САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA»

Выполнил(а): Бессонницын Евгений Сергеевич

студ. гр. М3138

Санкт-Петербург

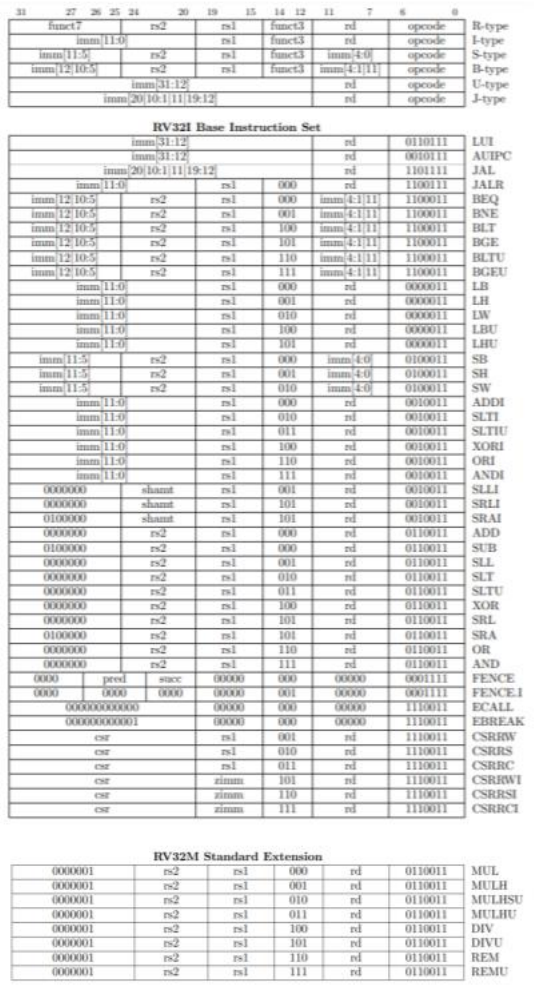
2020

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

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

Теоретическая часть

В системе кодирования RISC-V команды кодируются 32 битами, (подробное описание каждой представлено в таблице ниже, формат вывода каждой представлен на сайте https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#slti), каждая из них сначала идентифицируется значением opcode, потом по значению funct3 и в последнюю очередь по значению funct7.



31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

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
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

Рассмотрим саму структуру elf-файла:

Она состоит из Заголовка, таблицы заголовков секций (блоков), таблицы строк и таблицы символов.

Заголовок всегда находится в начале файла и содержит его общее описание и характеристики: тип, версия формата, архитектура процессора, размеры, смещения остальных частей файла. Он имеет размер 52 байта для 32-битных файлов, индексация байтов начинается с 0.

Байты 0 – 3 содержат сигнатуру файла.

Байт 4 содержит информацию о классе объектного файла

Байт 5 содержит информацию о методе кодирования данных(в нашем случае это little endian)

Байты 18-19 содержат информацию об архитектуре аппаратной платформы

Байты 32-35 содержат информацию о смещениях таблицы заголовков секций

Байты 48-49 содержат информацию о числе заголовков секций.

Байты 50-51 содержат информацию об индексе записи в таблице заголовков секций, описывающей таблицу названий секций

Далее идет таблица заголовков секций (блоков) содержит атрибуты секций файла, один такой заголовок имеет размер 40 байт (далее нумерация байтов идет относительно начала заголовка данного блока).

Байты 0-3 содержат информацию о смещении строки, содержащей название данной секции, относительно начала таблицы названий секций

Байты 4-7 содержат информацию о типе заголовка.

Байты 12-15 содержат информацию об адресе начиная с которого должна быть загружена секция (если должна).

Байты 16-19 содержат информацию о смещении секции от начала файла в байтах.

Байты 20-23 содержат информацию о размере секции

Таблица названий секций (таблица строк) содержит символы и элементы “\0”. Название каждой секции – это строка из символов таблицы названий секций начиная с индекса sh_name и заканчивая символом “\0”, не включая его

Таблица символов объектного файла содержит информацию, необходимую для поиска и перемещения символьных определений и ссылок программы.

Байты 0-3 содержат информацию о смещении строки, которая содержит название символа(относительно начала таблицы)

Байты 4-7 содержат информацию о непосредственно значении символа.

Байт 13 содержит информацию о типе символа и атрибутах привязки

Практическая часть

Написать программу, которая будет находить и дизассемблировать секцию кода (.text).

