Лабораторная работа №3	M3139	2022
ISA	Гоге Анастасия Эдуардовна	

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

Инструментарий и требования к работе: Java 17, компилятор - javac 17.0.4.1.

1. Описание системы кодирования команд RISC-V.

RISC-V — это бесплатная открытая Reg-Reg архитектура набора инструкций (ISA), порядок байтов – little endian.

Описание RISC-V включает обязательный для реализации набор инструкций I и несколько стандартных расширений (таблица 1).

I	Базовый набор с целочисленными операциями
M	Целочисленное умножение и деление
A	Атомарные операции
F	Арифметические операции с плавающей запятой над числами одинарной точности
D	Арифметические операции с плавающей запятой над числами двойной точности
Q	Арифметические операции с плавающей запятой над числами четверной точности
Zifencei	Инструкции синхронизации потоков команд и данных
Zicsr	Инструкции для работы с контрольными и статусными регистрами

Таблица 1. Наборы инструкций. [3]

Используется 6 форматов кодирования инструкций (рисунок 1). Типы инструкций: R – reg-reg, I – immediate, S – store, B – branch, U – upper immediate, J – jump.

- opcode – код команды (или группы команд);
- funct3, funct7 – определяют команду, если несколько команд имеют одинаковый opcode;
- rs1 — номер регистра, в котором находится первый операнд;
- rs2 — номер регистра, в котором находится второй операнд;
- rd — номер регистра, в который будет записан результат;
- imm – константа, с которой выполняется операция в I и адрес перехода в S, B, U, J.

.

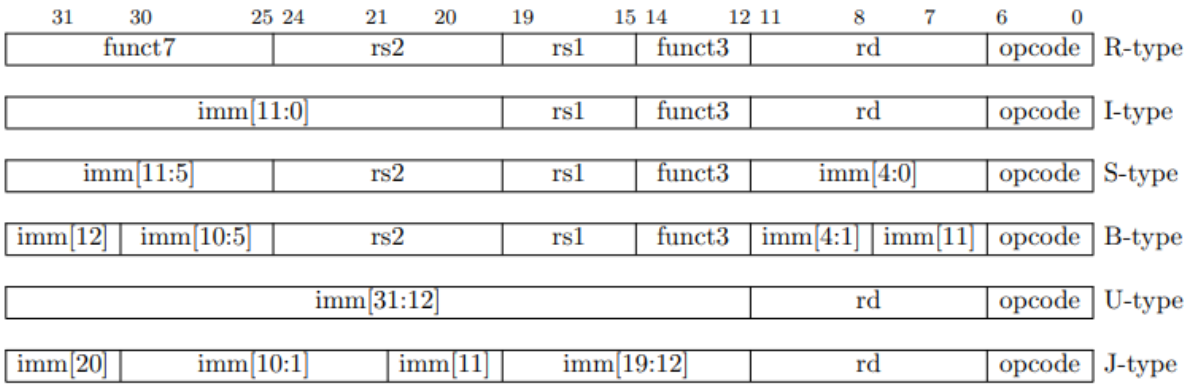

Рисунок 1. Формат записей о командах. [1]

Immediate собирается в число в соответствии с рисунком 2, при этом, если в промежутке написан 1 бит, то весь промежуток заполняется этим битом (например, в I-immediate биты с 11 по 31 будут равны 1, если inst[31] = 1, где inst – это закодированная инструкция).

