Лабораторная работа №3	38	2022
ISA	Ситкина Алена Николаевна	

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

Инструментарий и требования к работе: работа выполнена на Java, JDK 17.

Описание системы кодирования команд RISC-V.

Примечание: далее в тексте отчёта будут даваться ссылки на источники, откуда была взята информация. При этом та часть информации, которая была нами использована, переведена на русский язык и содержится в отчёте, поэтому переход по этим ссылкам не является необходимым.

ISA – это часть архитектуры компьютера, определяющая программируемую часть ядра процессора. RISC-V – это открытое ISA. Точнее говоря, RISC-V является семейством родственных ISA, из которых на данный момент существует 4 базовых ISA. ISA определяется базовым набором целочисленных ISA («base integer ISA» в документации RISC-V) с дополнительными расширениями (каждый базовый набор целочисленных команд (base integer instruction set) характеризуется размером целочисленных регистров (и соответствующим размером адресного пространства) и количеством этих регистров) (источник 5, страница 4). RISC-V – это ISA типа Reg-Reg (регистр-регистр), что означает, что доступ к памяти имеют только инструкции работы с памятью (группы команд типов load и store), остальные же работают только с регистрами (источник 5, страница 24).

Далее мы будем рассматривать наборы команд (иначе говоря, инструкций) RV32I и RV32M.

RV32I – это base instruction set. Он использует 32 целочисленных регистра + 1 дополнительный, содержащий адрес текущей инструкции. Для вывода регистров нами использовался формат, описанный на официальном github RISC-V (источник 6). RV32I описывает команды, работающие с целыми числами, команды условных и безусловных переходов, команды работы с памятью, а также некоторые другие. RV32M – это расширение, добавляющее команды умножения и целочисленного деления (само целочисленное деление и взятие остатка при делении) целых чисел. Каждая команда использует один из 6 форматов (R, I, S, U, B, J), каждый из которых

состоит из 32 бит и кодируется как 4-байтовая связка (4 байта, идущих подряд) (см рисунок 1, источник 5, страница 16).

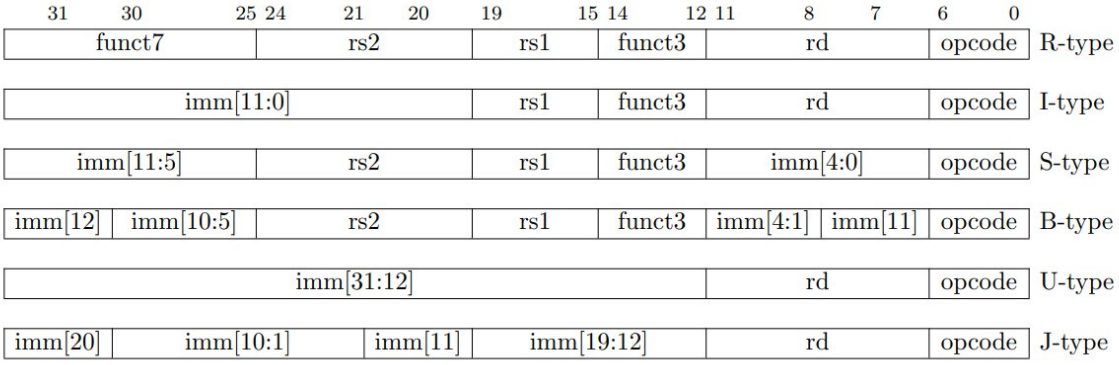

Рисунок 1 – Форматы кодирования инструкций RV32I и RV32M

imm (immediate) – это числовая константа, являющаяся одним из аргументов инструкции, rd – регистр, куда запишется результат работы команды, rs1 и rs2 – входные регистры. Поля func7, func3 и opcode используются для декодирования команды. Соответствие значений этих полей и команд можно увидеть на рисунках 3 и 4 (источник 5, страницы 130-131).

Как видно из рисунка 1, в разных форматах числовые константы кодируются по-разному. Декодирование числовых констант происходит согласно схеме на рисунке 2 (источник 5, страница 17). Константы для всех команд знаковые (даже если команда использует беззнаковое сравнение), знаковый бит – это всегда 31-ый бит в коде команды.

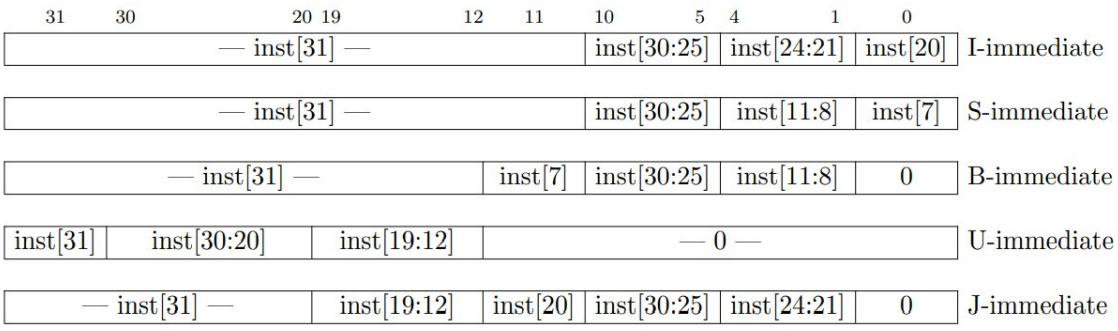

Рисунок 2 – Декодирование числовых констант (inst[x] – номер бита в коде команды)

Опишем, как интерпретируется значение immediate для различных типов команд. Для арифметических операций числовая константа (при её наличии) является одним из аргументов. Для команд битовых сдвигов

(SLLI, SRLI, SRAI) shamt – это величина сдвига. Команды load(LB-LHU на рисунке 3)/store(SB-SW) типа получают адрес в памяти, к которому нужно обратиться, добавлением immediate к значению rs1. Команды условного (BEQ-BGEU) и безусловного (JAL и JALR) перехода, использующие B-формат, получают адрес, на который нужно перейти, добавляя значение immediate к адресу текущей команды. LUI (U-формат) используется для построения 32-разрядных констант. LUI помещает значение immediate в старшие 20 битов регистра rd, заполняя младшие 12 бит нулями. AUIPC (U-формат) используется для построения относительных адресов. AUIPC формирует 32-разрядное смещение из 20-разрядного immediate, заполняя младшие 12 битов нулями, добавляет это смещение к адресу текущей инструкции, затем помещает результат в регистр rd. Для LUI и AUIPC наша программа выводит только саму числовую константу, без младших нулей, в формате hex. Был использован источник 5, страницы 13-28. Для команд условного перехода и команды JAL выводится адрес, на который осуществляется переход, в формате hex. Остальные числовые константы выводятся в dec формате.

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 3 – Кодирование команд RV32I

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

Рисунок 4 – Кодирование команд RV32M

Описание структуры файла ELF.

ELF файл – это файловый формат для объектных и исполняемых файлов (а так же некоторых других типов файлов). Опишем структурные компоненты elf файла, которые необходимо было проанализировать в ходе данной лабораторной работы.

