САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA. Ассемблер, дизассемблер»

Выполнил(а): Абраимов Илья Дмитриевич

Номер ИСУ: 336351

студ. гр. M3134

Санкт-Петербург

2021

Цель работы: знакомство с архитектурой набора команд RISC-V.

Инструментарий и требования к работе: работа может быть выполнена на любом из следующих языков: C/C++, Python, Java.

Теоретическая часть

RISC-V – это система набора команд и процессорная архитектура. Сами команды разделяются на принадлежащие базовому набору (нас интересует R32I) и расширенные (нас интересуют расширения M и C). Инструкции наборов R32I и R32M имеют длину 32 бита, где младшие 7 бит отвечают за opcode, определяющий тип команды. Заметим при этом, что каждый из двух младших бит opcode всегда равен единице в случае 32-битной инструкции. Рассмотрим различные типы инструкций. Далее будем нумеровать биты, начиная от младших к старшим, т. е. слева направо. U-type: используется для записи 20 бит в регистр; биты 7-11 отвечают за регистр rd, биты 12-31 – за старшие 20 бит immediate. I-type: используется для операций с временным значением; биты 7-1 отвечают за rd, биты 15-19, в зависимости от opcode, отвечают за rs1 или за zimm, биты 20-31, в зависимости от opcode, отвечают за immediate или за csr или биты 20-24 могут отвечать за shamt. R-type: используется в операциях задействующих исключительно три регистра; устроен почти так же как I, только без imm и инструкция всегда содержит три регистра. B-type: используется для операций условного пререхода; устроен почти так же как I. J-type: используется для операция безусловного перехода (прыжка); устроен подобно U-type, только imm считается по-другому. S-type: используется для записи значний в память; устроен подобно B-type, только imm считается по-другому. RISC-V использует 32 регистра, которые называются в формате x%s, где %s – число от 0 до 31. В таблице 1 представлено соглашение о названии регистров в unix системах.

Таблица №1 – соглашение об использовании регистров

Register	**ABI name**	**Описание**
x0	zero	Постоянно ноль
x1	ra	Возвращаемое значение
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-7	t0-t2	Временные регистры
x8-9	s0-1	Сохраненные регистры
x10-17	a0-7	Аргументы функций
x18-27	s2-11	Сохраненные регистры
x28-31	t3-6	Временные регистры

RISC-V также имеет модификации. Нас интересуют M (предназначена для операций, связанных с умножением и делением) и C (предназначена для хранение 16-битных команд, использование которых, в свою очередь помогает сократить размер файла).

ELF (Executable and Linkable Format) - формат двоичных файлов, используемый во многих современных UNIX-подобных операционных системах. Сам файл можно условно разделить на несколько блоков. ELF Header – представляет собой заголовок ELF файла, в котором хранится

основная информация о нем (метод кодирования данных; архитектура платформы, для которой создан файл…). Program Header - заголовки, каждый из которых описывает отдельный сегмент программы и его атрибуты либо другую информацию, необходимую операционной системе для подготовки программы к исполнению. Section Header – заголовки секций файла; тут содержатся основные сведения о секциях файла, в том числе информация, необходимая для нахождения этих секций. В самих секциях содержится всевозможные данные для работы программы, например, в секции .text может содержаться двоичное представление инструкций ассемблера, в .symtab – таблица строк, необходимых для вывода меток при дизасемблировании.

Описание работы написанного кода

1. Сначала проверим ELF файл на корректность, затем получим необходимые секции (будем хранить в памяти .symtab и смещение на начало .text)
2. Затем считываем 2 байта из .text. Если opcode не заканчивается на “11”, то это RVC модификация. Если opcode заканчивается на “11”, то это не RVC модификация; считываем еще 2 байта.
3. Перед непосредственным выводом комманд нужно пройтись по .text и при встрече команды условного или безусловного перехода, добавить номер строки, на которую мы переходим с список unknownMarks
4. Снова идем по .text, считывая 2 или 4 байта (см. п. 2). Проверяем, если адрес текущей строки есть в .symtab, то присваиваем метке для текущей строки значение из таблицы символов. Если нет, то проверяем на вхождение адреса в unknownMarks и, при наличии, присваиваем метке для текущей строки значение в формате

LOC_%05x, иначе – делаем метку для текущей строки равной пустой строке.

5. Дизасемблируем инструкции, для каждой команды выводим сначала адрес, затем метку (при наличии)
6. Если встретили команду условного или безусловного перехода, то выводим адрес строки, на которую мы прыгаем в формате #0x%05x и метку этой строки
7. Выводим .symtab. При этом все специальные значения index между HIOS и LOOS я буду называть OSSPEC.
8. Закроем незакртые файлы

Пример результата работы программы

```
000133c0            addi a0, s0, 0
000133c4            lw s0, 136(sp)
000133c8            lw s1, 132(sp)
000133cc            addi sp, sp, 144
000133d0            jalr zero, 0(ra)
000133d4            sub s0, zero, s0
000133d8            jal ra, 640 #0x13658 __errno
000133dc            sw s0, 0(a0)
000133e0            addi s0, zero, -1
000133e4            jal zero, -52 #0x133b0 LOC_133b0
000133e8   _isatty: addi sp, sp, -112
000133ec            addi a1, sp, 8
000133f0            sw ra, 108(sp)
```

