САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA»

Выполнил(а): Миленин Иван Александрович

студ. гр. М3135

Санкт-Петербург

2021

Цель работы: знакомство со системой набора команд RISC-V.

Инструментарий и требования к работе: работа выполняется в языке Java.

Теоретическая часть

RISC-V – это открытая и свободная система команд. Данный проект был создан в 2010 году исследователями из компании «Computer Science Division», после чего в 2015 году была создана ассоциация «RISC-V International» со штабом-квартирой в Цюрихе.

Уникальность и успех данного проекта заключается в свободе использования и открытости кода. Вышеперечисленные преимущества привлекают маленькие стартапы и компании, финансы которых не сильно велики. Также стоит отметить, что проект RISC-V поддерживают крупные компании, такие как Nvidia, Google и т.д.

Архитектура RISC-V имеет определенный набор команд, связанных с арифметическими и битовыми операциями на регистрах, служебные инструкции и работа с памятью. В базовое подмножество команд входит определенное количество регистров, а именно специальный регистр x0, целочисленный регистр общего назначения x1-x31 (их количество равно 31), регистр счетчика команд.

Следует отметить, что разрядность регистровых операций всегда соответствует размеру регистра, а одни и те же значения в регистрах могут трактоваться целыми числами как со знаком, так и без знака.

На данный момент существуют различные базовые наборы RISC-V, такие как:

- RV32I – 32-разрядный базовый набор целочисленных инструкций;
- RV64I – 64-разрядный базовый набор целочисленных команд;

- RV32E – 32-разрядный базовый набор целочисленных инструкций и 16 регистров с меньшим набором команд;
- RV128I - 128-разрядный базовый набор целочисленных инструкций;

Существуют различные типы кодирования инструкций (см. рисунок №1). Следующие конструкции состоят из 32 бит, в которые входят такие элементы, как opcode (Благодаря этому значению определяется тип кодирования инструкции), rd (Регистр, в который записывается операция), rs1 и rs2 (Регистры, с которыми производятся операции), funct3 и funct7 (коды команд).

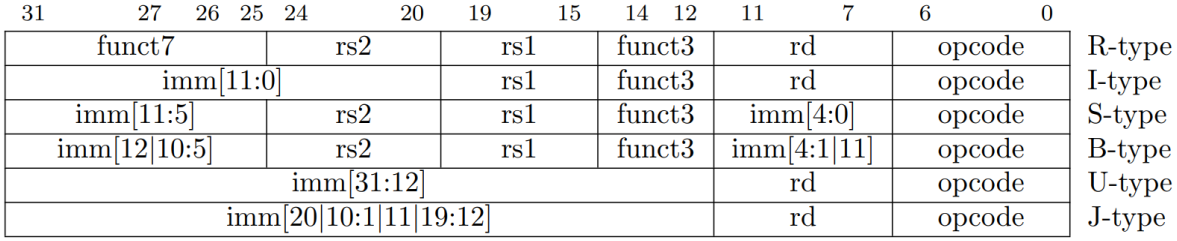

Рисунок №1 – типы кодирования инструкций RISC-V

Также существуют константы (imm). В каждом из типов существует определенный тип определения констант (см. рисунок №2). Написанные поля помечены битами команд, используемыми для построения их значения и последующего использования.