ELF файл состоит из заголовков и секций. Нам понадобятся 4 секции: .text, .symtab, .strtab (содержит строковые названия символов из .symtab) и .shstrtab (содержит строковые названия секций). Чтобы найти местоположение в файле нужных нам секций, нужно проанализировать заголовки.

Любой elf-файл начинается с elf header. Он имеет следующую структуру (чтобы увидеть полную таблицу, см источник 1, ниже представлена её сокращенная версия):

Offset (смещение относительно начала файла)	Размер поля (в байтах)	Значение поля
0x00	1	0x7F
0x01	1	0x45 (E in ASCII)
0x02	1	0x4c (L in ASCII)
0x03	1	0x46 (F in ASCII)
0x04	1	Формат кодирования, 1 для 32-битного, 2 для 64-битного
0x05	1	little(1)/big(2) endian
0x06	1	1 для оригинальной и текущей версии ELF
0x07	1	ABI
0x08	1	Версия ABI
0x09	7	Не используются; должны быть заполнены нулями
0x10	2	Тип файла
0x12	2	ISA
0x14	4	1 для оригинальной версии ELF
0x18	4	Входная точка процесса
0x1C	4	Указатель (здесь и далее, говоря “указатель” мы имеем в виду смещение от начала файла (если явно не указано иное) на начало program header
0x20	4	Указатель на начало section header table
0x24	4	Вспомогательная информация
0x28	2	Размер elf header

0x2A	2	Размер program header table entry (в байтах)
0x2C	2	Число элементов в program header table
0x2E	2	Размер section header table entry (в байтах)
0x30	2	Число элементов в section header table
0x32	2	Указатель на начало section header table
0x34	-	конец

Таблица 1 – структура elf header

Далее рассмотрим section header table. Это таблица, позволяющая узнать информацию о секциях в нашем elf файле. Она состоит из section header’ов, каждый из которых соответствует одной из секций файла. Section header имеет следующую структуру (см источник 2):

Offset (смещение относительно начала section header’a)	Размер поля (в байтах)	Значение поля
0x00	4	Offset в .shstrtab, по которому лежит имя данной секции
0x04	4	Тип соответствующей секции
0x08	4	Описание некоторых свойств секции
0x0C	4	Виртуальный адрес секции в памяти (если таковой имеется)
0x10	4	Указатель на начало секции в файле
0x14	4	Размер секции (в байтах)
0x18	4	Индекс секции
0x1C	4	Дополнительная информация о секции (нами она не используется)
0x20	4	Ещё некоторая информация о секции (нами она не используется)
0x24	4	Если секция содержит экземпляры фиксированного размера, то размер одного такого экземпляра (иначе 0)
0x28	-	конец

Таблица 2 – структура section header

Рассмотрим .strtab и .shstrtab. Обе эти секции – это секции типа string table. Такие секции (их может быть несколько в одном файле) хранят строки. .shstrtab хранит названия секций файла. Строки в этих таблицах заканчиваются нулевым символом (‘\0’), первый байт также хранит null character. Информация взята из источника 3.

Symbol table (.symtab) – таблица символов (в файле может существовать не более одной секции такого типа). Она содержит информацию о символьных определениях и ссылках в файле (например, здесь хранится информация о метках). Один элемент этой таблицы имеет следующую структуру (см источник 4):

Размер поля (в байтах)	Хранимое значение
4	name (offset в .strtab)
4	value
4	size
1	info (type and binding attributes)
1	other (symbol’s visibility)
2	Индекс (в section header table) соответствующей секции

Таблица 3 – структура symbol table entry

Соответствия между значениями полей и их байтовой записью выстроены согласно источнику 4 (рисунки 5-8).

Table 12-19 ELF Symbol Types, ELF32_ST_TYPE and ELF64_ST_TYPE

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_COMMON	5
STT_TLS	6
STT_LOOS	10
STT_HIOS	12
STT_LOPROC	13
STT_SPARC_REGISTER	13
STT_HIPROC	15

Рисунок 5 – symbol type (здесь и далее для повторяющихся value выбрано одно произвольное фиксированное значение поля)

Table 12-18 ELF Symbol Binding, ELF32_ST_BIND and ELF64_ST_BIND

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOOS	10
STB_HIOS	12
STB_LOPROC	13
STB_HIPROC	15

Рисунок 6 – symbol binding

Table 12-20 ELF Symbol Visibility

Name	Value
STV_DEFAULT	0
STV_INTERNAL	1
STV_HIDDEN	2
STV_PROTECTED	3
STV_EXPORTED	4
STV_SINGLETON	5
STV_ELIMINATE	6

Рисунок 7 – symbol visibility

Table 12-4 ELF Special Section Indexes

Name	Value
SHN_UNDEF	0
SHN_LORESERVE	0xff00
SHN_LOPROC	0xff00
SHN_BEFORE	0xff00
SHN_AFTER	0xff01
SHN_AMD64_LCOMMON	0xff02
SHN_HIPROC	0xff1f
SHN_LOOS	0xff20
SHN_LOSUNW	0xff3f
SHN_SUNW_IGNORE	0xff3f
SHN_HISUNW	0xff3f
SHN_HIOS	0xff3f
SHN_ABS	0xfff1
SHN_COMMON	0xfff2
SHN_XINDEX	0xffff
SHN_HIRESERVE	0xffff

Рисунок 8 – symbol index (для значений, не вошедших в эту таблицу, index равен самому значению 2-байтовой связки (как числа))

Секция .text – это секция, содержащая байты, кодирующие исполняемые инструкции (в случае исполняемых файлов). Именно эту информацию мы будем преобразовывать в текст программы на языке ассемблера. Информация о том, в каком именно виде закодированы инструкции (архитерктура, little/big endian и прочее), содержится в уже ранее описанном нами elf header’е.

Описание работы написанного кода.

Программа предполагает, что ей на вход подается два аргумента: имена входного и выходного файлов соответственно. Пример команды, компилирующей все файлы .java и запускающей программу (запускать необходимо файл Main):

```
javac *.java && java Main test_elf test_elf_res.txt
```

Листинг 1 – пример команды для запуска нашей программы

Класс Main (см листинг 2) принимает в качестве аргументов командной строки имена входного и выходного файлов, содержит обработку всех исключений, вылетающих из ElfParser и RISCVParser (в том числе и обработку нашего собственного исключения UnsupportedFileFormatException (см листинг 3), говорящего, что данный формат файла не поддерживается программой), побайтово читает весь файл и сохраняет его в массив. Далее создается и вызывается экземпляр класса ElfParser.

```
public class Main {
   public static void main(InputStream input, OutputStream output) throws IOException,
UnsupportedFileFormatException {
      List<Integer> file = new ArrayList<>();
      int new_byte = input.read();
      while (new_byte != -1) {
         file.add(new_byte);
         new_byte = input.read();
      }
      ElfParser parser = new ElfParser(file, output);
      parser.parse();
   }

   public static void main(String[] args) {
      try {
         InputStream input = new FileInputStream(args[0]);
         try {
            OutputStream output = new FileOutputStream(args[1]);
            try {
               main(input, output);
            } catch (UnsupportedFileFormatException e) {
               System.out.println("Unsupported file format: " + e.getMessage());
            } catch (IOException e) {
               System.out.println("I/O error: " + e.getMessage());
            }
         } catch (FileNotFoundException e) {
            System.out.println("Output file not found: " + e.getMessage());
         }
      } catch (FileNotFoundException e) {
         System.out.println("Input file not found: " + e.getMessage());
      } catch (Exception e) {
         System.out.println("Something went wrong, I give up: " + e.getMessage());
      }
   }
}
```

