as op1, reg as op2,
reg as to, 0000 0000 0000} {
        to = op1 % op2;
        ip = ip + 4;
    };

    instruction rem-imm2reg = { 0001 0011, reg as op1, imm16 as
value, reg as to} {
```

```
        to = op1 % value;
        ip = ip + 4;
    };

    instruction jump = { 0001 0100, imm16 as to, 0000 0000} {
        ip = to;
    };

    instruction jumpeq = { 0001 0101, reg as op1, imm16 as to,
0000} {
        if op1 == 0x0 then
        {
            ip = to;
        }
        else
        {
            ip = ip + 4;
        }
    };
    instruction jumpgt = { 0001 0110, reg as op1, imm16 as to,
0000} {
       if (op1 >> 15 == 0x0) && (op1 != 0x0) then
       {
           ip = to;
       }
       else
       {
           ip = ip + 4;
       }
    };
    instruction jumpge = { 0001 0111, reg as op1, imm16 as to,
0000} {
        if (op1 >> 15 == 0x0) then
        {
           ip = to;
        }
        else
        {
           ip = ip + 4;
        }
    };
    instruction jumplt = { 0001 1000, reg as op1, imm16 as to,
0000} {
        if op1 >> 15 == 0x1 then // op1 < 0
            ip = to;
        else
            ip = ip + 4;
    };
    instruction jumple = { 0001 1001, reg as op1, imm16 as to,
0000} {
       if (op1 >> 15 == 0x1) || (op1 == 0x0) then // op1 <= 0
```

```
        {
            ip = to;
        }
        else
        {
            ip = ip + 4;
        }
    };

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

     instruction st = { 0001 1010 0000, reg as from, imm16 as ptr }
{
          data_ram:1[ptr] = from;
          data_ram:1[ptr+1] = from>>8;

          ip = ip + 4;
     };

     instruction ld = { 0001 1011 0000, reg as to, imm16 as ptr } {
          to = data_ram:1[ptr] + (data_ram:1[ptr+1] << 8);

          ip = ip + 4;
     };
    instruction push = { 0001 1100, reg as from, 0000 0000 0000
0000 0000 } {
        sp = sp - 2;
        // data_ram:1[sp] = from;
        // data_ram:1[sp+1] = from >> 8;
        stack_ram:1[sp] = from;
        stack_ram:1[sp+1] = from >> 8;
           ip = ip + 4;
    };
    instruction pop = { 0001 1101, reg as to, 0000 0000 0000 0000
0000 } {
        // to = data_ram:1[sp] + (data_ram:1[sp+1] << 8);
        to = stack_ram:1[sp] + (stack_ram:1[sp+1] << 8);
        sp = sp + 2;
           ip = ip + 4;
    };

    instruction call = { 0001 1110, imm16 as ptr, 0000 0000 } {
        sp = sp - 2;
        data_ram:1[sp] = ip;
        data_ram:1[sp+1] = ip >> 8;
        sp = sp - 2;
        data_ram:1[sp] = fp;
```

```
        data_ram:1[sp+1] = fp >> 8;
        fp = sp;
        ip = ptr;
    };
    instruction ret = { 0001 1111 1111 1111 1111 1111 0000 0000 } {
        if fp != 0 then {
            sp = fp;
            fp = data_ram:1[sp] + (data_ram:1[sp+1] << 8);
            sp = sp + 2;
            ip = data_ram:1[sp] + (data_ram:1[sp+1] << 8);
            sp = sp + 2;
        }

        ip = ip + 4;
    };
     instruction hlt = { 1111 1111 1111 1111 1111 1111 1111 1111 }
{
     };

mnemonics:

     mnemonic hlt();
    mnemonic ret();

     format plain1 is "{1}";
     format plain2 is "{1}, {2}";
     format plain3 is "{1}, {2}, {3}";

     mnemonic store for st(from, ptr) plain2;
     mnemonic load for ld(ptr, to) plain2;