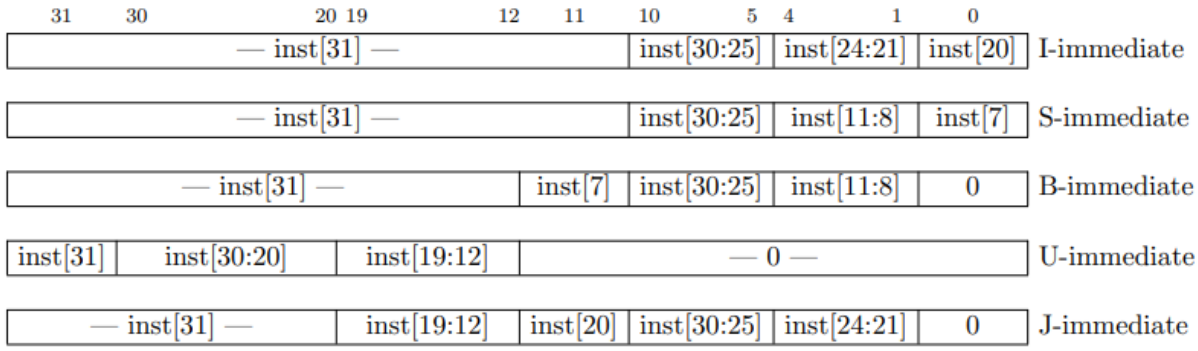

Рисунок 2. Формат констант. [1]

А регистры могут иметь названия, указанные в таблице 2, в зависимости от значения.

Значение	Имя
0	zero
1	ra
2	sp
3	gp
4	tp
5-7, 28-31	t0-t6
8-9, 18-27	s0-s11
10-17	a0-a7

Таблица 2. Регистры.

RV32I (Рисунок 3)

Инструкции регистр – константа:

Выполняют операции с константой(imm) и регистром rs1 и записывают результат в регистр rd. Тип инструкций I (U – для LUI и AUIPC)

- ADDI – сложение;
- SLTI/SLTIU – сравнения с учетом знака/без учета знака (1, если imm > rs1, и 0 в остальных случаях);
- ANDI, ORI, XORI – побитовые логические операции;
- SLLI/SRLI/SRAI – логические сдвиги (<</>>/>>>), имеют не совсем тип I, потому что вместо регистра в rs1 записана константа - сдвиг;
- LUI – записывает imm в старшие 20 битов rd, младшие 12 заполняет нулями;
- AUIPC – складывает imm << 12 и адрес AUIPC, записывает результат в rd.

Инструкции регистр – регистр:

Выполняют операции с регистрами rs1и rs2 и записывают результат в регистр rd. Тип инструкций R. Операции аналогичны I – операциям регистр – константа.

- ADD – сложение;
- SLT/SLTU – сравнения с учетом знака/без учета знака (1, если rs1 > rs2, и 0 в остальных случаях);
- AND, OR, XOR – побитовые логические операции;
- SLL/SRL/SRA – логические сдвиги (<</>>/>>>).

NOP инструкция:

Ничего не изменяет. Кодируется, как ADDI 0 0 0.

Переходы (jump):

- JAL – тип инструкции J, imm – смещение со знаком, сумма смещения и текущего адреса операции – это адрес перехода, в rd сохраняется адрес следующий после адреса перехода;
- JALR – тип инструкции I, адрес перехода – это сумма imm и rs1, в которой младший бит заменили на 0, в rd сохраняется адрес следующий после адреса перехода;

Переходы (branch):

Эти операции осуществляют переход при некоторых условиях. imm - – смещение со знаком, сумма смещения и текущего адреса операции – это адрес перехода.

- BEQ/BNE – выполняют переход, если регистры rs1 и rs2 равны/не равны;
- BLT/BLTU – выполняют переход, если rs1 меньше rs2, используя сравнение со знаком/без знака;
- BGE/BGEU – выполняют переход, если rs1 больше или равно rs2, используя сравнение со знаком/без знака.

Инструкции загрузки и сохранения (load и store):

В архитектуре RISC-V только инструкции загрузки и сохранения обращаются к памяти, а арифметические инструкции работают только с регистрами CPU. Load и store передают значение между регистрами и памятью. Load имеет тип операции I, a store - S.

Адрес для обращения к памяти получается сложением rs1 и imm – смещения со знаком. Load копирует значение из памяти в rd. Store копируют значение из rs2 в память.