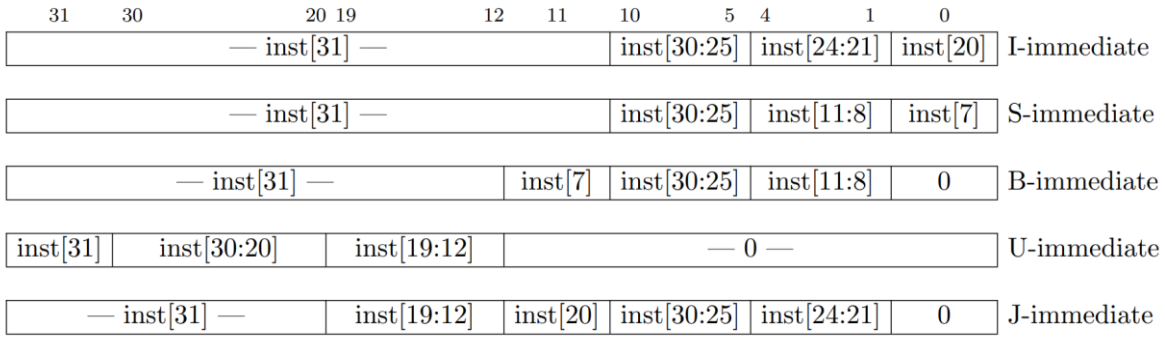

Рисунок №2 – типы констант в конструкциях RISC-V

Elf-файл – это формат исполняемых связываемых файлов (Executable and Linkable Format). Спецификация данного формата позволяет системе корректно работать с машинным кодом, содержащимся в файле. Данный файл впервые был разработан Лабораторией Юникс, как часть двоичного интерфейса операционной системы UNIX System V. Идея ELF файла была в том, чтобы упростить разработку, предоставив разработчикам более четкую и понятную структуру файла.

Теперь необходимо описать формат ELF файла (см. рисунок №3). Каждый такой файл состоит из заголовка файла (ELF Header), таблицы заголовков программы (program header table) и таблицы заголовков секций (section header table). Также в ELF файлах существуют сегменты – непрерывные области адресного пространства. Бывает и то, что сегменты разбиты на определенные части – секции. Чаще всего кодовый сегмент состоит из секции процедуры инициализации, секции связок, основного кода программы и секции процедуры финализации.

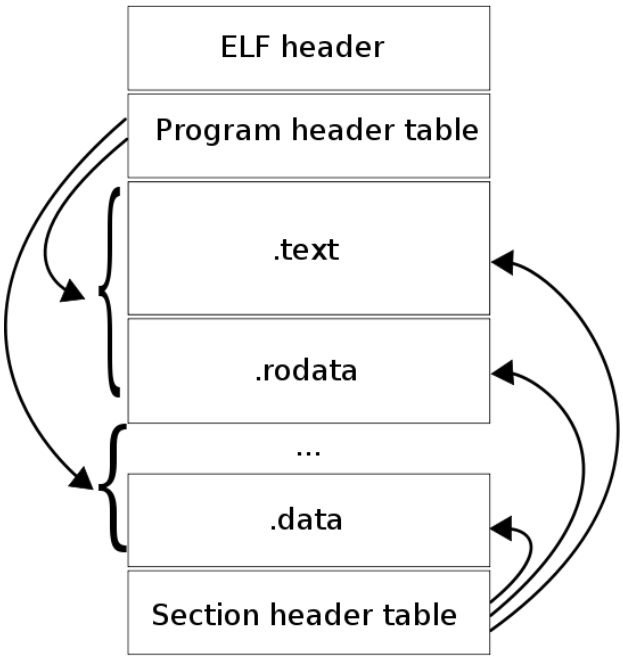

Рисунок 3 – структура ELF файла.

ELF Header содержит в себе структур основные характеристики файла, а именно тип, версия формата, архитектура процессора,

виртуальный адрес точки входа, размеры и смещения остальных частей файла.

Program header table находится сразу же после заголовка файла. В свою очередь таблица заголовков программы содержит описание отдельных сегментов и их свойств, а именно Типа сегмента, его расположение, точка входа, размер и флаги доступа к нему.

Section header table занимается характеристикой различных секций файла. Компоновщик оптимально собирает все эти секции и размещает в файле.

Практическая часть

Для получения таблицы символов и дизассемблера основными элементами для получения данных будут сдвиги относительно начала кода. Нижеприведенная программа начинается с того, что идет определение, является ли данный нам файл именно ELF файлом. После этого идет поиск смещения таблицы заголовков секций. Далее мы переходим к этому смещению и производим поиск заголовка №2, а именно секции, которая содержит таблицу строк, где можно получить размер всей секции, размер одного элемента секции и сдвиг Symbol Table. Также необходимо найти смещение и размер секции .text, которая содержит в себе инструкции RISC V (В секции №3). После этого необходимо считать данные из таблицы символов, где хранятся следующие значения: Value, size, type, bind, visibility, index, name. Причем значения из name необходимо соответственно передать в дизассемблер.