Листинг 2 – класс Main

```
public class UnsupportedFileFormatException extends Exception {
   public UnsupportedFileFormatException(String mess) {
      super(mess);
```

```
    }

    public UnsupportedFileFormatException() {
        super();
    }
}
```

Листинг 3 – класс UnsupportedFileFormatException

Также было создано два вспомогательных класса ElfSymbol и ElfSectionHeader (листинги 4 и 5). Они хранят данные, соответствующие полям описываемых структур, и не имеют методов.

```
public class ElfSymbol {
    public String name;
    public int value;
    public int size;
    public String type;
    public String bind;
    public String vis;
    public String index;
}
```

Листинг 4 – класс ElfSymbol

```
public class ElfSectionHeader {
    public String name;
    public int type;
    public int offset;
    public int size;
}
```

Листинг 5 – класс ElfSectionHeader

Класс OpCodes, содержащий только статические публичные поля, хранит информацию о том, как кодируются все команды (кроме ebreak и ecall, это было сделано для удобства реализации преобразования команд, см листинг 12), которые нам необходимо поддерживать (Map types позволяет получить тип кодирования по opcode, далее для каждого типа кодирования создан Map, позволяющий по opcode, func3 и func7 (при их наличии) получить название команды; также этот класс содержит названия регистров (registerNames)) (см листинг 6).

```
public class OpCodes {
    public static Map<Integer, Map<Integer, Map<Integer, String>>> Rcodes = Map.ofEntries(
            Map.entry(0b0010011, Map.of(
                    0b001, Map.of(0b0000000, "slli"),
                    0b101, Map.of(0b0000000, "srli", 0b0100000, "srai")
            )),
            Map.entry(0b0110011, Map.of(
                    0b000, Map.of(0b0000000, "add", 0b0100000, "sub", 0b0000001, "mul"),
                    0b001, Map.of(0b0000000, "sll", 0b0000001, "mulh"),
```

```
            0b010, Map.of(0b0000000, "slt", 0b0000001, "mulhsu"),
            0b011, Map.of(0b0000000, "sltu", 0b0000001, "mulhu"),
            0b100, Map.of(0b0000000, "xor", 0b0000001, "div"),
            0b101, Map.of(0b0000000, "srl", 0b0100000, "sra", 0b0000001, "divu"),
            0b110, Map.of(0b0000000, "or", 0b0000001, "rem"),
            0b111, Map.of(0b0000000, "and", 0b0000001, "remu")
        ))
);

public static Map<Integer, Map<Integer, String>> Icodes = Map.ofEntries(
        Map.entry(0b1100111, Map.of(0b000, "jalr")),
        Map.entry(0b0000011, Map.of(
            0b000, "lb",
            0b001, "lh",
            0b010, "lw",
            0b100, "lbu",
            0b101, "lhu"
        )),
        Map.entry(0b0010011, Map.of(
            0b000, "addi",
            0b010, "slti",
            0b011, "sltiu",
            0b100, "xori",
            0b110, "ori",
            0b111, "andi"
        ))
);

public static Map<Integer, Map<Integer, String>> Scodes = Map.ofEntries(
        Map.entry(0b0100011, Map.of(
            0b000, "sb",
            0b001, "sh",
            0b010, "sw"
        ))
);

public static Map<Integer, Map<Integer, String>> Bcodes = Map.ofEntries(
        Map.entry(0b1100011, Map.of(
            0b000, "beq",
            0b001, "bne",
            0b100, "blt",
            0b101, "bge",
            0b110, "bltu",
            0b111, "bgeu"
        ))
);

public static Map<Integer, String> Ucodes = Map.of(
        0b0110111, "lui",
        0b0010111, "auipc"
);
```

```
    public static Map<Integer, String> Jcodes = Map.of(
            0b1101111, "jal"
    );

    public static Map<Integer, Character> types = Map.ofEntries(
            Map.entry(0b0110111, 'U'),
            Map.entry(0b0010111, 'U'),
            Map.entry(0b1101111, 'J'),
            Map.entry(0b1100111, 'I'),
            Map.entry(0b1100011, 'B'),
            Map.entry(0b0000011, 'I'),
            Map.entry(0b0100011, 'S'),
            Map.entry(0b0010011, 'I'),
            Map.entry(0b0110011, 'R')
    );

    public static List<String> registerNames = List.of(
            "zero", "ra", "sp", "gp", "tp", "t0", "t1", "t2", "s0", "s1", "a0", "a1", "a2", "a3", "a4",
            "a5", "a6", "a7", "s2", "s3", "s4", "s5", "s6", "s7", "s8", "s9", "s10", "s11", "t3", "t4", "t5", "t6"
    );
}
```

Листинг 6 – класс OpCodes

Теперь перейдём к классам, непосредственно осуществляющим обработку elf файла (ElfParser) и обработку секции .text (RISCVParser).

Рассмотрим сначала класс ElfParser. Его метод parse (листинг 7) проверяет, является ли файл корректным (проверяет поля из elf header (таблица 1), если нет, то выбрасывает ошибку с соответствующим сообщением), затем по очереди обрабатывает section header table, symbol table и text.

```
public void parse() throws UnsupportedFileFormatException, IOException {
    if (!(elf[0] == 0x7f &&
            elf[1] == 0x45 &&
            elf[2] == 0x4c &&
            elf[3] == 0x46)) {
        throw new UnsupportedFileFormatException("File is not ELF");
    }
    if (!(elf[4] == 1)) {
        throw new UnsupportedFileFormatException("File is not 32-bit");
    }
    if (!(elf[18] == 0xf3 && elf[19] == 0x00)) {
        throw new UnsupportedFileFormatException("File is not RISC-V");
    }
    if (!(elf[5] == 1)) {
        throw new UnsupportedFileFormatException("File is not little endian");
    }
    parseSectionHeaderTable(); // we need .text .symtab .strtab sections
```

```
    parseSymbolTable();
    parseText();
    out.write(symtab.getBytes());
}
```