- LW/LH/LB достают 32-/16-/8-битные значения из памяти. 16- и 8-битные дополняются до 32 первым битом в начале;
- LHU/LBU достают 16-/8-битные значения из памяти, они дополняются до 32 нулями в начале;
- SW/SH/SB сохраняют 32-/16-/8-битные значения из младших битов rs2 в память.

FENCE:

Инструкция FENCE используется для упорядочения ввода/вывода устройства и доступа к памяти с точки зрения просмотра другими RISC-V устройствами или процессорами.

ECALL и EBREAK:

Эти инструкции имеют тип инструкции I. ECALL используется для отправки служебного запроса в среду выполнения. EBREAK используется для возврата управления в среду отладки.

RV32M (Рисунок 4)

Эти инструкции имеют тип инструкции R.

Умножение:

- MUL умножает rs1 на rs2 и размещает младшие биты в rd;

- MULH/MULHU/MULHSU умножают знаковый rs1 и знаковый rs/беззнаковый rs1 и беззнаковый rs2/ знаковый rs1 и беззнаковый rs2.

Деление:

- DIV/DIVU выполняют знаковое/беззнаковое целочисленное деление rs1 на rs2 (округление к нулю);
- REM/REMU вычисляют остаток от соответствующего деления.

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. RV32I. [1]

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. RV32M. [1]

2. Описание структуры файла ELF.

ELF (Executable and Linkable Format) — формат исполняемых двоичных файлов.

ELF-header:

В начале файла всегда идет ELF-header. Его структура описана в таблице 3. Красным выделены части файла, которые нужно проверить на соответствие условиям задания. Зеленым шрифтом то, что нам понадобится для разбора файла, где именно понадобится будет указано дальше.

Адрес	Размер	Назначение
0x00	4	Общая характеристика файла.
0x04	1	Формат файла 32-битный или 64-битный (1 – 32).
0x05	1	Little endian или big endian (1 – little endian).
0x06	1	Версия ELF заголовка.
0x07	1	Специфичные для операционной системы или ABI расширения, используемые в файле.
0x08	1	Версия ABI.
0x09	1	Зарезервированные для будущего использования значения.
0x10	2	Тип файла.
0x12	2	Архитектура аппаратной платформы (0xF3 – RISC-V).
0x14	4	Номер версии формата.
0x18	4	Виртуальный адрес точки входа, которому система передает управление при запуске процесса.
0x1C	4	Смещение таблицы заголовков программы от начала файла в байтах. Если у файла нет таблицы заголовков программы, это поле содержит 0.
0x20	4	Смещение section header table от начала файла в байтах. Если у файла нет таблицы заголовков секций, это поле содержит 0.
0x24	4	Связанные с файлом флаги, зависящие от процессора. При их отсутствии это поле содержит 0.
0x28	2	Размер заголовка файла в байтах (52 для 32-битных файлов

		и 64 для 64-битных).
0x2A	2	Размер одного заголовка программы. Все заголовки программы имеют одинаковый размер (32 для 32-битных файлов и 56 для 64-битных).
0x2C	2	Число заголовков программы. Если у файла нет таблицы заголовков программы, это поле содержит 0.
0x2E	2	Размер одного заголовка секции. Все заголовки секций имеют одинаковый размер. 40 для 32-битных файлов.
0x30	2	Число заголовков секций. Если у файла нет таблицы заголовков секций, это поле содержит 0.
0x32	2	Индекс записи в таблице заголовков секций, описывающей таблицу названий секций. Если файл не содержит таблицы названий секций, это поле содержит 0.

Таблица 3. Структура ELF-header. [2]

Section Header Table:

Также в файле есть таблица заголовков секций (section header table), ее смещение относительно начала файла, число заголовков в ней и размер заголовка написаны в ELF-header(0x20, 0x30, 0x2E). В section header table содержатся заголовки секций (section header) для каждой секции в файле (таблица 4), в нем записывается смещение от начала секции названий секций(.shstrtab) названия секции(0x00), смещение от начала файла самой секции(0x10) и размер секции(0x14). В .shstrtab написаны названия секции, разделенные нулями, по 2 байта на символ. Номер заголовка .shstrtab среди остальных заголовков в section header table написан в ELF-header(0x32), поэтому можно найти его заголовок и нем его смещение от начала файла, а потом с помощью .shstrtab определять к какой секции тот или иной заголовок.