Далее необходимо считывать данные из секции .text. Размер инструкции в данной секции составляет 4 байта. Сперва необходимо считать opcode для определения типа инструкции RISC-V. Далее необходимо считывать funct3, funct7 и т.д для определения функции, обращая внимание на таблицы, приведенную ниже (см. Рисунок №4). После

этого произвести по необходимости подсчет констант и произвести вывод .symtab и дизассемблера.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Рисунок №4 – базовый набор инструкций RISC-V

Ниже приведён результат работы программы:

```
Symbol Table (.symtab)
Symbol  Value   Size    Type    Bind    Vis     Index   Name
[0]     0x0     0       NOTYPE  LOCAL   DEFAULT UNDEF
[1]     0x0     0       FILE    LOCAL   DEFAULT ABS     test2.c
[2]     0x30    0       NOTYPE  LOCAL   DEFAULT 2       .LBB0_1
[3]     0x40    0       NOTYPE  LOCAL   DEFAULT 2       .LBB0_2
[4]     0x58    0       NOTYPE  LOCAL   DEFAULT 2       .LBB0_3
[5]     0x68    0       NOTYPE  LOCAL   DEFAULT 2       .LBB0_4
[6]     0x0     124     FUNC    GLOBAL  DEFAULT 2       main
```

```
0x00000000:    <test2.c >      ADDI    a2      a2      -32
0x00000004:                    SW      a2      a1      28
0x00000008:                    SW      a2      a8      24
0x0000000c:                    ADDI    a8      a2      32
0x00000010:                    ADDI    a10     zero    0
0x00000014:                    SW      a8      a10     -12
0x00000018:                    ADDI    a11     zero    64
0x0000001c:                    SW      a8      a11     -16
0x00000020:                    SW      a8      a10     -20
0x00000024:                    ADDI    a10     zero    1
0x00000028:                    SW      a8      a10     -24
0x0000002c:    <LOC_0000002c> JAL     zero    0
0x00000030:    <.LBB0_1 >      LW      a10     a8      -24
0x00000034:                    LW      a11     a8      -16
0x00000038:    <LOC_00000038> BGE     a10     a11     0
0x0000003c:    <LOC_0000003c> JAL     zero    0
0x00000040:    <.LBB0_2 >      LW      a10     a8      -24
0x00000044:                    MUL     a10     a10     a10
0x00000048:                    LW      a11     a8      -20
0x0000004c:                    ADD     a10     a11     a10
0x00000050:                    SW      a8      a10     -20
0x00000054:    <LOC_00000054> JAL     zero    0
0x00000058:    <.LBB0_3 >      LW      a10     a8      -24
0x0000005c:                    ADDI    a10     a10     1
0x00000060:                    SW      a8      a10     -24
0x00000064:    <LOC_00000064> JAL     zero    0
0x00000068:    <.LBB0_4 >      LW      a10     a8      -20
0x0000006c:                    LW      a8      a2      24
0x00000070:                    LW      a1      a2      28
0x00000074:                    ADDI    a2      a2      32
0x00000078:    <LOC_00000078> JALR    zero    a1      0
```

Листинг

Main.java; Java11;

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
public class Main {

    private static final Map<String, String> symbol_types = new HashMap<>();
    private static final Map<String, String> symbol_binds = new HashMap<>();
    private static final Map<String, String> symbol_visibility = new HashMap<>();
    private static final Map<String, String> id_of_names = new HashMap<>();
    private static int position = 0;
    private static int text_sh_offset;      //Смещение .text
    private static int text_sh_size;        //Размер .text
    private static int sht_strtab_offset;//Смещение таблицы строк от начала
    private static int e_shoff;         //Смещение таблицы заголовков секций
    private static int sh_offset;      //Смещение symTable
    private static int sh_size;         //Размер symTable
    private static int sh_entsize;      //Размер одной записи в symTable
    private static boolean hasOutFile = false;
    private static int[] tableOfElfFile;
    private static String[][] assembler_commands;
    private static String[][] symTable;