Класс Disassemble:

В начале определены методы, которые обеспечивают возможность различного вывода команд:

```
private static void printVariant0(String a, String b, String c, PrintStream out) {
    out.printf("%s\t%s, %s\n", a, b, c);
}

private static void printVariant1(String a, String b, String c, String d,
PrintStream out) {
    out.printf("%s\t%s, %s, %s\n", a, b, c, d);
}

private static void printVariant2(String a, String b, String c, String d,
PrintStream out) {
    out.printf("%s\t%s, %s(%s)\n", a, b, c, d);
}

private static void printVariant3(String a, PrintStream out) {
    out.printf("%s\n", a);
}
```

Метод register выводит название регистра:

```
private static String register(int x) {
    if (x == 0) {
        return "zero";
    }
    if (x == 1) {
        return "ra";
    }
    if (x == 2) {
        return "sp";
    }
    if (x == 3) {
        return "gp";
    }
    if (x == 4) {
        return "tp";
    }
    if (x == 5) {
        return "t0";
    }
    if (6 <= x && x <= 7) {
        return "t" + (x - 5);
    }
    if (x == 8) {
        return "s0";
    }
    if (x == 9) {
        return "s1";
    }
    if (10 <= x && x <= 11) {
        return "a" + (x - 10);
    }
    if (12 <= x && x <= 17) {
        return "a" + (x - 10);
    }
    if (18 <= x && x <= 27) {
        return "s" + (x - 16);
    }
    if (28 <= x && x <= 31) {
```

```
        return "t" + (x - 25);
    }
    return null;
}
```

Следующие два метода преобразуют целое число в строку:

```
private static String str(int x) {
    return Integer.toString(x);
}

private static String unsigned_str(int x) {
    return Integer.toUnsignedString(x);
}
```

Последний метод printDisassemble выводит закодированную команду (intstr). В случае возникновения ошибки – метод вернет “unknown instruction”(код приведен в разделе Листинг)

Класс elfReader:

Методы read возвращают числа записанные в байтах с указанными индексами, те в случае 2го метода вернется число записанное в байтах с номерами от ind до ind + size – 1;

```
public char read(int ind) throws IOException {
    move(ind);
    position++;
    return (char) reader.read();
}

int read(int ind, int size) throws IOException {
    int res = 0;
    int shift = 0;
    move(ind);
    for (int i = 0; i < size; i++) {
        res += (reader.read() << shift);
        shift += 8;
    }
    position += size;
    return res;
}
```

Метод move позволяет перейти от текущего байта к байту с номером ind, в случае, если он уже прочитан, ввод закроется и вновь откроется для чтения:

```
private void move(int ind) throws IOException {
    if (ind < position) {
        reader.close();
        reader = new BufferedInputStream(new FileInputStream(fileName));
        position = 0;
    }
    while (position < ind) {
        reader.read();
        position++;
    }
}
```

Основной класс hw4:

Методы printMem выводят адресса строчек и меток(если таковые есть):

```
private static void printMem(int mem, PrintStream out) {
    out.printf("%08x:\t", mem);
}
private static void printMem(int mem, String mark, PrintStream out) {
    out.printf("%08x: <%s>\t", mem, mark);
}
```

Метод title возвращает название секции по таблице строк:

```
private static String title(int name, char[] stringTable) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; ; i++) {
        if (stringTable[name + i] == '\0') {
            break;
        }
        sb.append(stringTable[name + i]);
    }
    return sb.toString();
}
```

Чтение входного файла, проверка аргументов и считывание полей заголовка файла:

```
PrintStream out = System.out;
try (elfReader in = new elfReader(args[0])) {
    if (args.length > 1) {
        out = new PrintStream(args[1]);
    }
```

```
    int e_shoff = in.read(32, 2);
    int e_shnum = in.read(48, 2);
    int e_shstrndx = in.read(50, 2);
```

Использование метода read(описанного выше), для чтения нужных байтов, и последующее чтение таблицы строк:

```
sh_offset = in.read(e_shoff + 40 * e_shstrndx + 16, 4);
sh_size = in.read(e_shoff + 40 * e_shstrndx + 20, 4);
char[] setOfStrings = new char[sh_size];
for (int i = 0; i < sh_size; i++) {
    setOfStrings[i] = in.read(sh_offset + i);
}
```