Адрес	Размер	Назначение
0x00	4	Смещение строки, содержащей название данной секции, относительно начала таблицы названий секций.
0x04	4	Тип заголовка.
0x08	4	Атрибуты секции.
0x0C	4	Если секция должна быть загружена в память при загрузке объектного файла, это поле указывает адрес, начиная с которого секция будет загружена, в противном случае поле содержит 0.
0x10	4	Смещение секции от начала файла в байтах.
0x14	4	Размер секции в файле. Может быть нулевым.
0x18	4	Индекс ассоциированной секции.
0x1C	4	Дополнительная информация о секции
0x20	4	Необходимое выравнивание секции.
0x24	4	Размер в байтах каждой записи.

Таблица 4. Структура записи об одной секции в section header table. [2]

Symbol Table:

В соответствующем section header можно найти смещение от начала файла и размер секции symbol table. В этой секции находятся записи вида: name -value-size-info-other-shndx (таблица 5). Bind, type, visibility и index вычисляются по таблицам 6, 7, 8 и 9 соответственно. В name записано смещение от начала секции .strtab, в которой записаны имена так же, как и в .shstrtab, соответственно, чтобы получить само имя, нужно его оттуда считать.

Размер в байтах	Поле
4	name
4	value
4	size
1	info
1	other
2	shndx

Таблица 5. Структура записи в symbol table [4]

Bind	info >> 4
LOCAL	0
GLOBAL	1
WEAK	2
LOOS	10
HIOS	12
LOPROC	13
HIPROC	15

Таблица 6. Bind [4]

Type	info & 15
NOTYPE	0
OBJECT	1
FUNC	2
SECTION	3
FILE	4
COMMON	5
TLS	6
LOOS	10
HIOS	12
LOPROC	13
SPARC_REGISTER	13
HIPROC	15

Таблица 7. Type [4]

Visibility	other & 3
DEFAULT	0
INTERNAL	1
HIDDEN	2
PROTECTED	3
EXPORTED	4
SINGLETON	5
ELIMINATE	6

Таблица 8. Visibility [4]

index	shndx
UNDEF	0
LORESERVE	0xff00
LOPROC	0xff00
BEFORE	0xff00
AFTER	0xff01
AMD64_LCOMMON	0xff02
HIPROC	0xff1f
LOOS	0xff20
LOSUNW	0xff3f
SUNW_IGNORE	0xff3f
HISUNW	0xff3f
HIOS	0xff3f
ABS	0xfff1
COMMON	0xfff2
XINDEX	0xffff
HIRESERVE	0xffff

Таблица 9. Index [4]

.text:

Также в соответствующем section header можно найти смещение от начала файла секции .text. Также нам понадобится виртуальный адрес этой секции, он записан в section header 0x0C. В этой секции находятся записи, соответствующие архитектуре RISC-V.

3. Описание работы написанного кода.

Чтобы запустить программу нужно записать имя входного elf-файла, как args[0], а выходного args[1]. Запустить программу можно, например, так:

javac Main.java && java Main input_file_name output_file_name

Класс Main:

Исполняемый класс, считывает файл, название которого записано в args[0], и записывает байты в массив elf (листинг 1).

```
ArrayList<Integer> elf = new ArrayList<>();
int b = in.read();
while (b != -1) {
    elf.add(b);
    b = in.read();
}
```

Листинг 1. Считывание файла.

Затем создает новый элемент класса Disassembler с этим файлом и запускает команду parseElf, результат которой будет записан в файл args[1].

Также этот класс отлавливает ошибки, если они случатся в течении программы.