Листинг

Используемый язык: Java 16

MainDisassembler.java

```
import java.io.*;
import java.util.InputMismatchException;

public class MainDisassembler {
    public static void main(String[] args) { //args[0] - input, args[1] - output
        String input;
        if (args.length >= 1) {
            input = args[0];
        } else {
            System.err.println("Can't find input file name");
            return;
        }

        try {
            OutputStreamWriter output =
                    new OutputStreamWriter(args.length > 1 ? new
FileOutputStream(args[1]) : System.out);
            try {

                RISCVDisassembler dis = new RISCVDisassembler(new
PrintWriter(output));
                dis.doDisassemble(input);
            } finally {
                output.close();
            }
        } catch (FileNotFoundException e){
            System.err.println("File not found");
        } catch (IOException e) {
            System.err.println("Something went wrong");
        } catch (InputMismatchException e) {
            System.err.println("Somethind went wrong: " + e);
        }

    }
}
```

RISCVDisassembler.java

```
import elf.ELF32File;

import java.io.*;
import java.util.HashSet;
import java.util.Set;

public class RISCVDisassembler {
```

```
    public ELF32File elf;
    public long addr;
    public PrintWriter output;
    int decimalWord;
    int rd;
    int funct3;
    int rs1;
    int rs2;
    int imm110;
    int funct7;
    int opcode;
    Set<Long> unknownMarks = new HashSet<>();
    String operation;
    int imm;


    public RISCVDisassembler(PrintWriter output) {
        this.output = output;
    }

    public void disassemble(String word) {
        String mark;
        mark = elf.getSym(addr);
        if (mark == null && unknownMarks.contains(addr)) {
            mark = String.format("LOC_%05x:", addr);
        } else if (mark == null){
            mark = "";
        } else mark += ":";
        decimalWord = (int) Long.parseLong(word, 2);

        if (word.length() == 32) { // Это не RVC
            getUsefulThingsFromWord();

            if (opcode == 0b0110011) { //R-type
                disR(word);
                output.printf("%08x %10s %s %s, %s, %s\n",
                        addr, mark, operation, getRegister(rd), getRegister(rs1),
getRegister(rs2));
            } else if (opcode == 0b0100011) { //S-type
                disS();
                output.printf("%08x %10s %s %s, %s(%s)\n", addr, mark, operation,
getRegister(rs2), imm, getRegister(rs1));
            } else if (opcode == 0b0110111) { //U-type, lui
                output.printf("%08x %10s %s %s, %s\n", addr, mark, "lui",
getRegister(rd),
                        Integer.toUnsignedString((decimalWord >>> 12) << 12));
//            output.printf("%6s %s, %s", "lui", getRegister(rd),
//                    Integer.toUnsignedString((decimalWord >>> 12) << 12));
            } else if (opcode == 0b0010111) { //U-type, auipc
                output.printf("%08x %10s %s %s, %s\n", addr, mark, "auipc",
getRegister(rd),
```

```
                        Integer.toUnsignedString((decimalWord >>> 12) << 12));
//            output.printf("%6s %s, %s", "auipc", getRegister(rd),
//                    Integer.toUnsignedString((decimalWord >>> 12) << 12));
            } else if (opcode == 0b1101111) { //J-type, jal
                disJ(word);
                String m = elf.getSym(addr + imm) != null ?
                        elf.getSym(addr + imm) : String.format("LOC_%05x", addr +
imm);
                output.printf("%08x %10s %s %s, %s #0x%05x %s\n", addr, mark,
"jal", getRegister(rd), imm, addr + imm, m);
            } else if (opcode == 0b1100111) { //I-type, jalr
                int imm_i = decimalWord >> 20;
                output.printf("%08x %10s %s %s, %s(%s)\n", addr, mark, "jalr",
getRegister(rd), imm_i, getRegister(rs1));
            } else if (opcode == 0b0000011) { //I-type, load
                disIload();
                int imm_i = decimalWord >> 20;
                output.printf("%08x %10s %s %s, %s(%s)\n", addr, mark, operation,
getRegister(rd), imm_i, getRegister(rs1));
            } else if (opcode == 0b0010011) { //I-type, arithmetic or shifts
                disIarithm(word);
                int imm_i;
                if (operation.equals("slli") || operation.equals("srli") ||
operation.equals("srai")) {
                    imm_i = (decimalWord << 7) >>> 27; //Тут imm_i означает shamt
                } else {
                    imm_i = decimalWord >> 20;
                }
                output.printf("%08x %10s %s %s, %s, %s\n", addr, mark, operation,
getRegister(rd), getRegister(rs1), imm_i);
            } else if (opcode == 0b1100011) { //B-type
                disB();
                StringBuilder sb = new StringBuilder();

sb.append(String.valueOf(word.charAt(0)).repeat(20)).append(word.charAt(24))
                        .append(word, 1, 7).append(word, 20, 25).append("0");
                int imm_b = (int) Long.parseLong(String.valueOf(sb), 2);
                String m = elf.getSym(addr + imm_b) != null ?
                        elf.getSym(addr + imm_b) : String.format("LOC_%05x", addr
+ imm_b);
                output.printf("%08x %10s %s %s, %s, %s #0x%05x %s\n",
                        addr, mark, operation, getRegister(rs1), getRegister(rs2),
imm_b, addr + imm_b, m);
            } else if (opcode == 0b1110011) { //System cmds
                if (funct3 == 0b000 && word.charAt(11) == '0') { //ecall
                    output.printf("%08x %10s %s\n", addr, mark, "ecall");
                } else if (funct3 == 0b000 && word.charAt(11) == '1') { //ebreak
                    output.printf("%08x %10s %s\n", addr, mark, "ebreak");
                } else { //csr
                    disCSR();
                    output.printf("%08x %10s %s %s, %s, %s\n",
```

```
                            addr, mark, operation, getRegister(rd), imm110,
getRegister(rs1));
                }
            } else {
                output.printf("%08x %10s\n", addr, "unknown_command");
            }
        } else {

            opcode = decimalWord & ((1 << 2) - 1);
            funct3 = decimalWord >>> 13;
            short imm;
            int uimm;

            if (opcode == 0b00) {
//                int rd = (decimalWord << 27) >>> 29;