Чтение таблицы символов:

```
Map<Integer, String> marks = new HashMap<>();
for (int block = 0; block < e_shnum; block++) {
    sh_type = in.read(e_shoff + 40 * block + 4, 4);
    sh_offset = in.read(e_shoff + 40 * block + 16, 4);
    sh_size = in.read(e_shoff + 40 * block + 20, 4);

    if (sh_type == 2) {
        for (int symb = 0; symb < sh_size / 16; symb++) {
            st_name = in.read(sh_offset + 16 * symb, 4);
            st_value = in.read(sh_offset + 16 * symb + 4, 4);
            st_info = in.read(sh_offset + 16 * symb + 12, 1);

            if ((st_info & 0b0000_1111) == 2) {
                marks.put(st_value, title(st_name, setOfStrings));
            }
        }
    }
}
```

Определение нужного блока(секции) и последующий дизассемеблинг:

```
for (int section = 0; section < e_shnum; section++) {
    sh_name = in.read(e_shoff + 40 * section, 4);
    sh_addr = in.read(e_shoff + 40 * section + 12, 4);
    sh_offset = in.read(e_shoff + 40 * section + 16, 4);
    sh_size = in.read(e_shoff + 40 * section + 20, 4);
    if (title(sh_name, setOfStrings).equals(".text")) {
        for (int c = 0; c < sh_size; c+=4) {
            if (!marks.containsKey(sh_addr + c)) {
                printMem(sh_addr + c, out);
            } else {
                printMem(sh_addr + c, marks.get(sh_addr + c), out);
            }
            Disassemble.printDisassemble(in.read(sh_offset + c, 4), out);
        }
        break;
    }
}
```

Результат работы программы на файле test_elf:

```
00000000: <main>   addi   sp, sp, -32

00000004:    sw    ra, 28(sp)

00000008:    sw    s0, 24(sp)

0000000c:    addi  s0, sp, 32

00000010:    addi  a0, zero, 0

00000014:    sw    a0, 4084(s0)

00000018:    addi  a1, zero, 64

0000001c:    sw    a1, 4080(s0)

00000020:    sw    a0, 4076(s0)

00000024:    addi  a0, zero, 1

00000028:    sw    a0, 4072(s0)

0000002c:    jal   zero, 0

00000030:    lw    a0, 4072(s0)

00000034:    lw    a1, 4080(s0)

00000038:    bge   a0, a1, 0

0000003c:    jal   zero, 0

00000040:    lw    a0, 4072(s0)

00000044:    mul   a0, a0, a0

00000048:    lw    a1, 4076(s0)

0000004c:    add   a0, a1, a0

00000050:    sw    a0, 4076(s0)

00000054:    jal   zero, 0
```

```
00000058:    lw     a0, 4072(s0)

0000005c:    addi   a0, a0, 1

00000060:    sw     a0, 4072(s0)

00000064:    jal    zero, 0

00000068:    lw     a0, 4076(s0)

0000006c:    lw     s0, 24(sp)

00000070:    lw     ra, 28(sp)

00000074:    addi   sp, sp, 32

00000078:    jalr   zero, ra, 0
```