    public static void main(String[] args) throws IOException {
        if (args.length == 0) {             //т.к нет названия файла
            throw new FileNotFoundException();
        } else if (args.length == 2) {
            hasOutFile = true;
        }
        downloadingMaps();
        tableOfElfFile = readElfFile(args[0]);
        if (!thisIsElfFile()) {//Проверка на то, является ли данный файл ELF
файлом
            System.out.println("This file is not .ELF file.");
            throw new FileNotFoundException();
        }
        position += 28; //Расстояние между смещение таблицы заголовков секций и
сигнатуры файла
        e_shoff = gettingByteLine(4);
        position = e_shoff;
        int temp_type_of_section = 0;
        while (temp_type_of_section != 3) { //Нахождение секции с таблицой строк
            position += 4;
            temp_type_of_section = gettingByteLine(4);
            position += 8;
            sht_strtab_offset = gettingByteLine(4);
            position += 20;
        }
        int tempPosition = position;
        gettingInformationAboutSymTable();
        gettingSymTable();
        int sh_name = 0; //смещение ".text"
        position = tempPosition;
        position = gettingOffset();
        int partSectionSet = 0;
        StringBuilder tempCheckingInSectionName = new StringBuilder();
        while (!tempCheckingInSectionName.toString().contains(".text")) {
            tempCheckingInSectionName = new StringBuilder();
            do {
                sh_name++;
                tempCheckingInSectionName.append((char) tableOfElfFile[position]);
            } while (tableOfElfFile[position++] != 0);
```

```
        }

        position = e_shoff;

        while (partSectionSet + 6 != sh_name) {//поиск информации о секции .text
            partSectionSet = gettingByteLine(4);
            position += 36;
        }
        position -= 36;
        position += 12; //Отправляемся в sh_offset
        text_sh_offset = gettingByteLine(4);
        text_sh_size = gettingByteLine(4);
        completingTextAndSections();
        if (!hasOutFile) outPrint();
        else outFilePrint(args[1]);
    }

    private static int gettingOffset() {
        int temp_type_of_section = 0;
        int tempOffset = 0;
        while (temp_type_of_section != 3 && temp_type_of_section != -1) {
            position += 4;
            temp_type_of_section = gettingByteLine(4);
            position += 8;
            tempOffset = gettingByteLine(4);
            position += 20;
        }
        if (temp_type_of_section == -1) return sht_strtab_offset;
        return tempOffset;
    }

    private static void outFilePrint(String nameOfFile) {
        try (PrintStream writer = new PrintStream(nameOfFile)) {
            writer.print("Symbol Table (.symtab)\n");
            writer.printf("%-8s%-20s%-11s%-11s%-11s%-11s%-11s%-11s%n", "Symbol",
"Value", "Size", "Type", "Bind", "Vis"
                    , "Index", "Name");
            for (int i = 0; i < symTable.length; i++) {
                writer.printf("%-8s%-20s%-11s%-11s%-11s%-11s%-11s%-11s%n", "[" +
symTable[i][0] + "]", "0x" + symTable[i][1], symTable[i][2]
                        , symTable[i][3], symTable[i][4], symTable[i][5],
symTable[i][6], symTable[i][7]);
            }
            for (int i = 0; i < 4; i++) System.out.println();
            for (int i = 0; i < text_sh_size / 4; i++) {
                String[] s = new String[7];
                assembler_commands[i][0] = "0x" + assembler_commands[i][0] + ":";
                s[0] = assembler_commands[i][0];
                s[1] = "";
                if (assembler_commands[i][1] != null) s[1] =
assembler_commands[i][1];
                for (int j = 2; j < 6; j++) {
                    if (assembler_commands[i][j] == null) assembler_commands[i][j]
= "";
                    if (assembler_commands[i][j].equals("a0"))
assembler_commands[i][j] = "zero";
                    s[j] = assembler_commands[i][j];
                }
                writer.printf("%-15s%-27s%-8s%-8s%-8s%-8s%n", s[0], s[1], s[2],
```