Листинг 7 – метод parse класса ElfParser

Метод parseSectionHeaderTable (листинг 8) итерируется по section header table, из каждого section header’а выделяет описанные выше поля (таблица 2) и проверяет (по полям, содержащим тип и имя секции), не является ли эта секция одной из нужных нам (symtab, text или strtab) (если да, то сохраним необходимую информацию в соответствующие поля).

```
private void parseSectionHeaderTable() {
    int sectionTablePosition = get4Bytes(32);
    int sectionTableSize = get2Bytes(48);
    shStringTableStart = get4Bytes(sectionTablePosition + get2Bytes(50) * SECTION_HEADER_SIZE +
16);
    for (int i = sectionTablePosition; i < sectionTablePosition + sectionTableSize *
SECTION_HEADER_SIZE; i += SECTION_HEADER_SIZE) {
        ElfSectionHeader header = new ElfSectionHeader();
        header.name = getSectionName(get4Bytes(i));
        header.type = get4Bytes(i + 4);
        header.offset = get4Bytes(i + 16);
        header.size = get4Bytes(i + 20);
        if (header.type == 0x2) {
            symbolTableStart = header.offset;
            symbolTableSize = header.size;
        }
        if (header.name.equals(".text")) {
            textStart = header.offset;
            textSize = header.size;
            textAddr = get4Bytes(i + 12);
        } else if (header.name.equals(".strtab")) {
            stringTableStart = header.offset;
            stringTableSize = header.size;
        }
    }
}
```

Листинг 8 – метод parseSectionHeaderTable класса ElfParser

Метод parseSymbolTable (листинг 9) записывает в поле symtab строку, являющуюся результатом парсинга symbol table. В методе происходит итерация по элементам symbol table. Для каждого элемента выделяются описанные выше поля (для получения числа из нескольких байтов используются методы get2Bytes и get4Bytes) (под каждое поле написан соответствующий метод, представляющий из себя один switch-case блок, соответствия между значениями полей и их байтовой записью в файле выстроены согласно таблице 3 и источнику 4 (см рисунки 5-8)). Элементы

с типом FUNC записываются в Map labels, так как они являются метками и будут использованы при парсинге секции .text.

```
private void parseSymbolTable() throws IOException {
    StringBuilder resToWrite = new StringBuilder();
    resToWrite.append(SYMBOL_TABLE_OUTPUT_HEADER);
    int idx = 0;
    for (int i = symbolTableStart; i < symbolTableStart + symbolTableSize; i += SYMBOL_SIZE) {
        ElfSymbol symbol = new ElfSymbol();
        symbol.name = getSymbolName(get4Bytes(i));
        symbol.value = get4Bytes(i + 4);
        symbol.size = get4Bytes(i + 8);
        int info = elf[i + 12];
        symbol.type = getSymbolType((info) & 0xf);
        symbol.bind = getSymbolBind((info) >> 4);
        symbol.vis = getSymbolVis(elf[i + 13]);
        symbol.index = getSymbolIndex(get2Bytes(i + 14));
        if (Objects.equals(symbol.type, "FUNC")) {
            labels.put(symbol.value, symbol.name);
        }
        resToWrite.append(String.format(SYMBOL_TABLE_LINE_TEMPLATE, idx, symbol.value,
symbol.size, symbol.type,
                symbol.bind, symbol.vis, symbol.index, symbol.name));
        idx++;
    }
    symtab = resToWrite.toString();
}
```

Листинг 9 – метод parseSymbolTable класса ElfParser

Метод parseText создает экземпляр класса RISCVParser и вызывает у него метод parse.

В методе parseText класса RISCVParser (листинг 10) происходит итерация по командам (размер кода каждой - 4 байта). Сначала каждая команда декодируется (метод parseLine, листинг 12), результат записывается в массив. После для каждой сначала проверяется наличие метки (в том числе и L-метки) (метод checkForLabel, листинг 11), затем выводится сама команда (уже декодированная). В методе parseLine сначала делается проверка на ebreak и ecall, затем на unknown_instruction (неподдерживаемые команды). Далее с помощью класса OpCode мы получаем формат кодирования команды, и в соответствии с ним декодируем её. Заметим, что если необходимо получить метку какого-либо адреса, мы сначала проверяем, есть ли у этого адреса метка в symtable (мы ранее договорились считать метками в symtable только элементы symtable с типом FUNC) или, возможно, мы уже ранее установили метку для этого адреса, и если нет, то используем новую L-метку (обозначение: L%i) с указанием адреса и увеличим счётчик меток (метод getLabel, листинг 13).

Инструкции печатаются в формате «адрес в памяти: полный код инструкции инструкция аргументы». Для декодирования 4 байта переводятся в массив бит (get4Bytes возвращает число, состоящее из 4 подряд идущих байтов, convertToBits возвращает массив бит, getBits возвращает число, полученное из двоичной записи на указанном подотрезке переданного массива, листинг 14).

```
public void parseText() throws IOException {
    List<String> commands = new ArrayList<>();
    for (int i = textStart; i < textStart + textSize; i += 4) {
        commands.add(parseLine(i));
    }
    for (int i = textStart; i < textStart + textSize; i += 4) {
        checkForLabel(i);
        out.write(commands.get((i - textStart) / 4).getBytes());
    }
}
```

Листинг 10 – метод parseText класса RISCVParser

```
private void checkForLabel(int idx) throws IOException {
    if (labels.containsKey(idx + textAddr)) {
        out.write(String.format(LabelStrTemplate, idx + textAddr, labels.get(idx +
textAddr)).getBytes());
    }
}
```

Листинг 11 – метод checkForLabel класса RISCVParser

```
private String parseLine(int idx) {
    StringBuilder out = new StringBuilder();
    boolean[] code = convertToBits(get4Bytes(idx));
    int opcode = getBits(code, 0, 7);
    if (get4Bytes(idx) == 0b00000000000000000000000001110011) {
        out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"ecall"));
        return out.toString();
    } else if (get4Bytes(idx) == 0b00000000000100000000000001110011) {
        out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"ebreak"));
        return out.toString();
    }
    if (!OpCodes.types.containsKey(opcode)) {
        out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"unknown_instruction"));
        return out.toString();
    }
    char type = OpCodes.types.get(opcode);
    int immSign = code[31] ? (-1) : 1;
    if (type == 'R') {
        String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
        int funct3 = getBits(code, 12, 15);
```

```
        String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
        int funct7 = getBits(code, 25, 32);
        String command = OpCodes.Rcodes.get(opcode).get(funct3).get(funct7);
        if (opcode == 0b0010011) {
            // shift
            int shamt = getBits(code, 20, 25);
            out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, rs1, shamt));
            return out.toString();
        }
        String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
        out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
command, rd, rs1, rs2));
        return out.toString();
    }
    if (type == 'I') {
        String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
        int funct3 = getBits(code, 12, 15);
        String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
        int imm = reverseImm(getBits(code,20, 31), immSign, 11);
        String command = OpCodes.Icodes.get(opcode).get(funct3);
        if (opcode == 0b0000011 || opcode == 0b1100111) {
            // load command or jalr
            out.append(String.format(LoadStoreInstructionTemplate, idx + textAddr, get4Bytes(idx),
command, rd, imm, rs1));
        } else {
            out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, rs1, imm));
        }
        return out.toString();
    }
    if (type == 'S') {
        int funct3 = getBits(code, 12, 15);
        String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
        String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
        int imm = reverseImm(getBits(code, 7, 12) + (getBits(code, 25, 31) << 5), immSign, 11);
        String command = OpCodes.Scodes.get(opcode).get(funct3);
        if (opcode == 0b0100011) {
            // store command
            out.append(String.format(LoadStoreInstructionTemplate, idx + textAddr, get4Bytes(idx),
command, rs2, imm, rs1));
        } else {
            out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rs1, rs2, imm));
        }
        return out.toString();
    }
    if (type == 'B') {
        int funct3 = getBits(code, 12, 15);
        String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
        String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
```

```
        int imm = reverseImm((getBits(code, 8, 12) << 1) +
                (getBits(code, 25, 31) << 5) +
                (getBits(code, 7, 8) << 11), immSign, 12);
        String command = OpCodes.Bcodes.get(opcode).get(funct3);
        out.append(String.format(ThreeArgumentBranchInstructionTemplate,
                idx + textAddr, get4Bytes(idx), command, rs1, rs2, idx + textAddr + imm, getLabel(idx +
textAddr + imm)));
        return out.toString();
    }
    if (type == 'U') {
        String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
        int imm = reverseImm(getBits(code, 12, 31), immSign, 31);
        String command = OpCodes.Ucodes.get(opcode);
        out.append(String.format(TwoArgumentUtypeInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, imm));
        return out.toString();
    }
    if (type == 'J') {
        // jal
        String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
        int imm = reverseImm((getBits(code, 21, 25) << 1) + (getBits(code, 25, 31) << 5) +
                (getBits(code, 20, 21) << 11 )+ (getBits(code, 12, 20) << 12), immSign, 20);
        String command = OpCodes.Jcodes.get(opcode);
        out.append(String.format(TwoArgumentJalInstructionTemplate, idx + textAddr, get4Bytes(idx),
command, rd,
                idx + textAddr + imm, getLabel(idx + textAddr + imm)));
    }
    return out.toString();
}
```