Листинг

Disassemble.java

```java
package evm;

import java.io.PrintStream;

public class Disassemble {
    private static void printVariant0(String a, String b, String c, PrintStream
out) {
        out.printf("%s\t%s, %s\n", a, b, c);
    }

    private static void printVariant1(String a, String b, String c, String d,
PrintStream out) {
        out.printf("%s\t%s, %s, %s\n", a, b, c, d);
    }

    private static void printVariant2(String a, String b, String c, String d,
PrintStream out) {
        out.printf("%s\t%s, %s(%s)\n", a, b, c, d);
    }

    private static void printVariant3(String a, PrintStream out) {
        out.printf("%s\n", a);
    }

    private static String register(int x) {
        if (x == 0) {
            return "zero";
        }
        if (x == 1) {
            return "ra";
        }
```

```
        if (x == 2) {
            return "sp";
        }
        if (x == 3) {
            return "gp";
        }
        if (x == 4) {
            return "tp";
        }
        if (x == 5) {
            return "t0";
        }
        if (6 <= x && x <= 7) {
            return "t" + (x - 5);
        }
        if (x == 8) {
            return "s0";
        }
        if (x == 9) {
            return "s1";
        }
        if (10 <= x && x <= 11) {
            return "a" + (x - 10);
        }
        if (12 <= x && x <= 17) {
            return "a" + (x - 10);
        }
        if (18 <= x && x <= 27) {
            return "s" + (x - 16);
        }
        if (28 <= x && x <= 31) {
            return "t" + (x - 25);
        }
        return null;
    }

    private static String str(int x) {
        return Integer.toString(x);
    }

    private static String unsigned_str(int x) {
        return Integer.toUnsignedString(x);
    }

    public static void printDisassemble(int instr, PrintStream out) {
        int funct7 = (instr & 0b1111111_00000_00000_000_00000_0000000) >>> 25;
        int rs2 = (instr & 0b0000000_11111_00000_000_00000_0000000) >>> 20;
        int rs1 = (instr & 0b0000000_00000_11111_000_00000_0000000) >>> 15;
        int funct3 = (instr & 0b0000000_00000_00000_111_00000_0000000) >>> 12;
        int rd = (instr & 0b0000000_00000_00000_000_11111_0000000) >>> 7;
        int opcode = (instr & 0b0000000_00000_00000_000_00000_1111111);
        int imm;
        int offset;
        int shamt;
        int pred;
        int succ;
        int zimm;
        if (opcode == 0b0110111) {
            printVariant0("lui", register(rd), unsigned_str((instr >>> 12) << 12),
```

```
out);
        } else if (opcode == 0b0010111) {
            printVariant0("auipc", register(rd), unsigned_str((instr >>> 12) <<
12), out);
        } else if (opcode == 0b1101111) {
            offset = 0;
            imm = instr >>> 12;
            offset |= (imm & 0b1_0000000000_0_00000000) << 1;
            offset |= (imm & 0b0_1111111111_0_00000000) >>> 8;
            offset |= (imm & 0b0_0000000000_1_00000000) << 3;
            offset |= (imm & 0b0_0000000000_0_11111111) << 12;
            offset -= (offset & (1 << 20)) << 1;
            printVariant0("jal", register(rd), str(offset), out);
        } else if (opcode == 0b1100111) {
            printVariant1("jalr", register(rd), register(rs1), str(instr >>> 20),
out);
        } else if (opcode == 0b1100011) {
            offset = 0;
            imm = funct7;
            offset |= (imm & 0b1_000000) << 6;
            offset |= (imm & 0b0_111111) << 5;
            imm = rd;
            offset |= (imm & 0b1111_0);
            offset |= (imm & 0b0000_1) << 11;
            if (funct3 == 0b000) {
                printVariant1("beq", register(rs1), register(rs2), str(offset),
out);
            } else if (funct3 == 0b001) {
                printVariant1("bne", register(rs1), register(rs2), str(offset),
out);
            } else if (funct3 == 0b100) {
                printVariant1("blt", register(rs1), register(rs2), str(offset),
out);
            } else if (funct3 == 0b101) {
                printVariant1("bge", register(rs1), register(rs2), str(offset),
out);
            } else if (funct3 == 0b110) {
                printVariant1("bltu", register(rs1), register(rs2), str(offset),
out);
            } else if (funct3 == 0b111) {
                printVariant1("bgeu", register(rs1), register(rs2), str(offset),
out);
            } else {
                out.println("unknown instruction");
            }
        } else if (opcode == 0b0000011) {
            offset = instr >>> 20;
            if (funct3 == 0b000) {
                printVariant2("lb", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b001) {
                printVariant2("lh", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b010) {
                printVariant2("lw", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b100) {
                printVariant2("lbu", register(rd), str(offset), register(rs1),
out);
```

```
            } else if (funct3 == 0b101) {
                printVariant2("lhu", register(rd), str(offset), register(rs1),
out);
            } else {
                out.println("unknown instruction");
            }
        } else if (opcode == 0b0100011) {
            offset = 0;
            imm = funct7;
            offset |= imm << 5;
            imm = rd;
            offset |= imm;
            if (funct3 == 0b000) {
                printVariant2("sb", register(rs2), str(offset), register(rs1),
out);
            } else if (funct3 == 0b001) {
                printVariant2("sh", register(rs2), str(offset), register(rs1),
out);
            } else if (funct3 == 0b010) {
                printVariant2("sw", register(rs2), str(offset), register(rs1),
out);
            } else {
                out.println("unknown instruction");
            }
        } else if (opcode == 0b0010011) {
            imm = instr >>> 20;
            imm -= (imm & 0b1000_00000000) << 1;
            shamt = rs2;
            if (funct3 == 0b000) {
                printVariant1("addi", register(rd), register(rs1), str(imm), out);
            } else if (funct3 == 0b010) {
                printVariant1("slti", register(rd), register(rs1), str(imm), out);
            } else if (funct3 == 0b011) {
                printVariant1("sltiu", register(rd), register(rs1), str(imm >= 0 ?
imm : imm + (1 << 12)), out);
            } else if (funct3 == 0b100) {
                printVariant1("xori", register(rd), register(rs1), str(imm), out);
            } else if (funct3 == 0b110) {
                printVariant1("ori", register(rd), register(rs1), str(imm), out);
            } else if (funct3 == 0b111) {
                printVariant1("andi", register(rd), register(rs1), str(imm), out);
            } else if (funct3 == 0b001) {
                printVariant1("slli", register(rd), register(rs1), str(shamt),
out);
            } else {
                if (funct7 == 0b0000000) {
                    printVariant1("srli", register(rd), register(rs1), str(shamt),
out);
                } else if (funct7 == 0b0100000) {
                    printVariant1("srai", register(rd), register(rs1), str(shamt),
out);
                } else {
                    out.println("unknown instruction");
                }
            }
        } else if (opcode == 0b0110011) {
            if (funct3 == 0b000) {
                if (funct7 == 0b0000000) {
                    printVariant1("add", register(rd), register(rs1),
```

```
register(rs2), out);
                } else if (funct7 == 0b0100000) {
                    printVariant1("sub", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("mul", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b001) {
                if (funct7 == 0b0000000) {
                    printVariant1("sll", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("mulh", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b010) {
                if (funct7 == 0b0000000) {
                    printVariant1("slt", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("mulhsu", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b011) {
                if (funct7 == 0b0000000) {
                    printVariant1("sltu", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("mulhu", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b100) {
                if (funct7 == 0b0000000) {
                    printVariant1("xor", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("div", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b101) {
                if (funct7 == 0b0000000) {
                    printVariant1("srl", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0100000) {
                    printVariant1("sra", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("divu", register(rd), register(rs1),
```

```
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b110) {
                if (funct7 == 0b0000000) {
                    printVariant1("or", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("rem", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            } else {
                if (funct7 == 0b0000000) {
                    printVariant1("and", register(rd), register(rs1),
register(rs2), out);
                } else if (funct7 == 0b0000001) {
                    printVariant1("remu", register(rd), register(rs1),
register(rs2), out);
                } else {
                    out.println("unknown instruction");
                }
            }
        } else if (opcode == 0b0001111) {
            pred = (instr & 0b00001111_00000000_00000000_00000000) >>> 24;
            succ = (instr & 0b00000000_11110000_00000000_00000000) >>> 20;
            if (funct3 == 0b000) {
                printVariant0("fence", str(pred), str(succ), out);
            } else if (funct3 == 0b001) {
                printVariant3("fence.i", out);
            } else {
                out.println("unknown instruction");
            }
        } else if (opcode == 0b1110011) {
            funct7 = instr >>> 20;
            offset = instr >>> 20;
            zimm = rs1;
            if (funct3 == 0b000) {
                if (funct7 == 0b000000000000) {
                    printVariant3("ecall", out);
                } else if (funct7 == 0b000000000001) {
                    printVariant3("ebreak", out);
                } else {
                    out.println("unknown instruction");
                }
            } else if (funct3 == 0b001) {
                printVariant1("csrrw", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b010) {
                printVariant1("csrrs", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b011) {
                printVariant1("csrrc", register(rd), str(offset), register(rs1),
out);
            } else if (funct3 == 0b101) {
                printVariant1("csrrwi", register(rd), str(offset), str(zimm),
out);
```

```
            } else if (funct3 == 0b110) {
                printVariant1("csrrsi", register(rd), str(offset), str(zimm),
out);
            } else if (funct3 == 0b111) {
                printVariant1("csrrci", register(rd), str(offset), str(zimm),
out);
            } else {
                out.println("unknown instruction");
            }
        } else {
            out.println("unknown instruction");
        }
    }
}
```