Класс SymbolTable:

Хранит массив из SymbolTablePart(будет описано дальше) – symTab, и два словаря с метками marks (листинг 2).

```
private ArrayList<SymbolTablePart> symTab;
private final Map<Integer, String> marks;
```

Листинг 2. Поля класса SymbolTable.

Функция add (листинг 3) добавляет в массив запись и если эта запись типа FUNC, то записывает в словарь, хранящийся в SymbolTable пару значение – имя, чтобы в дальнейшем можно было доставать метки.

```
public void add(SymbolTablePart part) {
    symTab.add(part);
}
public void add(SymbolTablePart part, int value, String name) {
    symTab.add(part);
    marks.put(value, name);
}
```

Листинг 3. Функции add.

isMark (листинг 4) проверяет, если в словаре переданное значение, а getMark достает по значению метку.

Функция addInMarks (листинг 4) добавляет в словарь marks пару значение – метка.

```
public boolean isMark(int value) {
    return marks.containsKey(value);
}
public String getMark(int value) {
    return marks.get(value);
}
public void addInMarks(int value, String mark) {
    marksL.put(value, mark);
}
```

Листинг 4. Функции в классе SymbolTable

Функция print выводит SymbolTable в файл (листинг 5).

```
public void print() throws IOException {
    Disassembler.print("\n.symtab\n");
    Disassembler.print("Symbol Value              Size Type    Bind
    Vis          Index Name\n");
    for (SymbolTablePart part : symTab) {
        Disassembler.print(part.toFormatString());
    }
}
```

Листинг 5. print.

Класс SymbolTablePart:

Хранит одну запись из symbol table. В конструкторе получает массив из i, value, size, info & 15, info >> 4, other & 3, shndx, nameInt и переводит их в нужные нам данные так, как написано в описании ELF-файла в части про symbol table.

В функции toFormatString (листинг 6) возвращается строка для вывода в файл.

```
public String toFormatString() {
    return String.format(format, num, value, size, type, bind, vis,
index, name);
}
```

Листинг 6. toFormatString

Функция addInSymTab добавляет запись в SymbolTable. Если запись имеет тип FUNC, вызывает функцию add из SymbolTable с двумя аргуметами, если нет, то с одним.

Класс Disassembler:

В классе Disassembler происходит весь парсинг файла elf. В нем хранится сам файл, SymbolTable, Writer, который будет выводить файл, все форматы для вывода, адреса и размеры, которые могут понадобиться в процессе парсинга. Также есть глобальный счетчик numForL – число для

новых меток, которые мы создаем, когда осуществляется переход по адресу, у которого нет метки. Создание новой метки происходит в функции getMark (листинг 7). Она проверяет есть ли метка у адреса, если да, возвращает нужную, если нет, то создает новую и увеличивает счетчик на 1.

```
private String getMark(int addr) {
    if (symbolTable.isMark(addr)) {
        return symbolTable.getMark(addr);
    }
    String newMark = "L" + numForL++;
    symbolTable.addInMarks(addr, newMark);
    return newMark;
}
```

Листинг 7. getMark.

В конструкторе присваиваем elf считанный файл.

Основная функция parseElf принимает название output-файла, создает FileWriter и закрывает его по завершению работы. Далее запускается функция checkFile (листинг 8). В ней проверяется то, что выделено в таблице 3 красным и выкидываются ошибки, если нет.

```
private void checkFile() {
    if (getBytes(0x04, 1) != 1) {
        throw new MyException("Not 32-bit format. Such file format not
supported.");
    }
    if (getBytes(0x05, 1) != 1) {
        throw new MyException("Not little endian. Such file format not
supported.");
    }
    if (getBytes(0x12, 2) != 0xF3) {
        throw new MyException("Not RISC-V. Such file format not
supported.");
    }
}
```

Листинг 8. Проверка файла.