Листинг 12 – метод parseLine класса RISCVParser

```
private String getLabel(int value) {
    if (labels.containsKey(value)) {
        return labels.get(value);
    }
    labelNum++;
    return "L" + (labelNum - 1);
}
```

Листинг 13 – метод getLabel класса RISCVParser

```
private int get4Bytes(int idx) {
    return elf[idx + 3] * 256 * 256 * 256 + elf[idx + 2] * 256 * 256 + elf[idx + 1] * 256 + elf[idx];
}

private int getBits(boolean[] n, int start, int end) {
    StringBuilder binNum = new StringBuilder();
    for (int i = end - 1; i >= start; i--) {
        binNum.append(n[i] ? '1' : '0');
    }
    return Integer.parseInt(binNum.toString(), 2);
}
```

```
private boolean[] convertToBits(int x) {
    boolean[] res = new boolean[32];
    for (int i = 0; i < 32; i++) {
        res[i] = false;
    }
    String s = Integer.toBinaryString(x);
    for (int i = 32 - s.length(); i < 32; i++) {
        res[31 - i] = (s.charAt(i - 32 + s.length()) == '1');
    }
    return res;
}
```

Листинг 14 – методы get4Bytes, getBits, convertToBits класса RISCVParser

Результат работы написанной программы на приложенном к заданию файле (дизассемблер и таблица символов).

```
.text
00010074   <main>:
   10074:  ff010113     addi    sp, sp, -16
   10078:  00112623       sw    ra, 12(sp)
   1007c:  030000ef      jal    ra, 0x100ac <mmul>
   10080:  00c12083       lw    ra, 12(sp)
   10084:  00000513     addi    a0, zero, 0
   10088:  01010113     addi    sp, sp, 16
   1008c:  00008067     jalr    zero, 0(ra)
   10090:  00000013     addi    zero, zero, 0
   10094:  00100137      lui    sp, 0x100
   10098:  fddff0ef      jal    ra, 0x10074 <main>
   1009c:  00050593     addi    a1, a0, 0
   100a0:  00a00893     addi    a7, zero, 10
   100a4:  0ff0000f   unknown_instruction
   100a8:  00000073     ecall
000100ac   <mmul>:
   100ac:  00011f37      lui    t5, 0x11
   100b0:  124f0513     addi    a0, t5, 292
   100b4:  65450513     addi    a0, a0, 1620
   100b8:  124f0f13     addi    t5, t5, 292
   100bc:  e4018293     addi    t0, gp, -448
   100c0:  fd018f93     addi    t6, gp, -48
   100c4:  02800e93     addi    t4, zero, 40
000100c8   <L2>:
   100c8:  fec50e13     addi    t3, a0, -20
   100cc:  000f0313     addi    t1, t5, 0
   100d0:  000f8893     addi    a7, t6, 0
   100d4:  00000813     addi    a6, zero, 0
000100d8   <L1>:
   100d8:  00088693     addi    a3, a7, 0
   100dc:  000e0793     addi    a5, t3, 0
```

```
  100e0:  00000613     addi    a2, zero, 0
000100e4   <L0>:
  100e4:  00078703       lb    a4, 0(a5)
  100e8:  00069583       lh    a1, 0(a3)
  100ec:  00178793     addi    a5, a5, 1
  100f0:  02868693     addi    a3, a3, 40
  100f4:  02b70733      mul    a4, a4, a1
  100f8:  00e60633      add    a2, a2, a4
  100fc:  fea794e3      bne    a5, a0, 0x100e4 <L0>
  10100:  00c32023       sw    a2, 0(t1)
  10104:  00280813     addi    a6, a6, 2
  10108:  00430313     addi    t1, t1, 4
  1010c:  00288893     addi    a7, a7, 2
  10110:  fdd814e3      bne    a6, t4, 0x100d8 <L1>
  10114:  050f0f13     addi    t5, t5, 80
  10118:  01478513     addi    a0, a5, 20
  1011c:  fa5f16e3      bne    t5, t0, 0x100c8 <L2>
  10120:  00008067     jalr    zero, 0(ra)

.symtab
Symbol Value         Size Type       Bind       Vis      Index Name
[   0] 0x0             0 NOTYPE   LOCAL    DEFAULT   UNDEF
[   1] 0x10074         0 SECTION  LOCAL    DEFAULT       1
[   2] 0x11124         0 SECTION  LOCAL    DEFAULT       2
[   3] 0x0             0 SECTION  LOCAL    DEFAULT       3
[   4] 0x0             0 SECTION  LOCAL    DEFAULT       4
[   5] 0x0             0 FILE     LOCAL    DEFAULT     ABS test.c
[   6] 0x11924         0 NOTYPE   GLOBAL   DEFAULT     ABS __global_pointer$
[   7] 0x118F4       800 OBJECT   GLOBAL   DEFAULT       2 b
[   8] 0x11124         0 NOTYPE   GLOBAL   DEFAULT       1 __SDATA_BEGIN__
[   9] 0x100AC       120 FUNC     GLOBAL   DEFAULT       1 mmul
[  10] 0x0             0 NOTYPE   GLOBAL   DEFAULT   UNDEF _start
[  11] 0x11124      1600 OBJECT   GLOBAL   DEFAULT       2 c
[  12] 0x11C14         0 NOTYPE   GLOBAL   DEFAULT       2 __BSS_END__
[  13] 0x11124         0 NOTYPE   GLOBAL   DEFAULT       2 __bss_start
[  14] 0x10074        28 FUNC     GLOBAL   DEFAULT       1 main
[  15] 0x11124         0 NOTYPE   GLOBAL   DEFAULT       1 __DATA_BEGIN__
[  16] 0x11124         0 NOTYPE   GLOBAL   DEFAULT       1 _edata
[  17] 0x11C14         0 NOTYPE   GLOBAL   DEFAULT       2 _end
[  18] 0x11764       400 OBJECT   GLOBAL   DEFAULT       2 a
```

Список источников.