    mnemonic call for call(ptr) plain1;

    mnemonic neg for negative (op1, to) plain2;
    mnemonic _not for invert (op1, to) plain2;

     mnemonic add for add-reg2reg (op1, op2, to) plain3,
                      for add-imm2reg (op1, value, to) plain3;

     mnemonic sub for sub-reg2reg (op1, op2, to) plain3,
                      for sub-imm2reg (op1, value, to) plain3;
     mnemonic mov for mov-reg2reg (op1, to) plain2,
                      for mov-imm2reg ( value, to) plain2;
     mnemonic _and for and-reg2reg (op1, op2, to) plain3,
                      for and-imm2reg (op1, value, to) plain3;
     mnemonic _or for or-reg2reg (op1, op2, to) plain3,
                      for or-imm2reg (op1, value, to) plain3;
     mnemonic mul for mul-reg2reg (op1, op2, to) plain3,
                      for mul-imm2reg (op1, value, to) plain3;
     mnemonic div for div-reg2reg (op1, op2, to) plain3,
                      for div-imm2reg (op1, value, to) plain3;
```

```
     mnemonic rem for rem-reg2reg (op1, op2, to) plain3,
                      for rem-imm2reg (op1, value, to) plain3;
    mnemonic push for push(from) plain1;
    mnemonic pop for pop(to) plain1;

    mnemonic jump for jump(to) plain1;
    mnemonic jumpeq for jumpeq(op1, to) plain2;
    mnemonic jumpgt for jumpgt(op1, to) plain2;
    mnemonic jumpge for jumpge(op1, to) plain2;
    mnemonic jumplt for jumplt(op1, to) plain2;
    mnemonic jumple for jumple(op1, to) plain2;
}
```

Пример

Код

```
int read();
void write(int r);

void printZeroError(){
    write(101);
    write(114);
    write(114);
    write(111);
    write(114);
    write(10);
}

void printNumber(int num, bool n){
    int nextLine = 10;
    int revertedNum = 0;

    if (num == 0) {
        write(48);
    } else {

    while (num != 0) {
        revertedNum = (revertedNum * 10) + (num % 10);
        num = num / 10;
    }
    while (revertedNum != 0) {
        write((revertedNum % 10) + 0x30);
        revertedNum = revertedNum / 10;
```

```
    }
    if(n){
        write(10);
    }
    }
}

void printR(int num, bool n, bool neg){
    int nextLine = 10;
    int revertedNum = 0;

    if (num == 0) {
        write(48);
    } else {

    if (neg == true){
        write(45);
    }

    while (num != 0) {
        revertedNum = (revertedNum * 10) + (num % 10);
        num = num / 10;
    }
    while (revertedNum != 0) {
        write((revertedNum % 10) + 0x30);
        revertedNum = revertedNum / 10;
    }
    if(n){
        write(10);
    }
    }
}

void printResult(int a, int b, int operation, int res, bool neg){
    printNumber(a, false);
    if (operation == 1){
        write(43);
    } else if (operation == 2) {
        write(45);
    } else if (operation == 3) {
        write(42);
    } else if (operation == 4) {
        write(47);
    }
    printNumber(b, false);
    write(61);
    printR(res, true, neg);
}

void main() {
    int firstNumber = 0;
```

```
    int secondNumber = 0;
    int operation = 0;
    int state = 0;
    int res = 0;
    bool error = false;
    bool neg = false;
    while(true){
        int i = read();
        if ((i >= 0x30) && (i <= 0x39)){
            int num = i - 0x30;
            if (state == 0){
                firstNumber = firstNumber * 10 + num;
            } else if (state == 2) {
                secondNumber = secondNumber * 10 + num;
            } else {
                state = -1;
                write(i);
                write(10);
                break;
            }
        } else if (state == 0){
            if (i == 0x2b){ // +
                state = 2;
                operation = 1;
            }else if (i == 0x2d){ // -
                state = 2;
                operation = 2;
            }else if (i == 0x2a){ // *
                state = 2;
                operation = 3;
            }else if (i == 0x2f){ // /
                state = 2;
                operation = 4;
            }
        } else if (state == 2) {
            state = 3;
            break;
        } else {
            write(i);
            write(10);
            state = -1;
            break;
        }
    }
    if (state == 3) {
        if (operation == 1){
            res = firstNumber + secondNumber;
        } else if (operation == 2) {
            if (secondNumber > firstNumber){
                neg = true;
                res = secondNumber - firstNumber;
```