elfReader.java

```
package evm;

import java.io.*;

public class elfReader implements AutoCloseable {
    private int position;                // индекс следующего байта для чтения
    private InputStream reader;          // ввод данных
    private final String fileName;       //название входного файла

    public elfReader(String fileName) throws IOException {
        this.fileName = fileName;
        this.reader = new BufferedInputStream(new FileInputStream(fileName));
        this.position = 0;
        if (read(4) != 1) {
            throw new IOException("Incorrect file: it is not 32-bit object file");
        }
        if (read(5) != 1) {
            throw new IOException("Incorrect file: method for encoding data is not
Little Endian");
        }
        if (read(18, 2) != 0xF3) {
            throw new IOException("Incorrect file: the architecture of the
hardware platform is not RISC-V");
        }
    }

    public char read(int ind) throws IOException {
        move(ind);
        position++;
        return (char) reader.read();
    }

    int read(int ind, int size) throws IOException {
        int res = 0;
        int shift = 0;
        move(ind);
        for (int i = 0; i < size; i++) {
            res += (reader.read() << shift);
            shift += 8;
        }
        position += size;
```

```
        return res;
    }

    public void close() throws IOException {
        reader.close();
    }

    private void move(int ind) throws IOException {
        if (ind < position) {
            reader.close();
            reader = new BufferedInputStream(new FileInputStream(fileName));
            position = 0;
        }
        while (position < ind) {
            reader.read();
            position++;
        }
    }
}
```