```
s[3], s[4], s[5]);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private static void outPrint() {
            System.out.println("Symbol Table (.symtab)");
            System.out.printf("%-8s%-20s%-11s%-11s%-11s%-11s%-11s%-11s%n", "Symbol",
"Value", "Size", "Type", "Bind", "Vis"
                    , "Index", "Name");
            for (int i = 0; i < symTable.length; i++) {
                System.out.printf("%-8s%-20s%-11s%-11s%-11s%-11s%-11s%-11s%n", "[" +
symTable[i][0] + "]", "0x" + symTable[i][1], symTable[i][2]
                        , symTable[i][3], symTable[i][4], symTable[i][5],
symTable[i][6], symTable[i][7]);
            }
            for (int i = 0; i < 4; i++) System.out.println();
            for (int i = 0; i < text_sh_size / 4; i++) {
                String[] s = new String[7];
                assembler_commands[i][0] = "0x" + assembler_commands[i][0] + ":";
                s[0] = assembler_commands[i][0];
                s[1] = "";
                if (assembler_commands[i][1] != null) s[1] = assembler_commands[i][1];
                for (int j = 2; j < 6; j++) {
                    if (assembler_commands[i][j] == null) assembler_commands[i][j] =
"";
                    if (assembler_commands[i][j].equals("a0"))
assembler_commands[i][j] = "zero";
                    s[j] = assembler_commands[i][j];
                }
                System.out.printf("%-15s%-27s%-8s%-8s%-8s%-8s%n", s[0], s[1], s[2],
s[3], s[4], s[5]);
            }
        }

        private static void completingTextAndSections() {
            int counts = text_sh_size / 4;//Оазмер команды 4 байта
            assembler_commands = new String[counts][6]; //0xX code, mark, FUNC, x1,
x2, x3, comm
            position = text_sh_offset;//переходим в сдвиг
            for (int i = 0; i < counts; i++) {
                StringBuilder command_code = new StringBuilder();
                for (int j = 0; j < 4; j++) {
                    command_code.insert(0,
Integer.toBinaryString(tableOfElfFile[position++]));
                    while (command_code.length() < 8 * (j + 1)) command_code.insert(0,
"0");
                }
                baseInstructionSetRV32I(command_code.toString(), i);
                StringBuilder outOfNumCommand = new
StringBuilder(Integer.toHexString(i * 4));
                while (outOfNumCommand.length() < 8) outOfNumCommand.insert(0, "0");
                assembler_commands[i][0] = outOfNumCommand.toString();
                if (assembler_commands[i][2].equals("JAL") ||
assembler_commands[i][2].equals("JALR") || assembler_commands[i][2].equals("BGE"))
{
                    assembler_commands[i][1] = "<LOC_" + assembler_commands[i][0] +
```

```
">";
            }//метки
            if (checkingInMapOfNameSymTab(assembler_commands[i][0]))
{//обозначение меток из symTable
                assembler_commands[i][1] =
id_of_names.get(assembler_commands[i][0]);
            }
        }


    }

    private static void baseInstructionSetRV32I(String command_code, int
numOfOperation) {
        String opcode = command_code.substring(25, 32);
        String func3 = command_code.substring(17, 20);
        if (opcode.equals("0110111")) {
            assembler_commands[numOfOperation][2] = "LUI";
            uDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010111")) {
            assembler_commands[numOfOperation][2] = "AUIPC";
            uDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1101111")) {
            assembler_commands[numOfOperation][2] = "JAL";
            jDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100111")) {
            assembler_commands[numOfOperation][2] = "JALR";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("000")) {
            assembler_commands[numOfOperation][2] = "BEQ";
            bDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("001")) {
            assembler_commands[numOfOperation][2] = "BNE";
            bDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("100")) {
            assembler_commands[numOfOperation][2] = "BLT";
            bDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("101")) {
            assembler_commands[numOfOperation][2] = "BGE";
            bDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("110")) {
            assembler_commands[numOfOperation][2] = "BLTU";
            uDecoding(command_code, numOfOperation);
        } else if (opcode.equals("1100011") && func3.equals("111")) {
            assembler_commands[numOfOperation][2] = "BGEU";
            uDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0000011") && func3.equals("000")) {
            assembler_commands[numOfOperation][2] = "LB";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0000011") && func3.equals("001")) {
            assembler_commands[numOfOperation][2] = "LH";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0000011") && func3.equals("010")) {
            assembler_commands[numOfOperation][2] = "LW";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0000011") && func3.equals("100")) {
            assembler_commands[numOfOperation][2] = "LBU";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0000011") && func3.equals("101")) {
```