Далее считываем адреса, размеры и индексы в соответствии с тем, как описано их расположение в структуре ELF-файла. Функция getBytes (листинг 9) считывает столько байтов, сколько указано во втором аргументе, с адреса, указанного в первом аргументе, в соответствии с little endian. А функция findSectionIdx (листинг 10) проходится по заголовкам секций и ищет индекс заголовка с нужным именем и выкидывает ошибку, если такого заголовка нет. Имя она считывает из .shstrtab с помощью функции readString (листинг 11), которая читает строку по нужному адресу, пока не встретит 2 ноля.

```
private int getBytes(int firstByte, int countBytes) {
    int bytes = 0;
```

```
    for (int i = 0; i < countBytes; i++) {
        bytes += elf.get(firstByte + i) << (8 * i);
    }
    return bytes;
}
```

Листинг 9. getBytes.

```
private int findSectionIdx(String name) {
    int i = 0;
    while (i < countSectionHeaders) {
        int firstByteOfSectionHeader = addrSectionHeaderTable + 40 *
i;
        int addrInShSrtTab = addrShStrTab +
getBytes(firstByteOfSectionHeader, 4);
        String nameOfSection = readString(addrInShSrtTab);
        if (nameOfSection.equals(name)) {
            return i;
        }
        i++;
    }
    throw new MyException("There is no " + name + " header in file.");
}
```

Листинг 10. findSetionIdx.

```
public String readString(int addr) {
    int j = 1;
    char chr = (char) getBytes(addr, 1);
    StringBuilder str = new StringBuilder();
    while ((int) chr != 0) {
        str.append(chr);
        chr = (char) getBytes(addr + j, 1);
        j++;
    }
    return str.toString();
}
```

Листинг 11. readString.

Потом происходит парсинг Symbol Table в функции parseSymTab. В ней создается SymbolTable, потом считываются поля в соответствии с описанием структуры одного заголовка, создается SymbolTablePart и добавляется в SymbolTable. Размер одной записи 16 байт, поэтому идем до sizeSymTab / 16.

```
private void parseSymTab() {
    symbolTable = new SymbolTable();
    for (int i = 0; i < sizeSymTab / 16; i++) {
        int nameInt = getBytes(addrSymTab + i * 16, 4);
        int value = getBytes(addrSymTab + i * 16 + 4, 4);
        int size = getBytes(addrSymTab + i * 16 + 8, 4);
        int info = getBytes(addrSymTab + i * 16 + 12, 1);
        int other = getBytes(addrSymTab + i * 16 + 13, 1);
        int shndx = getBytes(addrSymTab + i * 16 + 14, 2);
```

```
        SymbolTablePart part = new SymbolTablePart(List.of(i, value,
size, info & 15, info >> 4, other & 3, shndx, nameInt));
        part.addInSymTab(symbolTable);
    }
}
```

Листинг 12. parseSymTab.

Дальше парсим .text. Одна запись занимает 4 байта, поэтому считываем sizeText / 4 записи с помощью функции parseTextPart. Сначала пройдем по .text и найдем все метки, а потом пройдем по .text и выведем в файл. В листинге 13 можно увидеть, что функция parseTextPart считывает текущий адрес в файле, чтобы считать line(это сама запись), и текущий виртуальных адрес (vAddr). Если у виртуального адреса есть метка, то к ответу, который потом вернет эта функция, добавим строку из адреса и метки. Потом функция считывает нужные поля из line в соответствии со структурой записи в RISC-V с помощью функции getInLine, которая возвращает биты с begin до end включительно. При этом в конце делается логический сдвиг >>, который заполняет все первые биты тем битом, что был в начале числа (imm[31]), что нам и нужно при считывании констант. Это не мешает при считывании funct7, так как у всех funct7 команд, которые мы разбирает старший бит – 0.