//                rs1 = (decimalWord << 19) >>> 26;
                int rd = Integer.parseInt(word.substring(11, 14),2);
                rs1 = Integer.parseInt(word.substring(6, 9), 2);

                int intUimm = Integer.parseInt(
                        word.charAt(10) + word.substring(3, 6) + word.charAt(9) +
"00", 2);
                switch (funct3){
                    case (0b000): //c.addi4spn
                        int nzuimm = Integer.parseInt(word.substring(5, 9) +
word.substring(3,5) +
                                word.charAt(10) + word.charAt(9) + "00", 2);
                        output.printf("%08x %10s %s %s, %s, %s\n",
                                addr, mark, "c.addi4spn", getABIRegister(rd),
"sp", nzuimm);
                        break;
                    case (0b010): //c.lw
                        uimm = intUimm;
                        output.printf("%08x %10s %s %s, %s(%s)\n",
                                addr, mark, "c.lw", getABIRegister(rd), uimm,
getABIRegister(rs1));
                        break;
                    case (0b110): //c.sw
                        uimm = intUimm;
                        output.printf("%08x %10s %s %s, %s(%s)\n",
                                addr, mark, "c.sw", getABIRegister(rd), uimm,
getABIRegister(rs1));
                        break;
                    default:
                        output.printf("%08x %10s\n", addr, "unknown_command");
                        break;
                }
            } else if (opcode == 0b01) {
                int intImm =
Integer.parseInt(String.valueOf(word.charAt(3)).repeat(10) +
                        word.charAt(3) + word.substring(9, 14), 2);
                switch (funct3) {
```

```
                    case (0b000):
                        if (decimalWord == 1) { //c.nop
                            output.printf("%08x %10s %s\n",
                                    addr, mark, "c.nop");
                            break;
                        } else { //c.addi
                            short nzuimm = (short) intImm;
                            output.printf("%08x %10s %s %s, %s\n",
                                    addr, mark, "c.addi",
getRegister(word.substring(4, 9)), nzuimm);
                            break;
                        }
                    case (0b001): //c.jal
                        imm = getImmForRVCJumps(word);
                        String m = elf.getSym(addr + imm) != null ?
                                elf.getSym(addr + imm) : String.format("LOC_%05x",
addr + imm);
                        output.printf("%08x %10s %s %s #0x%05x %s\n",
                                addr, mark, "c.jal", imm, addr + imm, m);
                        break;
                    case (0b010): //c.li
                        imm = (short) intImm;
                        output.printf("%08x %10s %s %s, %s\n",
                                addr, mark, "c.li", getRegister(word.substring(4,
9)), imm);
                        break;
                    case (0b011): //c.addi16sp
                        if (word.startsWith("00010", 4)) { //c.addi16sp
                            imm = (short)
Integer.parseInt(String.valueOf(word.charAt(3)).repeat(6) +
                                    word.charAt(3) + word.substring(11, 13) +
word.charAt(10) + word.charAt(13) +
                                    word.charAt(9) + "0000", 2);
                            output.printf("%08x %10s %s %s, %s\n",
                                    addr, mark, "c.addi16sp", "sp", imm);
                            break;
                        } else { //c.lui
                            int luiImm =
Integer.parseInt(String.valueOf(word.charAt(3)).repeat(14) +
                                    word.charAt(3) + word.substring(9, 14), 2);
                            output.printf("%08x %10s %s %s, %s\n",
                                    addr, mark, "c.lui",
getRegister(word.substring(4, 9)), luiImm);
                            break;
                        }
                    case (0b100):
                        String operation = disRVCArithm(word);
                        output.printf("%08x %10s %s %s, %s\n", addr, mark,
operation,
                                getABIRegister(word.substring(6, 9)),
getABIRegister(word.substring(11, 14)));
```

```
                        break;
                    case (0b101): //c.j
                        imm = getImmForRVCJumps(word);
                        m = elf.getSym(addr + imm) != null ?
                                elf.getSym(addr + imm) : String.format("LOC_%05x",
addr + imm);
                        output.printf("%08x %10s %s %s #0x%05x %s\n",
                                addr, mark, "c.j", imm, addr + imm, m);
                        break;
                    case (0b110): //c.beqz, c.bnez
                    case (0b111):
                        imm = (short)
Integer.parseInt(String.valueOf(word.charAt(3)).repeat(7) +
                                word.charAt(3) + word.substring(9, 11) +
word.charAt(13) + word.substring(4, 6) +
                                word.substring(11, 13) + "0", 2);
                        m = elf.getSym(addr + imm) != null ?
                                elf.getSym(addr + imm) : String.format("LOC_%05x",
addr + imm);
                        output.printf("%08x %10s %s %s %s #0x%05x %s\n",
                                addr, mark, word.startsWith("110") ? "c.beqz" :
"c.bnez",
                                getABIRegister(word.substring(6, 9)), imm, addr +
imm, m);
                        break;
                    default:
                        output.printf("%08x %10s\n", addr, "unknown_command");
                        break;
                }
            } else if (opcode == 0b10) {
                switch (funct3) {
                    case (0b000): //c.slli
                        uimm = Integer.parseInt(word.charAt(3) + word.substring(9,
14));
                        output.printf("%08x %10s %s %s, %s\n",
                                addr, mark, "c.slli",
getRegister(word.substring(4, 9)), uimm);
                        break;
                    case (0b010): //c.lwsp
                        uimm = Integer.parseInt(word.substring(12, 14) +
word.charAt(3) +