```
            assembler_commands[numOfOperation][2] = "LHU";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0100011") && func3.equals("000")) {
            assembler_commands[numOfOperation][2] = "SB";
            sDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0100011") && func3.equals("001")) {
            assembler_commands[numOfOperation][2] = "SH";
            sDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0100011") && func3.equals("010")) {
            assembler_commands[numOfOperation][2] = "SW";
            sDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("000")) {//////////
            assembler_commands[numOfOperation][2] = "ADDI";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("010")) {
            assembler_commands[numOfOperation][2] = "SLTI";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("011")) {
            assembler_commands[numOfOperation][2] = "SLTIU";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("100")) {
            assembler_commands[numOfOperation][2] = "XORI";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("110")) {
            assembler_commands[numOfOperation][2] = "ORI";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("111")) {
            assembler_commands[numOfOperation][2] = "ANDI";
            iDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("001")) {
            assembler_commands[numOfOperation][2] = "SLLI";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("101") &&
command_code.charAt(1) == '0') {
            assembler_commands[numOfOperation][2] = "SRLI";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0010011") && func3.equals("101") &&
command_code.charAt(1) == '1') {
            assembler_commands[numOfOperation][2] = "SRAI";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("000") &&
command_code.charAt(1) == '0' && command_code.charAt(6) != '1') {////////
            assembler_commands[numOfOperation][2] = "ADD";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("000") &&
command_code.charAt(1) == '1' && command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SUB";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("001") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SLL";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("010") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SLT";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("011") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SLTU";
```

```
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("100") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "XOR";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("101") &&
command_code.charAt(1) == '0' && command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SRL";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("101") &&
command_code.charAt(1) == '1' && command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "SRA";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("110") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "OR";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("111") &&
command_code.charAt(6) != '1') {
            assembler_commands[numOfOperation][2] = "AND";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0001111")) {
            assembler_commands[numOfOperation][2] = "FENCE";
        } else if (opcode.equals("1110011") && command_code.charAt(11) == '0') {
            assembler_commands[numOfOperation][2] = "ECALL";
        } else if (opcode.equals("1110011") && command_code.charAt(11) == '1') {
            assembler_commands[numOfOperation][2] = "EBREAK";
        } else if (opcode.equals("0110011") && func3.equals("000") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "MUL";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("001") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "MULH";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("010") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "MULHSU";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("011") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "MULHU";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("100") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "DIV";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("101") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "DIVU";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("110") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "REM";
            rDecoding(command_code, numOfOperation);
        } else if (opcode.equals("0110011") && func3.equals("111") &&
command_code.charAt(6) == '1') {///////
            assembler_commands[numOfOperation][2] = "REMU";
            rDecoding(command_code, numOfOperation);
```

```
        } else {
            assembler_commands[numOfOperation][2] = "UNKNOWN COMMAND";
        }
    }

    private static void downloadingMaps() {
        String[] typeArrKey = new String[]{"0", "1", "2", "3", "4", "5", "a", "c",
"d", "f"};
        String[] typeArrValue = new String[]{"NOTYPE", "OBJECT", "FUNC",
"SECTION", "FILE", "COMMON", "LOOS", "HIOS"
                , "LOPROC", "HIPROC"};
        String[] bindArrKey = new String[]{"0", "1", "2", "a", "c", "d", "f"};
        String[] bindArrValue = new String[]{"LOCAL", "GLOBAL", "WEAK", "LOOS",
"HIOS", "LOPROC", "HIPROC"};
        String[] visArrKey = new String[]{"0", "1", "2", "3"};
        String[] visArrValue = new String[]{"DEFAULT", "INTERNAL", "HIDDEN",
"PROTECTED"};
        for (int i = 0; i < typeArrKey.length; i++) {
            symbol_types.put(typeArrKey[i], typeArrValue[i]);
        }
        for (int i = 0; i < bindArrKey.length; i++) {
            symbol_binds.put(bindArrKey[i], bindArrValue[i]);
        }
        for (int i = 0; i < visArrKey.length; i++) {
            symbol_visibility.put(visArrKey[i], visArrValue[i]);
        }
    }