```
            } else {
                res = firstNumber - secondNumber;
            }
        } else if (operation == 3) {
            res = firstNumber * secondNumber;
        } else if (operation == 4) {
            if(secondNumber == 0){
                error = true;
                printZeroError();
            } else {
                res = firstNumber / secondNumber;
            }
        }
        if(error == false){
            printResult(firstNumber, secondNumber, operation, res,
neg);
        }
    }
}
```

Сгенерированный ассемблер

```
[section data_ram]
label_write_val: dw 0x0
printNumber_1: dw 0x0
printNumber_2: dw 0x0
label_17: dw 0x0
label_18: dw 0x0
printR_4: dw 0x0
printR_5: dw 0x0
printR_6: dw 0x0
label_32: dw 0x0
label_33: dw 0x0
printResult_8: dw 0x0
printResult_9: dw 0x0
printResult_10: dw 0x0
printResult_11: dw 0x0
printResult_12: dw 0x0
label_65: dw 0x0
label_66: dw 0x0
label_67: dw 0x0
label_68: dw 0x0
label_69: dw 0x0
label_70: dw 0x0
label_71: dw 0x0
label_74: dw 0x0
label_81: dw 0x0
```

```
[section code_ram]
      jump start
read_15:
      mov inReg, r0
      ret

write_16:
      load label_write_val, outReg
      ret

printZeroError_0:
      mov 101, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 114, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 114, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 111, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 114, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 10, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      ret
printNumber_3:
      mov 10, r0
      push r0
```

```
      pop r0
      store r0, label_17
      mov 0, r0
      push r0
      pop r0
      store r0, label_18
            ;if
            ;EQUALITY(num, 0)
      load printNumber_1, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_19
      mov 0, r0
      jump label_20
label_19:
      mov 1, r0
label_20:
      push r0
      pop r0
      jumpeq r0, label_22
            ;then
      mov 48, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      jump label_21
label_22:
            ;else
            ;while
label_23:
            ;NOTEQUAL(num, 0)
      load printNumber_1, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_24
      mov 1, r0
      jump label_25
label_24:
      mov 0, r0
label_25:
      push r0
```

```
      pop r0
      jumpeq r0, label_26
            ;while body
      load label_18, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      mul r0, r1, r0
      push r0
      load printNumber_1, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      rem r0, r1, r0
      push r0
      pop r1
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_18
      load printNumber_1, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      div r0, r1, r0
      push r0
      pop r0
      store r0, printNumber_1
      jump label_23
label_26:
            ;end while
            ;while
label_27:
            ;NOTEQUAL(revertedNum, 0)
      load label_18, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_28
      mov 1, r0
      jump label_29
```

```
label_28:
      mov 0, r0
label_29:
      push r0
      pop r0
      jumpeq r0, label_30
            ;while body
      load label_18, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      rem r0, r1, r0
      push r0
      mov 48, r0
      push r0
      pop r1
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      load label_18, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      div r0, r1, r0
      push r0
      pop r0
      store r0, label_18
      jump label_27
label_30:
            ;end while
            ;if
            ;n
      load printNumber_2, r0
      push r0
      pop r0
      jumpeq r0, label_31
            ;then
      mov 10, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
```