hw4.java

```
package evm;

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

public class hw4 {
    private static void printMem(int mem, PrintStream out) {
        out.printf("%08x:\t", mem);
    }
    private static void printMem(int mem, String mark, PrintStream out) {
        out.printf("%08x: <%s>\t", mem, mark);
    }
    private static String title(int name, char[] stringTable) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; ; i++) {
            if (stringTable[name + i] == '\0') {
                break;
            }
            sb.append(stringTable[name + i]);
        }
        return sb.toString();
    }
    public static void main(String[] args) {
        PrintStream out = System.out;
        try (elfReader in = new elfReader(args[0])) {
            if (args.length > 1) {
                out = new PrintStream(args[1]);
            }
            int e_shoff = in.read(32, 2);
            int e_shnum = in.read(48, 2);
            int e_shstrndx = in.read(50, 2);
            int sh_name;
            int sh_type;
```

```
            int sh_addr;
            int sh_offset;
            int sh_size;
            int st_name;
            int st_value;
            int st_info;
            sh_offset = in.read(e_shoff + 40 * e_shstrndx + 16, 4);
            sh_size = in.read(e_shoff + 40 * e_shstrndx + 20, 4);
            char[] setOfStrings = new char[sh_size];
            for (int i = 0; i < sh_size; i++) {
                setOfStrings[i] = in.read(sh_offset + i);
            }
            Map<Integer, String> marks = new HashMap<>();
            for (int block = 0; block < e_shnum; block++) {
                sh_type = in.read(e_shoff + 40 * block + 4, 4);
                sh_offset = in.read(e_shoff + 40 * block + 16, 4);
                sh_size = in.read(e_shoff + 40 * block + 20, 4);

                if (sh_type == 2) {
                    for (int symb = 0; symb < sh_size / 16; symb++) {
                        st_name = in.read(sh_offset + 16 * symb, 4);
                        st_value = in.read(sh_offset + 16 * symb + 4, 4);
                        st_info = in.read(sh_offset + 16 * symb + 12, 1);

                        if ((st_info & 0b0000_1111) == 2) {
                            marks.put(st_value, title(st_name, setOfStrings));
                        }
                    }
                }
            }
            for (int section = 0; section < e_shnum; section++) {
                sh_name = in.read(e_shoff + 40 * section, 4);
                sh_addr = in.read(e_shoff + 40 * section + 12, 4);
                sh_offset = in.read(e_shoff + 40 * section + 16, 4);
                sh_size = in.read(e_shoff + 40 * section + 20, 4);
                if (title(sh_name, setOfStrings).equals(".text")) {
                    for (int c = 0; c < sh_size; c+=4) {
                        if (!marks.containsKey(sh_addr + c)) {
                            printMem(sh_addr + c, out);
                        } else {
                            printMem(sh_addr + c, marks.get(sh_addr + c), out);
                        }
                        Disassemble.printDisassemble(in.read(sh_offset + c, 4),
out);
                    }
                    break;
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found");
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } finally {
            out.close();
        }
    }
}
```