    private static void gettingSymTable() {
        int countsOfElements = sh_size / sh_entsize;
        position = sh_offset;
        symTable = new String[countsOfElements][8]; //num, value, size, type,
bind, visibility, index, name
        for (int i = 0; i < countsOfElements; i++) {
            symTable[i][0] = Integer.toString(i);
            int tempName = gettingByteLine(4);  //Смещение относительно таблдицы
строк для n
            // олучение нимени файлов и m.n
            symTable[i][1] = Integer.toHexString(gettingByteLine(4));//value
            symTable[i][2] = Integer.toString(gettingByteLine(4));//size
            StringBuilder struct_sym_info = new
StringBuilder(Integer.toHexString(tableOfElfFile[position++]));
            if (struct_sym_info.length() == 1) {
                struct_sym_info.insert(0, '0');
            }
            symTable[i][3] =
symbol_types.get(Character.toString(struct_sym_info.charAt(1)));//type
            symTable[i][4] =
symbol_binds.get(Character.toString(struct_sym_info.charAt(0)));//bind
            symTable[i][5] =
symbol_visibility.get(Integer.toString(gettingByteLine(1)));
            symTable[i][6] = Integer.toString(gettingByteLine(2));
            if (symTable[i][3].equals("FILE")) {
                symTable[i][6] = "ABS";//index
            } else if (symTable[i][6].equals("0")) {
                symTable[i][6] = "UNDEF";
            }
            position = tempName + sht_strtab_offset;
            StringBuilder tempOutName = new StringBuilder();
```

```
            int tempNum = 0;
            do {
                tempNum = tableOfElfFile[position];
                if (tableOfElfFile[position] != -1) {
                    tempOutName.append((char) tableOfElfFile[position++]);
                } else {
                    break;
                }
            } while (tempNum != 0);
            tempOutName.deleteCharAt(tempOutName.length() - 1);
            symTable[i][7] = tempOutName.toString();
            position = sh_offset + sh_entsize * (i + 1);
        }
        gettingNamesOfSymTab(); //Занесение имен и значений
    }

    private static boolean checkingInMapOfNameSymTab(String line) {
        for (Map.Entry<String, String> entry : id_of_names.entrySet()) {
            if (line.equals(entry.getKey())) {
                return true;
            }
        }
        return false;
    }

    private static void gettingNamesOfSymTab() {
        for (int i = symTable.length - 1; i > 0; i--) {
            if (symTable[i][7] != null) {
                StringBuilder tempValue = new StringBuilder(symTable[i][1]);
                while (tempValue.length() < 8) tempValue.insert(0, "0");
                id_of_names.put(tempValue.toString(), "<" + symTable[i][7] + ">");
            }
        }
    }

    private static int gettingByteLine(int countsByteReadingElement) {
        try {

            StringBuilder byteHex = new StringBuilder();
            for (int i = 0; i < countsByteReadingElement; i++) {
                String tempChar = Integer.toHexString(tableOfElfFile[position +
i]);
                byteHex.insert(0, tempChar);
                if (tempChar.length() == 1) {
                    byteHex.insert(0, "0");
                }
            }
            position += countsByteReadingElement;
            return Integer.parseInt(byteHex.toString(), 16);
        } catch (Exception e) {
            return -1;
        }

    }

    private static String getImmStringToInt(String immString) {
        if (immString.charAt(0) == '0') {
            return Integer.toString(Integer.parseInt(immString, 2));
        } else {
```