```
          ;endif
label_31:
          ;endif
label_21:
     ret
printR_7:
     mov 10, r0
     push r0
     pop r0
     store r0, label_32
     mov 0, r0
     push r0
     pop r0
     store r0, label_33
          ;if
          ;EQUALITY(num, 0)
     load printR_4, r0
     push r0
     mov 0, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_34
     mov 0, r0
     jump label_35
label_34:
     mov 1, r0
label_35:
     push r0
     pop r0
     jumpeq r0, label_37
          ;then
     mov 48, r0
     push r0
     pop r0
     store r0, label_write_val
     call write_16
     push r0
     jump label_36
label_37:
          ;else
          ;if
          ;EQUALITY(neg, TRUE(NULL, NULL))
     load printR_6, r0
     push r0
     mov 1, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
```

```
      jumpeq r0, label_38
      mov 0, r0
      jump label_39
label_38:
      mov 1, r0
label_39:
      push r0
      pop r0
      jumpeq r0, label_40
            ;then
      mov 45, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
            ;endif
label_40:
            ;while
label_41:
            ;NOTEQUAL(num, 0)
      load printR_4, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_42
      mov 1, r0
      jump label_43
label_42:
      mov 0, r0
label_43:
      push r0
      pop r0
      jumpeq r0, label_44
            ;while body
      load label_33, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      mul r0, r1, r0
      push r0
      load printR_4, r0
      push r0
      mov 10, r0
      push r0
      pop r1
```

```
      pop r0
      rem r0, r1, r0
      push r0
      pop r1
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_33
      load printR_4, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      div r0, r1, r0
      push r0
      pop r0
      store r0, printR_4
      jump label_41
label_44:
            ;end while
            ;while
label_45:
            ;NOTEQUAL(revertedNum, 0)
      load label_33, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_46
      mov 1, r0
      jump label_47
label_46:
      mov 0, r0
label_47:
      push r0
      pop r0
      jumpeq r0, label_48
            ;while body
      load label_33, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      rem r0, r1, r0
      push r0
      mov 48, r0
```

```
      push r0
      pop r1
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      load label_33, r0
      push r0
      mov 10, r0
      push r0
      pop r1
      pop r0
      div r0, r1, r0
      push r0
      pop r0
      store r0, label_33
      jump label_45
label_48:
            ;end while
            ;if
            ;n
      load printR_5, r0
      push r0
      pop r0
      jumpeq r0, label_49
            ;then
      mov 10, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
            ;endif
label_49:
            ;endif
label_36:
      ret
printResult_13:
      load printResult_8, r0
      push r0
      pop r0
      store r0, printNumber_1
      mov 0, r0
      push r0
      pop r0
      store r0, printNumber_2
      call printNumber_3
      push r0
```

```
          ;if
          ;EQUALITY(operation, 1)
     load printResult_10, r0
     push r0
     mov 1, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_50
     mov 0, r0
     jump label_51
label_50:
     mov 1, r0
label_51:
     push r0
     pop r0
     jumpeq r0, label_53
          ;then
     mov 43, r0
     push r0
     pop r0
     store r0, label_write_val
     call write_16
     push r0
     jump label_52
label_53:
          ;else
          ;if
          ;EQUALITY(operation, 2)
     load printResult_10, r0
     push r0
     mov 2, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_54
     mov 0, r0
     jump label_55
label_54:
     mov 1, r0
label_55:
     push r0
     pop r0
     jumpeq r0, label_57
          ;then
     mov 45, r0
     push r0
     pop r0
     store r0, label_write_val
```

```
      call write_16
      push r0
      jump label_56
label_57:
            ;else
            ;if
            ;EQUALITY(operation, 3)
      load printResult_10, r0
      push r0
      mov 3, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_58
      mov 0, r0
      jump label_59
label_58:
      mov 1, r0
label_59:
      push r0
      pop r0
      jumpeq r0, label_61
            ;then
      mov 42, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      jump label_60
label_61:
            ;else
            ;if
            ;EQUALITY(operation, 4)
      load printResult_10, r0
      push r0
      mov 4, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_62
      mov 0, r0
      jump label_63
label_62:
      mov 1, r0
label_63:
      push r0
      pop r0
      jumpeq r0, label_64
```