                                word.substring(9, 12), 2);
                        output.printf("%08x %10s %s %s, %s(%s)\n", addr, mark,
"c.lwsp",
                                getRegister(word.substring(4, 9)), uimm, "sp");
                        break;
                    case (0b100):
                        if (word.charAt(3) == '0' && word.substring(9,
14).equals("00000")) { //c.jr
                            output.printf("%08x %10s %s %s\n", addr, mark, "c.jr",
getRegister(word.substring(4, 9)));
```

```
                            break;
                        } else if (word.charAt(3) == '0') { //c.mv
                            output.printf("%08x %10s %s %s, %s\n", addr, mark,
"c.mv",
                                    getRegister(word.substring(4, 9)),
getRegister(word.substring(9, 14)));
                            break;
                        } else if (word.charAt(3) == '1' && word.substring(4,
9).equals("00000") &&
                                    word.substring(9, 14).equals("00000")) {
//c.ebreak
                            output.printf("%08x %10s %s\n", addr, mark,
"c.ebreak");
                            break;
                        } else if (word.charAt(3) == '1' && word.substring(9,
14).equals("00000")) { //c.jalr
                            output.printf("%08x %10s %s %s\n", addr, mark,
"c.jalr", getRegister(word.substring(4, 9)));
                            break;
                        } else { //c.add
                            output.printf("%08x %10s %s %s, %s\n", addr, mark,
"c.add",
                                    getRegister(word.substring(4, 9)),
getRegister(word.substring(9, 14)));
                            break;
                        }
                    case (0b110): //c.swsp
                        uimm = Integer.parseInt(word.substring(7, 9) +
word.substring(3, 7) + "00", 2);
                        output.printf("%08x %10s %s %s, %s(%s)\n", addr, mark,
"c.swsp",
                                getRegister(word.substring(9, 14)), uimm, "sp");
                        break;
                    default:
                        output.printf("%08x %10s\n", addr, "unknown_command");
                }
            }
        }
    }

    private void getUsefulThingsFromWord() {
        opcode = decimalWord & ((1 << 7) - 1);
        rd = decimalWord >> 7 & ((1 << 5) - 1);
        funct3 = decimalWord >> 12 & ((1 << 3) - 1);
        rs1 = decimalWord >> 15 & ((1 << 5) - 1);
        rs2 = decimalWord >> 20 & ((1 << 5) - 1);
        imm110 = decimalWord >> 20 & ((1 << 12) - 1);
        funct7 = decimalWord >> 25;
    }

    private short getImmForRVCJumps(String word) {
```

```
        short imm;
        imm = (short) Integer.parseInt(
                String.valueOf(word.charAt(3)).repeat(4) + word.charAt(3) +
word.charAt(7) +
                        word.substring(5, 7) + word.charAt(9) + word.charAt(13) +
                        word.charAt(4) + word.substring(10, 13) + "0", 2);
        return imm;
    }

    private String disRVCArithm(String word) {
        int code11_10 = Integer.parseInt(word.substring(4, 6), 2);
        int code6_5 = Integer.parseInt(word.substring(9, 11), 2);
        if (code11_10 == 0b00) return "c.srli";
        else if (code11_10 == 0b01) return "c.srai";
        else if (code11_10 == 0b10) return "c.endi";
        else if (code11_10 == 0b11) {
            switch (code6_5) {
                case (0b00):
                    return "c.sub";
                case (0b01):
                    return "c.xor";
                case (0b10):
                    return "c.or";
                case (0b11):
                    return "c.and";
                default:
                    return "unknown_command";
            }
        } else return "unknown_command";
    }

    private String getABIRegister(int rd) {
        String[] regs = new String[]{"s0", "s1", "a0", "a1", "a2", "a3", "a4",
"a5"};
        return regs[rd];
    }

    private String getABIRegister(String rd) {
        return getABIRegister(Integer.parseInt(rd, 2));
    }

    private void disCSR() {
        switch (funct3) {
            case (0b001):
                operation = "csrrw";
                break;
            case (0b010):
                operation = "csrrs";
                break;
            case (0b011):
                operation = "csrrc";
```

```
            break;
        case (0b101):
            operation = "csrrwi";
            break;
        case (0b110):
            operation = "csrrsi";
            break;
        case (0b111):
            operation = "csrrci";
            break;
    }
}

private void disB() {
    switch (funct3) {
        case (0b000):
            operation = "beq";
            break;
        case (0b001):
            operation = "bne";
            break;
        case (0b100):
            operation = "blt";
            break;
        case (0b101):
            operation = "bge";
            break;
        case (0b110):
            operation = "bltu";
            break;
        case (0b111):
            operation = "bgeu";
            break;
    }
}

private void disIarithm(String word) {
    switch (funct3) {
        case (0b000):
            operation = "addi";
            break;
        case (0b010):
            operation = "slti";
            break;
        case (0b011):
            operation = "sltiu";
            break;
        case (0b100):
            operation = "xori";
            break;
        case (0b110):
```

```
            operation = "ori";
            break;
        case (0b111):
            operation = "andi";
            break;
        case (0b001):
            operation = "slli";
            break;
        case (0b101):
            if (word.charAt(1) == '0') operation = "srli";
            else operation = "srai";
            break;
    }
}

private void disIload() {
    switch (funct3) {
        case (0b000):
            operation = "lb";
            break;
        case (0b001):
            operation = "lh";
            break;
        case (0b010):
            operation = "lw";
            break;
        case (0b100):
            operation = "lbu";
            break;
        case (0b101):
            operation = "lhu";
            break;
    }
}

private void disJ(String word) {
    StringBuilder sb  = new StringBuilder();