```
            int endRezult = 0;
            for (int i = immString.length() - 1; i > -1; i--) {
                if (immString.charAt(i) == '0') endRezult += Math.pow(2,
immString.length() - i - 1);
            }

            return Integer.toString(-endRezult - 1);
        }
    }

    private static void gettingInformationAboutSymTable() throws IOException {
        position = e_shoff;
        int sh_type = -1;
        do {
            position += 4;
            sh_type = gettingByteLine(4);
            position += 32;
        } while (sh_type != 2); //SHT_SYMTAB   ЗНАЧЕНИЕ - 2   Секция содержит
таблицу символов.
        position -= 32;
        position += 8;
        sh_offset = gettingByteLine(4);
        sh_size = gettingByteLine(4);
        position += 12;
        sh_entsize = gettingByteLine(4);

    }

    private static boolean thisIsElfFile() {
        while (position < tableOfElfFile.length) {
            StringBuilder checkingString = new StringBuilder();
            for (int i = 0; i < 4; i++) {
                if (position + i < tableOfElfFile.length)
checkingString.append((char) tableOfElfFile[position + i]);
            }
            if (checkingString.toString().equals("\u007FELF")) {
                position += 4;
                return true;
            }
        }

        return false;
    }

    private static int[] readElfFile(String name) throws IOException {
        try (FileInputStream fos = new FileInputStream(name)) {
            int i;
            List<Integer> tempArr = new ArrayList<>();
            while ((i = fos.read()) != -1) {
                tempArr.add(i);
            }
            int[] byteArr = new int[tempArr.size()];
            for (int j = 0; j < byteArr.length; j++) {
                byteArr[j] = tempArr.get(j);
            }
            return byteArr;
        } catch (FileNotFoundException e) {
            System.out.println("The file with this name was not found.");
        }
```

```
        return null;
    }

    private static void rDecoding(String code, int num) {  //
        int rd = Integer.parseInt(code.substring(20, 25), 2);
        int rs1 = Integer.parseInt(code.substring(12, 17), 2);
        int rs2 = Integer.parseInt(code.substring(7, 12), 2);
        assembler_commands[num][3] = "a" + rd;
        assembler_commands[num][4] = "a" + rs1;
        assembler_commands[num][5] = "a" + rs2;

    }

    private static void iDecoding(String code, int num) {
        int rd = Integer.parseInt(code.substring(20, 25), 2);
        int rs1 = Integer.parseInt(code.substring(12, 17), 2);
        String immString = code.substring(0, 12);
        assembler_commands[num][3] = "a" + rd;
        assembler_commands[num][4] = "a" + rs1;
        assembler_commands[num][5] = getImmStringToInt(immString);
    }

    private static void sDecoding(String code, int num) {
        int rs2 = Integer.parseInt(code.substring(7, 12), 2);
        int rs1 = Integer.parseInt(code.substring(12, 17), 2);
        String immString = code.substring(1, 7) + code.substring(20, 25);
        assembler_commands[num][3] = "a" + rs1;
        assembler_commands[num][4] = "a" + rs2;
        assembler_commands[num][5] = getImmStringToInt(immString);
    }

    private static void bDecoding(String code, int num) {
        int rs2 = Integer.parseInt(code.substring(7, 12), 2);
        int rs1 = Integer.parseInt(code.substring(12, 17), 2);
        String immString = Character.toString(code.charAt(0)) +
Character.toString(code.charAt(24)) + code.substring(1, 7) + code.substring(20,
24) + "0";
        assembler_commands[num][3] = "a" + rs1;
        assembler_commands[num][4] = "a" + rs2;
        assembler_commands[num][5] = getImmStringToInt(immString);
    }

    private static void uDecoding(String code, int num) {
        int rd = Integer.parseInt(code.substring(20, 25), 2);
        StringBuilder immStringBuilder = new StringBuilder(code.substring(0, 20));
        while (immStringBuilder.length() < 32) immStringBuilder.append("0");
        assembler_commands[num][3] = "a" + rd;
        assembler_commands[num][4] =
getImmStringToInt(immStringBuilder.toString());
    }

    private static void jDecoding(String code, int num) {
        int rd = Integer.parseInt(code.substring(20, 25), 2);
        String immString = code.substring(12, 20) +
Character.toString(code.charAt(11)) + code.substring(1, 7) + code.substring(7, 11)
+ "0";
        assembler_commands[num][3] = "a" + rd;
        assembler_commands[num][4] = getImmStringToInt(immString);
    }
```

```
}
```