```
            ;then
      mov 47, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
            ;endif
label_64:
            ;endif
label_60:
            ;endif
label_56:
            ;endif
label_52:
      load printResult_9, r0
      push r0
      pop r0
      store r0, printNumber_1
      mov 0, r0
      push r0
      pop r0
      store r0, printNumber_2
      call printNumber_3
      push r0
      mov 61, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      load printResult_11, r0
      push r0
      pop r0
      store r0, printR_4
      mov 1, r0
      push r0
      pop r0
      store r0, printR_5
      load printResult_12, r0
      push r0
      pop r0
      store r0, printR_6
      call printR_7
      push r0
      ret
start:
main_14:
      mov 0, r0
      push r0
      pop r0
```

```
      store r0, label_65
      mov 0, r0
      push r0
      pop r0
      store r0, label_66
      mov 0, r0
      push r0
      pop r0
      store r0, label_67
      mov 0, r0
      push r0
      pop r0
      store r0, label_68
      mov 0, r0
      push r0
      pop r0
      store r0, label_69
      mov 0, r0
      push r0
      pop r0
      store r0, label_70
      mov 0, r0
      push r0
      pop r0
      store r0, label_71
            ;while
label_72:
            ;TRUE(NULL, NULL)
      mov 1, r0
      push r0
      pop r0
      jumpeq r0, label_73
            ;while body
      call read_15
      push r0
      pop r0
      store r0, label_74
            ;if
            ;AND(braces(GREATERTHANEQ(i, 48), NULL),
braces(LESSTHANEQ(i, 57), NULL))
      load label_74, r0
      push r0
      mov 48, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpge r0, label_75
      mov 0, r0
      jump label_76
label_75:
```

```
      mov -1, r0
label_76:
      push r0
      load label_74, r0
      push r0
      mov 57, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumple r0, label_77
      mov 0, r0
      jump label_78
label_77:
      mov -1, r0
label_78:
      push r0
      pop r1
      pop r0
      _and r0, r1, r0
      push r0
      pop r0
      jumpeq r0, label_80
            ;then
      load label_74, r0
      push r0
      mov 48, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      push r0
      pop r0
      store r0, label_81
            ;if
            ;EQUALITY(state, 0)
      load label_68, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_82
      mov 0, r0
      jump label_83
label_82:
      mov 1, r0
label_83:
      push r0
      pop r0
```

```
     jumpeq r0, label_85
          ;then
     load label_65, r0
     push r0
     mov 10, r0
     push r0
     pop r1
     pop r0
     mul r0, r1, r0
     push r0
     load label_81, r0
     push r0
     pop r1
     pop r0
     add r0, r1, r0
     push r0
     pop r0
     store r0, label_65
     jump label_84
label_85:
          ;else
          ;if
          ;EQUALITY(state, 2)
     load label_68, r0
     push r0
     mov 2, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_86
     mov 0, r0
     jump label_87
label_86:
     mov 1, r0
label_87:
     push r0
     pop r0
     jumpeq r0, label_89
          ;then
     load label_66, r0
     push r0
     mov 10, r0
     push r0
     pop r1
     pop r0
     mul r0, r1, r0
     push r0
     load label_81, r0
     push r0
     pop r1
```

```
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_66
      jump label_88
label_89:
            ;else
      mov 1, r0
      push r0
      pop r0
      neg r0, r0
      push r0
      pop r0
      store r0, label_68
      load label_74, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 10, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
            ;break
      jump label_73
            ;endif
label_88:
            ;endif
label_84:
      jump label_79
label_80:
            ;else
            ;if
            ;EQUALITY(state, 0)
      load label_68, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_90
      mov 0, r0
      jump label_91
label_90:
      mov 1, r0
label_91:
```