    sb.append(word.substring(0, 1).repeat(12)).append(word, 12, 20)
            .append(word.charAt(11)).append(word, 1, 11).append("0");
    imm = (int) Long.parseLong(String.valueOf(sb),2);

}

private void disS() {
    switch (funct3) {
        case 0b000:
            operation = "sb";
            break;
        case 0b001:
            operation = "sh";
            break;
```

```
        case 0b010:
            operation = "sw";
            break;
    }

    imm = rd | ((imm110 >>> 5) << 5);
}

private void disR(String word) {
    switch (funct3) {
        case 0b000:
            if (word.charAt(6) == '1') operation = "mul";
            else if (word.charAt(1) == '0') operation = "add";
            else operation = "sub";
            break;
        case 0b001:
            if (word.charAt(6) == '1') operation = "mulh";
            else operation = "sll";
            break;
        case 0b010:
            if (word.charAt(6) == '1') operation = "mulsu";
            else operation = "slt";
            break;
        case 0b011:
            if (word.charAt(6) == '1') operation = "mulu";
            else operation = "sltu";
            break;
        case 0b100:
            if (word.charAt(6) == '1') operation = "div";
            else operation = "xor";
        case 0b101:
            if (word.charAt(6) == '1') operation = "divu";
            else if (word.charAt(1) == '0') operation = "srl";
            else operation = "sra";
            break;
        case 0b110:
            if (word.charAt(6) == '1') operation = "rem";
            else operation = "or";
            break;
        case 0b111:
            if (word.charAt(6) == '1') operation = "remu";
            else operation = "and";
            break;
    }

}

private String getRegister(int decimalReg) {
    if (decimalReg == 0) return "zero";
    if (decimalReg == 1) return "ra";
    if (decimalReg == 2) return "sp";
```

```
        if (decimalReg == 3) return "gp";
        if (decimalReg == 4) return "tp";
        if (decimalReg >= 5 && decimalReg <= 7) return "t" + (decimalReg - 5);
        if (decimalReg >= 8 && decimalReg <= 9) return "s" + (decimalReg - 8);
        if (decimalReg >= 10 && decimalReg <= 17) return "a" + (decimalReg - 10);
        if (decimalReg >= 18 && decimalReg <= 27) return "s" + (decimalReg - 16);
        if (decimalReg >= 28 && decimalReg <= 31) return "t" + (decimalReg - 25);
        throw new AssertionError("Unknown register: " + decimalReg);
    }

    private String getRegister(String binReg) {
        return getRegister(Integer.parseInt(binReg, 2));
    }



    public void doDisassemble(String input) throws IOException {
        output.println(".text");

        BufferedInputStream stream = new BufferedInputStream(new
FileInputStream(input));
        elf = new ELF32File(stream);
        elf.setStreamName(input);
        elf.checkHeader();
        elf.getSections();
        elf.readSectionsNames();
        elf.getStringTableToString();
        elf.getSymTable();
        ELF32File elfText = elf.prepareTextSection();
        prepareMarks(elfText);
        elfText = elf.prepareTextSection();
        int bytesRead = 0;
        addr = elf.addr;
        while (bytesRead < elf.textSize) {
            String next = elf.textSectionNext(elfText);
            if (next.endsWith("11")) {// Это не RVC модификация
                String next2 = elf.textSectionNext(elfText);
                disassemble(next2 + next);
                addr += 4;
                bytesRead += 4;
            } else { // Это RVC модификация
                disassemble(next);
                addr += 2;
                bytesRead += 2;
            }
        }


        output.println();
        output.println(".symtab");
        elf.printSymTab(output);
```

```
        output.flush();
        stream.close();

    }

    private void prepareMarks(ELF32File elfText) throws IOException {
        int bytesRead = 0;
        addr = elf.addr;
        while (bytesRead < elf.textSize) {

            String next = elf.textSectionNext(elfText);
            if (next.endsWith("11")) { //не RVC
                String next2 = elf.textSectionNext(elfText);
                String word = next2 + next;
                decimalWord = (int) Long.parseLong(word, 2);
                getUsefulThingsFromWord();
                if (opcode == 0b1101111) { // J-type, jal
                    StringBuilder sb = new StringBuilder();
                    sb.append(word.substring(0, 1).repeat(12)).append(word, 12,
20)
                            .append(word.charAt(11)).append(word, 1,
11).append("0");
                    int imm_j = (int) Long.parseLong(String.valueOf(sb), 2);
                    long jumpTo = addr + imm_j;
                    unknownMarks.add(jumpTo);
                } else if (opcode == 0b1100011) {// B-type
                    StringBuilder sb = new StringBuilder();

sb.append(String.valueOf(word.charAt(0)).repeat(20)).append(word.charAt(24))
                            .append(word, 1, 7).append(word, 20, 25).append("0");
                    int imm_b = (int) Long.parseLong(String.valueOf(sb), 2);
                    long jumpTo = addr + imm_b;
                    unknownMarks.add(jumpTo);
                }
                addr += 4;
                bytesRead += 4;
            }
            else { //Это RVC
                opcode = decimalWord & ((1 << 2) - 1);
                funct3 = decimalWord >>> 13;
                short imm;
                int uimm;

                if (opcode == 0b01 && funct3 == 0b001 || opcode == 0b01 && funct3
== 0b101) { //c.jal or c.j
                    imm = getImmForRVCJumps(next);
                    long jumpTo = addr + imm;
                    unknownMarks.add(jumpTo);
                } else if (opcode == 0b01 && funct3 == 110 || opcode == 0b01 &&
funct3 == 111) { //c.beqz or c.bnez
                    imm = (short)
```

```
Integer.parseInt(String.valueOf(next.charAt(3)).repeat(7) +
                            next.charAt(3) + next.substring(9, 11) +
next.charAt(13) + next.substring(4, 6) +
                            next.substring(11, 13) + "0", 2);
                    long jumpTo = addr + imm;
                    unknownMarks.add(jumpTo);
                }
                addr += 2;
                bytesRead += 2;
            }
        }
    }
}
```