1. Структура elf header: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#Section_header:~:text=header%5B4%5D-,File%20header,-%5Bedit%5D

2. Структура section header: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#Section_header:~:text=Program%20Header%20(size).-,Section%20header,-%5Bedit%5D
3. String table: String Table Section - Linker and Libraries Guide (oracle.com)
4. Symbol table: Symbol Table Section - Linker and Libraries Guide (oracle.com)
5. Официальная документация RISC-V: https://riscv.org/technical/specifications/#:~:text=Volume%201%2C%20Unprivileged%20Spec%20v.%2020191213%C2%A0%20%5BPDF%5D
6. RISC-V Calling conventions: riscv-elf-psabi-doc/riscv-cc.adoc at master · riscv-non-isa/riscv-elf-psabi-doc (github.com)

Листинг кода.

Main.java

```
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(InputStream input, OutputStream output) throws IOException,
UnsupportedFileFormatException {
        List<Integer> file = new ArrayList<>();
        int new_byte = input.read();
        while (new_byte != -1) {
            file.add(new_byte);
            new_byte = input.read();
        }
        ElfParser parser = new ElfParser(file, output);
        parser.parse();
    }

    public static void main(String[] args) {
        try {
            InputStream input = new FileInputStream(args[0]);
            try {
```

```
            OutputStream output = new FileOutputStream(args[1]);
            try {
               main(input, output);
            } catch (UnsupportedFileFormatException e) {
               System.out.println("Unsupported file format: " + e.getMessage());
            } catch (IOException e) {
               System.out.println("I/O error: " + e.getMessage());
            }
         } catch (FileNotFoundException e) {
            System.out.println("Output file not found: " + e.getMessage());
         }
      } catch (FileNotFoundException e) {
         System.out.println("Input file not found: " + e.getMessage());
      } catch (Exception e) {
         System.out.println("Something went wrong, I give up: " + e.getMessage());
      }
   }
}
```

ElfSectionHeader.java

```
public class ElfSectionHeader {
   public String name;
   public int type;
   public int offset;
   public int size;
}
```

ElfSymbol.java

```
public class ElfSymbol {
   public String name;
   public int value;
   public int size;
   public String type;
   public String bind;
   public String vis;
   public String index;
}
```

UnsupportedFileFormatException.java

```
public class UnsupportedFileFormatException extends Exception {
   public UnsupportedFileFormatException(String mess) {
      super(mess);
   }
```

```
    public UnsupportedFileFormatException() {
        super();
    }
}
```

OpCodes.java

```
import java.util.List;
import java.util.Map;

public class OpCodes {
    public static Map<Integer, Map<Integer, Map<Integer, String>>> Rcodes = Map.ofEntries(
            Map.entry(0b0010011, Map.of(
                    0b001, Map.of(0b0000000, "slli"),
                    0b101, Map.of(0b0000000, "srli", 0b0100000, "srai")
            )),
            Map.entry(0b0110011, Map.of(
                    0b000, Map.of(0b0000000, "add", 0b0100000, "sub", 0b0000001, "mul"),
                    0b001, Map.of(0b0000000, "sll", 0b0000001, "mulh"),
                    0b010, Map.of(0b0000000, "slt", 0b0000001, "mulhsu"),
                    0b011, Map.of(0b0000000, "sltu", 0b0000001, "mulhu"),
                    0b100, Map.of(0b0000000, "xor", 0b0000001, "div"),
                    0b101, Map.of(0b0000000, "srl", 0b0100000, "sra", 0b0000001, "divu"),
                    0b110, Map.of(0b0000000, "or", 0b0000001, "rem"),
                    0b111, Map.of(0b0000000, "and", 0b0000001, "remu")
            ))
    );

    public static Map<Integer, Map<Integer, String>> Icodes = Map.ofEntries(
            Map.entry(0b1100111, Map.of(0b000, "jalr")),
            Map.entry(0b0000011, Map.of(
                    0b000, "lb",
                    0b001, "lh",
                    0b010, "lw",
                    0b100, "lbu",
                    0b101, "lhu"
            )),
            Map.entry(0b0010011, Map.of(
                    0b000, "addi",
                    0b010, "slti",
                    0b011, "sltiu",
                    0b100, "xori",
                    0b110, "ori",
                    0b111, "andi"
            ))
    );

    public static Map<Integer, Map<Integer, String>> Scodes = Map.ofEntries(
            Map.entry(0b0100011, Map.of(
                    0b000, "sb",
```

```
                0b001, "sh",
                0b010, "sw"
        ))
    );

    public static Map<Integer, Map<Integer, String>> Bcodes = Map.ofEntries(
        Map.entry(0b1100011, Map.of(
                0b000, "beq",
                0b001, "bne",
                0b100, "blt",
                0b101, "bge",
                0b110, "bltu",
                0b111, "bgeu"
        ))
    );

    public static Map<Integer, String> Ucodes = Map.of(
        0b0110111, "lui",
        0b0010111, "auipc"
    );

    public static Map<Integer, String> Jcodes = Map.of(
        0b1101111, "jal"
    );

    public static Map<Integer, Character> types = Map.ofEntries(
        Map.entry(0b0110111, 'U'),
        Map.entry(0b0010111, 'U'),
        Map.entry(0b1101111, 'J'),
        Map.entry(0b1100111, 'I'),
        Map.entry(0b1100011, 'B'),
        Map.entry(0b0000011, 'I'),
        Map.entry(0b0100011, 'S'),
        Map.entry(0b0010011, 'I'),
        Map.entry(0b0110011, 'R')
    );

    public static List<String> registerNames = List.of(
        "zero", "ra", "sp", "gp", "tp", "t0", "t1", "t2", "s0", "s1", "a0", "a1", "a2", "a3", "a4",
        "a5", "a6", "a7", "s2", "s3", "s4", "s5", "s6", "s7", "s8", "s9", "s10", "s11", "t3", "t4", "t5", "t6"
    );
}
```