```
      push r0
      pop r0
      jumpeq r0, label_93
            ;then
            ;if
            ;EQUALITY(i, 43)
      load label_74, r0
      push r0
      mov 43, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_94
      mov 0, r0
      jump label_95
label_94:
      mov 1, r0
label_95:
      push r0
      pop r0
      jumpeq r0, label_97
            ;then
      mov 2, r0
      push r0
      pop r0
      store r0, label_68
      mov 1, r0
      push r0
      pop r0
      store r0, label_67
      jump label_96
label_97:
            ;else
            ;if
            ;EQUALITY(i, 45)
      load label_74, r0
      push r0
      mov 45, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_98
      mov 0, r0
      jump label_99
label_98:
      mov 1, r0
label_99:
      push r0
      pop r0
```

```
     jumpeq r0, label_101
          ;then
     mov 2, r0
     push r0
     pop r0
     store r0, label_68
     mov 2, r0
     push r0
     pop r0
     store r0, label_67
     jump label_100
label_101:
          ;else
          ;if
          ;EQUALITY(i, 42)
     load label_74, r0
     push r0
     mov 42, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_102
     mov 0, r0
     jump label_103
label_102:
     mov 1, r0
label_103:
     push r0
     pop r0
     jumpeq r0, label_105
          ;then
     mov 2, r0
     push r0
     pop r0
     store r0, label_68
     mov 3, r0
     push r0
     pop r0
     store r0, label_67
     jump label_104
label_105:
          ;else
          ;if
          ;EQUALITY(i, 47)
     load label_74, r0
     push r0
     mov 47, r0
     push r0
     pop r1
     pop r0
```

```
      sub r0, r1, r0
      jumpeq r0, label_106
      mov 0, r0
      jump label_107
label_106:
      mov 1, r0
label_107:
      push r0
      pop r0
      jumpeq r0, label_108
            ;then
      mov 2, r0
      push r0
      pop r0
      store r0, label_68
      mov 4, r0
      push r0
      pop r0
      store r0, label_67
            ;endif
label_108:
            ;endif
label_104:
            ;endif
label_100:
            ;endif
label_96:
      jump label_92
label_93:
            ;else
            ;if
            ;EQUALITY(state, 2)
      load label_68, r0
      push r0
      mov 2, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_109
      mov 0, r0
      jump label_110
label_109:
      mov 1, r0
label_110:
      push r0
      pop r0
      jumpeq r0, label_112
            ;then
      mov 3, r0
      push r0
```

```
      pop r0
      store r0, label_68
            ;break
      jump label_73
      jump label_111
label_112:
            ;else
      load label_74, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 10, r0
      push r0
      pop r0
      store r0, label_write_val
      call write_16
      push r0
      mov 1, r0
      push r0
      pop r0
      neg r0, r0
      push r0
      pop r0
      store r0, label_68
            ;break
      jump label_73
            ;endif
label_111:
            ;endif
label_92:
            ;endif
label_79:
      jump label_72
label_73:
            ;end while
            ;if
            ;EQUALITY(state, 3)
      load label_68, r0
      push r0
      mov 3, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_113
      mov 0, r0
      jump label_114
label_113:
      mov 1, r0
```

```
label_114:
      push r0
      pop r0
      jumpeq r0, label_115
            ;then
            ;if
            ;EQUALITY(operation, 1)
      load label_67, r0
      push r0
      mov 1, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_116
      mov 0, r0
      jump label_117
label_116:
      mov 1, r0
label_117:
      push r0
      pop r0
      jumpeq r0, label_119
            ;then
      load label_65, r0
      push r0
      load label_66, r0
      push r0
      pop r1
      pop r0
      add r0, r1, r0
      push r0
      pop r0
      store r0, label_69
      jump label_118
label_119:
            ;else
            ;if
            ;EQUALITY(operation, 2)
      load label_67, r0
      push r0
      mov 2, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_120
      mov 0, r0
      jump label_121
label_120:
      mov 1, r0
```