elf/ELF32File.java

```
package elf;

import java.io.*;
import java.util.*;

public class ELF32File {

    // Смещение таблицы заголовков секций от начала файла в байтах.
    // Если у файла нет таблицы заголовков секций, это поле содержит 0.
    public int e_shoff;
    public String e_shentsize;
    public int e_shnum;
    public String e_shstrndx;
    public Parser parser;
    public Section[] sections;
    public String streamName;
    public Map<Integer, String> strings = new LinkedHashMap<>();
    public List<SymTabString> symbolTable = new ArrayList<>();
    public StringBuilder binaryStrings = new StringBuilder();
    public int textSize;
    public long addr;
    public Map<Long, String> symTableValues = new HashMap<>();



    public ELF32File(BufferedInputStream stream) {
        this.parser = new Parser(stream);
    }

    public void setStreamName(String s) {
        this.streamName = s;
    }

    public void checkHeader() throws IOException {
```

```
        String[] ident = parser.nextNHexBytes(16);
        if (!(ident[0].equals("7f") && ident[1].equals("45")
                && ident[2].equals("4c") && ident[3].equals("46"))) {
            throw new InputMismatchException("Invalid magic numbers");
        }

        if (!ident[4].equals("01")) {
            throw new InputMismatchException("Not a 32bit file");
        }

        if (!ident[5].equals("01")) {
            throw new InputMismatchException("Not a littleEndian file");
        }

        parser.skipNBytes(2);

        String e_type = parser.readTwoBytes();
        if (!e_type.equals("00f3")) {
            throw new InputMismatchException("Not a RISC-V file");
        }

        String version = parser.readFourBytes();
        if (Integer.valueOf(version, 16).equals(0)) {
            throw new InputMismatchException("Incorrect version");
        }

        // Пропускаем информация e_entry и про program-header
        parser.skipNBytes(8);

        e_shoff = Integer.valueOf(parser.readFourBytes(), 16);
        if (e_shoff == 0){
            throw new InputMismatchException("elf.Section header doesn't exists");
        }

        parser.skipNBytes(10);
        e_shentsize = parser.readTwoBytes();

        e_shnum = Integer.valueOf(parser.readTwoBytes(), 16);

        e_shstrndx = parser.readTwoBytes();
    }

    public void getSections() throws IOException {
        int bytesToSkip = (e_shoff - 52);
        //Видимо, больше никакая информация, идущая до SectionHeader нам не
интересна, поэтому можно ее пропустить
        parser.skipNBytes(bytesToSkip);

        sections = new Section[e_shnum];
        for (int i = 0; i < e_shnum; i++) {
            Section section = new Section();
```

```
            section.name = Integer.valueOf(parser.readFourBytes(), 16);
            int sTypeNum = Integer.parseInt(parser.readFourBytes(), 16);
            section.type = SH_TYPE.values()[ sTypeNum <= 18 ? sTypeNum : 19];
            parser.skipNBytes(4);
            section.addr = parser.readFourBytes();
            section.offset = Integer.valueOf(parser.readFourBytes(), 16);
            section.size = Integer.valueOf(parser.readFourBytes(), 16);
            section.link = parser.readFourBytes();
            parser.skipNBytes(8);
            section.entsize = parser.readFourBytes();
            sections[i] = section;
        }
    }

    public void readSectionsNames() throws IOException {
        if (streamName == null) {
            throw new InputMismatchException("Undefined stream name, firstly you
have to set it using setStreamName");
        }
        try {
            BufferedInputStream newStream = new BufferedInputStream(new
FileInputStream(streamName));

            try {
                ELF32File elfShStrTab = new ELF32File(newStream);
                int shstrStarts = sections[Integer.valueOf(e_shstrndx,
16)].offset;
                elfShStrTab.parser.skipNBytes(shstrStarts);

                int nameOffset = 0;
                for (int i = 0; i < e_shnum; i++) {
                    String name = elfShStrTab.parser.nextNullTermString();
                    if (name == null) {
                        for (int j = 0; j < e_shnum; j++) {
                            if (sections[j].type == SH_TYPE.SHT_NULL) {
                                sections[j].stringName = "0";
                                nameOffset++;
                                break;
                            }
                        }
                    } else {
                        for (int j = 0; j < e_shnum; j++) {
                            if (sections[j].name == nameOffset) {
                                sections[j].stringName = name;
                                nameOffset += name.length() + 1;
                                break;
                            }
                        }
                    }
                }
            } finally {
```

```
                newStream.close();
            }
        } catch (IOException e) {
            System.out.println("Something went wrong: " + e);
        }
    }


    public void getStringTableToString() {
        if (streamName == null) {
            throw new InputMismatchException("Undefined stream name, firstly you
have to set it using setStreamName");
        }
        try {
            BufferedInputStream newStream = new BufferedInputStream(new
FileInputStream(streamName));

            try {
                ELF32File elfStrTab = new ELF32File(newStream);
                int bytesToSkip = 0;
                int strTabSize = 0;
                for (int i = 0; i < e_shnum; i++) {
                    if (sections[i].stringName != null &&
sections[i].stringName.equals(".strtab")) {
                        bytesToSkip = sections[i].offset;
                        strTabSize = sections[i].size;
                        break;
                    }
                }
                if (bytesToSkip == 0 || strTabSize == 0) {
                    throw new InputMismatchException("No .strtab found or it's
empty");
                } else {
                    elfStrTab.parser.skipNBytes(bytesToSkip);
                }
                int bytesRead = 0;

                while (bytesRead < strTabSize) {
                    binaryStrings.append((char) elfStrTab.parser.nextByte());
                    bytesRead++;
                }

            } finally {
                newStream.close();
            }
        } catch (IOException e) {
            System.out.println("Something went wrong: " + e);
        }
    }

    public void getSymTable() throws InputMismatchException{
```

```
        getStringTableToString();
        Map<Integer, String> compareType = new HashMap<>(Map.of(0, "NOTYPE", 1,
"OBJECT", 2, "FUNC",
                3, "SECTION", 4, "FILE", 5, "COMMON",
                6, "TLS", 10, "LOOS", 12, "HIOS",
                13, "LOPROC"));
        compareType.put(15, "HIPROC");

        Map<Integer, String> compareBinding = new HashMap<>(Map.of(0, "LOCAL", 1,
"GLOBAL",
                2, "WEAK", 10, "LOOS", 12, "HIOS", 13, "LOWPROC", 15, "HIPROC"));

        Map<Integer, String> compareVis = new HashMap<>(Map.of(0, "DEFAULT", 1,
"INTERNAL",
                2, "HIDDEN", 3, "PROTECTED"));
        if (streamName == null) {
            throw new InputMismatchException("Undefined stream name, firstly you
have to set it using setStreamName");
        }
        try {
            BufferedInputStream newStream = new BufferedInputStream(new
FileInputStream(streamName));

            try {
                ELF32File elfSymTab = new ELF32File(newStream);
                int bytesToSkip = 0;
                int bytesRead = 0;
                int symTabSize = 0;
                for (int i = 0; i < e_shnum; i++) {
                    if (sections[i].stringName.equals(".symtab")) {
                        bytesToSkip = sections[i].offset;
                        symTabSize = sections[i].size;
                        break;
                    }
                }
                if (bytesToSkip == 0 || symTabSize == 0) {
                    throw new InputMismatchException("No .symtab found or it's
empty");
                }
                elfSymTab.parser.skipNBytes(bytesToSkip);
                int counter = 0;
                while (bytesRead < symTabSize) {
                    SymTabString symTab = new SymTabString();
                    symTab.name =
Integer.valueOf(elfSymTab.parser.readFourBytes(), 16);
                    bytesRead += 4;
                    if (symTab.name != 0) {
//                        symTab.stringName = strings.get(symTab.name);
                        StringBuilder stringName = new StringBuilder();
                        for (int i = symTab.name; i < binaryStrings.length(); i++)
{
```

```
                            if (binaryStrings.charAt(i) != 0) {
                                stringName.append(binaryStrings.charAt(i));
                            } else {break;}
                        }
                        symTab.stringName = String.valueOf(stringName);
                    }
                    symTab.value =
Long.parseLong(elfSymTab.parser.readFourBytes(), 16);
                    bytesRead += 4;
                    symTab.size =
Integer.valueOf(elfSymTab.parser.readFourBytes(), 16);
                    bytesRead += 4;
                    int info = elfSymTab.parser.nextByte();
                    bytesRead++;
                    int bind = info >> 4;
                    symTab.bind = compareBinding.get(bind);
                    int type = info & 0xf;
                    symTab.type = compareType.get(type);
                    int other = elfSymTab.parser.nextByte();
                    bytesRead++;
                    int vis = other & 0x3;
                    symTab.vis = compareVis.get(vis);
                    int index = Integer.valueOf(elfSymTab.parser.readTwoBytes(),
16);
                    if (index == 0) symTab.index = "UNDEF";
                    else if (index == 0xfff1) symTab.index = "ABS";
                    else if (index == 0xff00) symTab.index = "LORESERVE";
                    else if (index == 0xff01) symTab.index = "AFTER";
                    else if (index == 0xff1f) symTab.index = "HIPROC";
                    else if (index == 0xff20) symTab.index = "LOOS";
                    else if (index == 0xff3f) symTab.index = "HIOS";
                    else if (index == 0xfff2) symTab.index = "COMMON";
                    else if (index == 0xffff) symTab.index = "XINDEX";
                    else if (index > 0xff20 && index < 0xff3f) symTab.index =
"OSSPEC";
                    else symTab.index = String.valueOf(index);
                    bytesRead += 2;
                    symTab.num = counter;
                    counter++;
                    symbolTable.add(symTab);
                }
            } finally {
                newStream.close();
            }
        } catch (IOException e) {
            System.out.println("Something went wrong: " + e);
        }
    }


    public ELF32File prepareTextSection() throws IOException {
```

```
        if (streamName == null) {
            throw new InputMismatchException("Undefined stream name, firstly you
have to set it using setStreamName");
        }
        BufferedInputStream newStream = new BufferedInputStream(new
FileInputStream(streamName));
        ELF32File elfText = new ELF32File(newStream);
            int bytesToSkip = 0;
            for (int i = 0; i < e_shnum; i++) {
                if (sections[i].stringName != null &&
sections[i].stringName.equals(".text")) {
                    bytesToSkip = sections[i].offset;
                    textSize = sections[i].size;
                    addr = Long.parseLong(sections[i].addr,16);
                    break;
                }
            }
            if (bytesToSkip == 0 || textSize == 0) {
                throw new InputMismatchException("No .text found or it's empty");
            } else {
                elfText.parser.skipNBytes(bytesToSkip);
            }

        for (SymTabString symTabString : symbolTable) {
            symTableValues.put(symTabString.value, symTabString.stringName);
        }


        return elfText;
    }

    public String make16bit(String str){
        StringBuilder bin = new StringBuilder();
        bin.append("0".repeat(Math.max(0, 16 - str.length())));
        bin.append(str);
        return String.valueOf(bin);
    }

    public String textSectionNext(ELF32File elf) throws IOException {
        String str = elf.parser.readTwoBytes();
        return make16bit(Long.toBinaryString(Long.parseLong(str, 16)));
    }

    public void printSymTab(PrintWriter output) {
        output.printf("%s %-15s %7s %-8s %-8s %-8s %6s %s\n",
                "Symbol", "Value", "Size", "Type", "Bind", "Vis", "Index",
"Name");
        for (SymTabString s : symbolTable) {
            output.printf("[%4s] 0x%-15X %5s %-8s %-8s %-8s %6s %s\n",
                    s.num, s.value, s.size, s.type, s.bind, s.vis, s.index,
s.stringName);
```

```
        }
    }


    public String getSym(long addr) {
        for (SymTabString s : symbolTable) {
            if (s.value == addr && s.type.equals("FUNC")) {
                return s.stringName;
            }
        }
        return null;
    }
}
```