ElfParser.java

```
import java.io.IOException;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
import java.util.Objects;

public class ElfParser {
    public static final int SECTION_HEADER_SIZE = 40; // size of section header in bytes
    public static final int SYMBOL_SIZE = 16; // size of symbol in symbol table in bytes
    public static final String SYMBOL_TABLE_OUTPUT_HEADER = "\n.symtab\nSymbol Value          Size
Type       Bind       Vis      Index Name\n";
    public static final String SYMBOL_TABLE_LINE_TEMPLATE = "[%4d] 0x%-13X %5d %-8s %-8s %-
8s %6s %s\n";
    private final int[] elf;
    private final OutputStream out;
    private int shStringTableStart, stringTableStart, stringTableSize;
    private int symbolTableStart, symbolTableSize;
    private int textStart, textSize, textAddr;
    private final Map<Integer, String> labels;
    private String symtab;

    public ElfParser(List<Integer> elf, OutputStream out) {
        this.elf = new int[elf.size()];
        for (int i = 0; i < elf.size(); i++) {
            this.elf[i] = elf.get(i);
        }
        this.out = out;
        labels = new HashMap<>();
    }

    public void parse() throws UnsupportedFileFormatException, IOException {
        if (!(elf[0] == 0x7f &&
                elf[1] == 0x45 &&
                elf[2] == 0x4c &&
                elf[3] == 0x46)) {
            throw new UnsupportedFileFormatException("File is not ELF");
        }
        if (!(elf[4] == 1)) {
            throw new UnsupportedFileFormatException("File is not 32-bit");
        }
        if (!(elf[18] == 0xf3 && elf[19] == 0x00)) {
            throw new UnsupportedFileFormatException("File is not RISC-V");
        }
        if (!(elf[5] == 1)) {
            throw new UnsupportedFileFormatException("File is not little endian");
        }
        parseSectionHeaderTable(); // we need .text .symtab .strtab sections
        parseSymbolTable();
        parseText();
        out.write(symtab.getBytes());
    }

    private void parseSectionHeaderTable() {
        int sectionTablePosition = get4Bytes(32);
        int sectionTableSize = get2Bytes(48);
```

```
        shStringTableStart = get4Bytes(sectionTablePosition + get2Bytes(50) * SECTION_HEADER_SIZE
+ 16);
        for (int i = sectionTablePosition; i < sectionTablePosition + sectionTableSize *
SECTION_HEADER_SIZE; i += SECTION_HEADER_SIZE) {
            ElfSectionHeader header = new ElfSectionHeader();
            header.name = getSectionName(get4Bytes(i));
            header.type = get4Bytes(i + 4);
            header.offset = get4Bytes(i + 16);
            header.size = get4Bytes(i + 20);
            if (header.type == 0x2) {
                symbolTableStart = header.offset;
                symbolTableSize = header.size;
            }
            if (header.name.equals(".text")) {
                textStart = header.offset;
                textSize = header.size;
                textAddr = get4Bytes(i + 12);
            } else if (header.name.equals(".strtab")) {
                stringTableStart = header.offset;
                stringTableSize = header.size;
            }
        }
    }

    private void parseText() throws IOException {
        out.write(".text\n".getBytes());
        RISCVParser parser = new RISCVParser(elf, out, labels, textStart, textSize, textAddr);
        parser.parseText();
    }

    private void parseSymbolTable() throws IOException {
        StringBuilder resToWrite = new StringBuilder();
        resToWrite.append(SYMBOL_TABLE_OUTPUT_HEADER);
        int idx = 0;
        for (int i = symbolTableStart; i < symbolTableStart + symbolTableSize; i += SYMBOL_SIZE) {
            ElfSymbol symbol = new ElfSymbol();
            symbol.name = getSymbolName(get4Bytes(i));
            symbol.value = get4Bytes(i + 4);
            symbol.size = get4Bytes(i + 8);
            int info = elf[i + 12];
            symbol.type = getSymbolType((info) & 0xf);
            symbol.bind = getSymbolBind((info) >> 4);
            symbol.vis = getSymbolVis(elf[i + 13]);
            symbol.index = getSymbolIndex(get2Bytes(i + 14));
            if (Objects.equals(symbol.type, "FUNC")) {
                labels.put(symbol.value, symbol.name);
            }
            resToWrite.append(String.format(SYMBOL_TABLE_LINE_TEMPLATE, idx, symbol.value,
symbol.size, symbol.type,
                    symbol.bind, symbol.vis, symbol.index, symbol.name));
            idx++;
```

```
        }
        symtab = resToWrite.toString();
    }

    private String getSymbolType (int type) {
        return switch (type) {
            case 0 -> "NOTYPE";
            case 1 -> "OBJECT";
            case 2 -> "FUNC";
            case 3 -> "SECTION";
            case 4 -> "FILE";
            case 5 -> "COMMON";
            case 6 -> "TLS";
            case 10 -> "LOOS";
            case 12 -> "HIOS";
            case 13 -> "LOPROC";
            case 15 -> "HIPROC";
            default -> "";
        };
    }

    private String getSymbolBind (int bind) {
        return switch (bind) {
            case 0 -> "LOCAL";
            case 1 -> "GLOBAL";
            case 2 -> "WEAK";
            case 10 -> "LOOS";
            case 12 -> "HIOS";
            case 13 -> "LOPROC";
            case 15 -> "HIPROC";
            default -> "";
        };
    }

    private String getSymbolVis (int vis) {
        return switch (vis) {
            case 0 -> "DEFAULT";
            case 1 -> "INTERNAL";
            case 2 -> "HIDDEN";
            case 3 -> "PROTECTED";
            case 4 -> "EXPORTED";
            case 5 -> "SINGLETON";
            case 6 -> "ELIMINATE";
            default -> "";
        };
    }

    private String getSymbolIndex (int index) {
        return switch (index) {
            case 0 -> "UNDEF";
            case 0xff00 -> "LORESERVE";
```

```
            case 0xff01 -> "AFTER";
            case 0xff02 -> "AMD64_LCOMMON";
            case 0xff1f -> "HIPROC";
            case 0xff20 -> "LOOS";
            case 0xff3f -> "LOSUNW";
            case 0xfff1 -> "ABS";
            case 0xfff2 -> "COMMON";
            case 0xffff -> "XINDEX";
            default -> Integer.toString(index);
        };
    }

    private String getSymbolName (int offset) {
        if (offset == 0) {
            return "";
        }
        StringBuilder name = new StringBuilder();
        for (int idx = stringTableStart + offset; elf[idx] != 0; idx++) {
            name.append((char) elf[idx]);
        }
        return name.toString();
    }

    private String getSectionName(int offset) {
        StringBuilder name = new StringBuilder();
        for (int idx = shStringTableStart + offset; elf[idx] != 0; idx++) {
            name.append((char) elf[idx]);
        }
        return name.toString();
    }

    private int get4Bytes(int idx) {
        return elf[idx + 3] * 256 * 256 * 256 + elf[idx + 2] * 256 * 256 + elf[idx + 1] * 256 + elf[idx];
    }

    private int get2Bytes(int idx) {
        return elf[idx + 1] * 256 + elf[idx];
    }
}
```