```
label_121:
      push r0
      pop r0
      jumpeq r0, label_123
            ;then
            ;if
            ;GREATERTHAN(secondNumber, firstNumber)
      load label_66, r0
      push r0
      load label_65, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpgt r0, label_124
      mov 0, r0
      jump label_125
label_124:
      mov -1, r0
label_125:
      push r0
      pop r0
      jumpeq r0, label_127
            ;then
      mov 1, r0
      push r0
      pop r0
      store r0, label_71
      load label_66, r0
      push r0
      load label_65, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      push r0
      pop r0
      store r0, label_69
      jump label_126
label_127:
            ;else
      load label_65, r0
      push r0
      load label_66, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      push r0
      pop r0
      store r0, label_69
```

```
           ;endif
label_126:
     jump label_122
label_123:
           ;else
           ;if
           ;EQUALITY(operation, 3)
     load label_67, r0
     push r0
     mov 3, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_128
     mov 0, r0
     jump label_129
label_128:
     mov 1, r0
label_129:
     push r0
     pop r0
     jumpeq r0, label_131
           ;then
     load label_65, r0
     push r0
     load label_66, r0
     push r0
     pop r1
     pop r0
     mul r0, r1, r0
     push r0
     pop r0
     store r0, label_69
     jump label_130
label_131:
           ;else
           ;if
           ;EQUALITY(operation, 4)
     load label_67, r0
     push r0
     mov 4, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_132
     mov 0, r0
     jump label_133
label_132:
     mov 1, r0
```

```
label_133:
      push r0
      pop r0
      jumpeq r0, label_134
            ;then
            ;if
            ;EQUALITY(secondNumber, 0)
      load label_66, r0
      push r0
      mov 0, r0
      push r0
      pop r1
      pop r0
      sub r0, r1, r0
      jumpeq r0, label_135
      mov 0, r0
      jump label_136
label_135:
      mov 1, r0
label_136:
      push r0
      pop r0
      jumpeq r0, label_138
            ;then
      mov 1, r0
      push r0
      pop r0
      store r0, label_70
      call printZeroError_0
      push r0
      jump label_137
label_138:
            ;else
      load label_65, r0
      push r0
      load label_66, r0
      push r0
      pop r1
      pop r0
      div r0, r1, r0
      push r0
      pop r0
      store r0, label_69
            ;endif
label_137:
            ;endif
label_134:
            ;endif
label_130:
            ;endif
label_122:
```

```
           ;endif
label_118:
           ;if
           ;EQUALITY(error, FALSE(NULL, NULL))
     load label_70, r0
     push r0
     mov 0, r0
     push r0
     pop r1
     pop r0
     sub r0, r1, r0
     jumpeq r0, label_139
     mov 0, r0
     jump label_140
label_139:
     mov 1, r0
label_140:
     push r0
     pop r0
     jumpeq r0, label_141
          ;then
     load label_65, r0
     push r0
     pop r0
     store r0, printResult_8
     load label_66, r0
     push r0
     pop r0
     store r0, printResult_9
Резул load label_67, r0
     push r0
     pop r0
     store r0, printResult_10
     load label_69, r0
     push r0
     pop r0
     store r0, printResult_11
     load label_71, r0
     push r0
     pop r0
     store r0, printResult_12
     call printResult_13
     push r0
          ;endif
label_141:
           ;endif
label_115:
     ret
     jump halt
halt:
     hlt
```

```
```

Входной файл в программу

```
21-1092
```

Результат запуска

```
Task ExecuteBinaryWithInput started with id 953fbd6c-0294-4bbc-8d11-80c770594f15
Waiting for completion...
Here is task output interpreted with UTF8:
21-1092=-1071
```

Данный калькулятор поддерживает базовые операции “+”, “-”, “*”, “/” и вывод об ошибках.

Вывод

В ходе выполнения данной лабораторной работы было написано описание архитектуры для эмулятора, а также реализован транслятор в ассемблер для этой архитектуры. А также написан пример калькулятора и вывода текстовых сообщений (артов) в консоль.