elf/Section.java

```
package elf;

public class Section {
    public int name; //Смещение относительно начала таблицы названий
    public SH_TYPE type;
    public int offset;
    public int size;
    public String link;
    public String entsize;
    public String stringName;
    public String addr = "0";

    @Override
    public String toString() {
        return addr + " " + type + " " +  offset + " " + size + " " + link + " " +
entsize
                + " " + stringName;
    }
}
```

elf/SH_TYPE.java

```
package elf;

public enum SH_TYPE {
    SHT_NULL, SHT_PROGBITS, SHT_SYMTAB, SHT_STRTAB, SHT_RELA, SHT_HASH,
SHT_DYNAMIC,
    SHT_NOTE, SHT_NOBITS, SHT_REL, SHT_SHLIB, SHT_DYNSYM, SHT_UNDEF1, SHT_UNDEF2,
    SHT_INIT_ARRAY, SHT_FINI_ARRAY, SHT_PREINIT_ARRAY, SHT_GROUP,
SHT_SYMTAB_SHNDX, SHT_BIGNUMS
}
```

elf/SymTabString.java

```
package elf;

public class SymTabString {
    int num;
    long value;
    int size;
    String type;
    String bind;
    String vis;
    String index;
    int name;
    String stringName;

    @Override
    public String toString() {
        if (stringName == null) {
            stringName = "";
        }
        return num + " " + value + " " + size + " " + type + " " + bind + " " +
vis + " " + index + " " + stringName;
    }
}
```