RISCVParser.java

```
import java.io.IOException;
import java.io.OutputStream;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;
```

```
public class RISCVParser {
    private static final String LabelStrTemplate = "%08x   <%s>:\n";
    private static final String ThreeArgumentInstructionTemplate =
"   %05x:\t%08x\t%7s\t%s, %s, %s\n";
    private static final String ThreeArgumentBranchInstructionTemplate =
"   %05x:\t%08x\t%7s\t%s, %s, 0x%05x <%s>\n";
    private static final String TwoArgumentJalInstructionTemplate = "   %05x:\t%08x\t%7s\t%s,
0x%05x <%s>\n";
    private static final String TwoArgumentUtypeInstructionTemplate = "   %05x:\t%08x\t%7s\t%s,
0x%x\n";
    private static final String ZeroArgumentInstructionTemplate = "   %05x:\t%08x\t%7s\n";
    private static final String LoadStoreInstructionTemplate = "   %05x:\t%08x\t%7s\t%s, %s(%s)\n";
    private final int[] elf;
    private final OutputStream out;
    private final int textStart, textSize, textAddr;
    private final Map<Integer, String> labels;
    private int labelNum = 0;

    public RISCVParser(int[] elf, OutputStream out, Map<Integer, String> labels, int textStart, int
textSize, int textAddr) {
        this.elf = elf;
        this.out = out;
        this.labels = new HashMap<>(labels);
        this.textStart = textStart;
        this.textSize = textSize;
        this.textAddr = textAddr - textStart;
    }

    public void parseText() throws IOException {
        List<String> commands = new ArrayList<>();
        for (int i = textStart; i < textStart + textSize; i += 4) {
            commands.add(parseLine(i));
        }
        for (int i = textStart; i < textStart + textSize; i += 4) {
            checkForLabel(i);
            out.write(commands.get((i - textStart) / 4).getBytes());
        }
    }

    private void checkForLabel(int idx) throws IOException {
        if (labels.containsKey(idx + textAddr)) {
            out.write(String.format(LabelStrTemplate, idx + textAddr, labels.get(idx +
textAddr)).getBytes());
        }
    }

    private String getLabel(int value) {
        if (labels.containsKey(value)) {
            return labels.get(value);
        }
        labelNum++;
```

```
            labels.put(value, "L" + (labelNum - 1));
            return "L" + (labelNum - 1);
        }

        private String parseLine(int idx) {
            StringBuilder out = new StringBuilder();
            boolean[] code = convertToBits(get4Bytes(idx));
            int opcode = getBits(code, 0, 7);
            if (get4Bytes(idx) == 0b00000000000000000000000001110011) {
                out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"ecall"));
                return out.toString();
            } else if (get4Bytes(idx) == 0b00000000000100000000000001110011) {
                out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"ebreak"));
                return out.toString();
            }
            if (!OpCodes.types.containsKey(opcode)) {
                out.append(String.format(ZeroArgumentInstructionTemplate, idx + textAddr, get4Bytes(idx),
"unknown_instruction"));
                return out.toString();
            }
            char type = OpCodes.types.get(opcode);
            int immSign = code[31] ? (-1) : 1;
            if (type == 'R') {
                String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
                int funct3 = getBits(code, 12, 15);
                String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
                int funct7 = getBits(code, 25, 32);
                String command = OpCodes.Rcodes.get(opcode).get(funct3).get(funct7);
                if (opcode == 0b0010011) {
                    // shift
                    int shamt = getBits(code, 20, 25);
                    out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, rs1, shamt));
                    return out.toString();
                }
                String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
                out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, rs1, rs2));
                return out.toString();
            }
            if (type == 'I') {
                String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
                int funct3 = getBits(code, 12, 15);
                String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
                int imm = reverseImm(getBits(code,20, 31), immSign, 11);
                String command = OpCodes.Icodes.get(opcode).get(funct3);
                if (opcode == 0b0000011 || opcode == 0b1100111) {
                    // load command or jalr
                    out.append(String.format(LoadStoreInstructionTemplate, idx + textAddr, get4Bytes(idx),
```

```
command, rd, imm, rs1));
            } else {
                out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, rs1, imm));
            }
            return out.toString();
        }
        if (type == 'S') {
            int funct3 = getBits(code, 12, 15);
            String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
            String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
            int imm = reverseImm(getBits(code, 7, 12) + (getBits(code, 25, 31) << 5), immSign, 11);
            String command = OpCodes.Scodes.get(opcode).get(funct3);
            if (opcode == 0b0100011) {
                // store command
                out.append(String.format(LoadStoreInstructionTemplate, idx + textAddr, get4Bytes(idx),
command, rs2, imm, rs1));
            } else {
                out.append(String.format(ThreeArgumentInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rs1, rs2, imm));
            }
            return out.toString();
        }
        if (type == 'B') {
            int funct3 = getBits(code, 12, 15);
            String rs1 = OpCodes.registerNames.get(getBits(code, 15, 20));
            String rs2 = OpCodes.registerNames.get(getBits(code, 20, 25));
            int imm = reverseImm((getBits(code, 8, 12) << 1) +
                    (getBits(code, 25, 31) << 5) +
                    (getBits(code, 7, 8) << 11), immSign, 12);
            String command = OpCodes.Bcodes.get(opcode).get(funct3);
            out.append(String.format(ThreeArgumentBranchInstructionTemplate,
                    idx + textAddr, get4Bytes(idx), command, rs1, rs2, idx + textAddr + imm, getLabel(idx
+ textAddr + imm)));
            return out.toString();
        }
        if (type == 'U') {
            String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
            int imm = reverseImm(getBits(code, 12, 31), immSign, 31);
            String command = OpCodes.Ucodes.get(opcode);
            out.append(String.format(TwoArgumentUtypeInstructionTemplate, idx + textAddr,
get4Bytes(idx), command, rd, imm));
            return out.toString();
        }
        if (type == 'J') {
            // jal
            String rd = OpCodes.registerNames.get(getBits(code, 7, 12));
            int imm = reverseImm((getBits(code, 21, 25) << 1) + (getBits(code, 25, 31) << 5) +
                    (getBits(code, 20, 21) << 11 )+ (getBits(code, 12, 20) << 12), immSign, 20);
            String command = OpCodes.Jcodes.get(opcode);
            out.append(String.format(TwoArgumentJalInstructionTemplate, idx + textAddr,
```

```
get4Bytes(idx), command, rd,
                idx + textAddr + imm, getLabel(idx + textAddr + imm)));
        }
        return out.toString();
    }

    private boolean[] convertToBits(int x) {
        boolean[] res = new boolean[32];
        for (int i = 0; i < 32; i++) {
            res[i] = false;
        }
        String s = Integer.toBinaryString(x);
        for (int i = 32 - s.length(); i < 32; i++) {
            res[31 - i] = (s.charAt(i - 32 + s.length()) == '1');
        }
        return res;
    }

    private int reverseImm(int imm, int sign, int bitSize) {
        if (sign == 1) {
            return imm;
        }
        return ((1 << bitSize) - imm) * sign;
    }

    private int getBits(boolean[] n, int start, int end) {
        StringBuilder binNum = new StringBuilder();
        for (int i = end - 1; i >= start; i--) {
            binNum.append(n[i] ? '1' : '0');
        }
        return Integer.parseInt(binNum.toString(), 2);
    }

    private int get4Bytes(int idx) {
        return elf[idx + 3] * 256 * 256 * 256 + elf[idx + 2] * 256 * 256 + elf[idx + 1] * 256 + elf[idx];
    }
}
```