elf/Parser.java

```
package elf;

import java.io.BufferedInputStream;
import java.io.IOException;

public class Parser {
    BufferedInputStream stream;
    int curOffset = 0;

    public Parser(BufferedInputStream stream) {
        this.stream = stream;
    }

    public int nextByte() throws IOException {
        curOffset++;
        return stream.read();
    }

    public String nextNullTermString() throws IOException {
        int b = 0;
        StringBuilder sb = new StringBuilder();
        do {
            b = stream.read();
            curOffset++;
            sb.append((char) b);
        } while (b != 0);
```

```
        return sb.length() > 1 ? String.valueOf(sb.substring(0, sb.length() - 1))
: null;
    } // Так можно прочитать названия секция в shstrtab

    public void skipNBytes(int n) throws IOException {
        if (n != 0) {
            byte[] toSkip = stream.readNBytes(n);
            curOffset += n;
        }
    }

    public String[] nextNHexBytes(int n) throws IOException {
        String[] hexBytes = new String[n];
        for (int i = 0; i < n; i++) {
            int hexByte = stream.read();
            curOffset++;
            hexBytes[i] = String.format("%2s", Integer.toHexString(hexByte &
0xFF)).replace(' ', '0');
        }
        return hexBytes;
    }

    // Читаем байты с учетом little-endian
    public String readTwoBytes() throws IOException {
        int b1 = stream.read();
        int b2 = stream.read();
        curOffset += 2;
        return String.format("%2s", Integer.toHexString(b2 & 0xFF)).replace(' ',
'0') +
                String.format("%2s", Integer.toHexString(b1 & 0xFF)).replace(' ',
'0');
    }

    public String readFourBytes() throws IOException {
        int b1 = stream.read();
        int b2 = stream.read();
        int b3 = stream.read();
        int b4 = stream.read();
        curOffset += 4;
        return String.format("%2s", Integer.toHexString(b4 & 0xFF)).replace(' ',
'0') +
                String.format("%2s", Integer.toHexString(b3 & 0xFF)).replace(' ',
'0') +
                String.format("%2s", Integer.toHexString(b2 & 0xFF)).replace(' ',
'0') +
                String.format("%2s", Integer.toHexString(b1 & 0xFF)).replace(' ',
'0');
    }
}
```