Регистры сразу получают названия с помощью функции getReg, которая переделывает значение в имя, как в таблице 2. В некоторых командах нет каких-то полей, в таком случае мы не будем их использовать.

```
int currAddr = addrText + 4 * i;
int line = getBytes(currAddr, 4);
int vAddr = virtualAddrText + 4 * i;
String ans = "";
if (symbolTable.isMark(vAddr)) {
    ans += String.format("\n%08x   <%s>:\n", vAddr,
symbolTable.getMark(vAddr));
}

String rd = getRegName(getInLine(line, 7, 11));
String rs1 = getRegName(getInLine(line, 15, 19));
String rs2 = getRegName(getInLine(line, 20, 24));
int funct3 = getInLine(line, 12, 14);
int funct7 = getInLine(line, 25, 31);
int opcode = getInLine(line, 0, 6);
String command = "";
```

Листинг 13. Часть parseTextPart.

Далее идет switch, который определяет команду по opcode, funct3 и funct7, считает константы, где они нужны, в соответствии с их описанием в RISC-V и добавляет строку с нужным форматом в ответ. И в конце ответ возвращается.

После вывода .text выводится SymbolTable с помощью функции print.

4. Результат работы написанной программы на приложенном к заданию файле (дизассемблер и таблицу 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         fence
   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     UND
[   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     UND _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
```

5. Список источников.

1. https://riscv.org/technical/specifications/ RISC-V. Specifications. (Volume 1, Unprivileged Spec v. 20191213)
2. https://en.wikipedia.org/wiki/Executable_and_Linkable_Format Wikipedia. Executable and Linkable Format.
3. https://en.wikipedia.org/wiki/RISC-V Wikipedia. RISC-V.
4. https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter7-27 Oracle. Linker and Libraries Guide. Symbol Table Section.

6. Листинг кода.

Main.java

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.NoSuchFileException;
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        try (InputStream in = new FileInputStream(args[0])) {
            ArrayList<Integer> elf = new ArrayList<>();
            int b = in.read();
            while (b != -1) {
                elf.add(b);
                b = in.read();
            }
            Disassembler dis = new Disassembler(elf);
```

```
            dis.parseElf(args[1]);
        } catch (NoSuchFileException e) {
            System.out.println("Where is elf-file? Give me my file!");
        } catch (IOException e) {
            System.out.println("IOException");
        } catch (MyException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Disassembler.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.ArrayList;
import java.util.List;

public class Disassembler {
    private static ArrayList<Integer> elf;
    private SymbolTable symbolTable;
    private static Writer writer;
    private int addrSectionHeaderTable;
    private int countSectionHeaders;
    private int sizeSecHedPart;
    private int addrShStrTab;
    private static int addrStrTab;
    private int addrSymTab;
    private int sizeSymTab;
    private int addrText;
    private int sizeText;
    private int virtualAddrText;
    private int numForL = 0;
    private final String outputFormat3 = "   %05x:\t%08x\t%7s\t%s, %s, %s\n";
    private final String outputFormat2Hex = "   %05x:\t%08x\t%7s\t%s, 0x%h\n";
    private final String outputFormatLSJ = "   %05x:\t%08x\t%7s\t%s, %s(%s)\n";
    private final String outputFormatJarl = "   %05x:\t%08x\t%7s\t%s, 0x%h
<%s>\n";
    private final String outputFormatB = "   %05x:\t%08x\t%7s\t%s, %s, 0x%h
<%s>\n";
    private final String outputFormatEF = "   %05x:\t%08x\t%7s\n";

    Disassembler(ArrayList<Integer> elf) {
            Disassembler.elf = elf;
    }

    public void parseElf(String outputFile) throws IOException {
        writer = new FileWriter(outputFile);
        try {
            checkFile();
            addrSectionHeaderTable = getBytes(0x20, 4);
            countSectionHeaders = getBytes(0x30, 2);
            sizeSecHedPart = getBytes(0x2E, 2);

            int idxShStrTab = getBytes(0x32, 2);
            addrShStrTab = getBytes(addrSectionHeaderTable + idxShStrTab *
```

```
sizeSecHedPart + 0x10, 4);

            int idxSymTab = findSectionIdx(".symtab");
            addrSymTab = getBytes(addrSectionHeaderTable + idxSymTab *
sizeSecHedPart + 0x10, 4);
            sizeSymTab = getBytes(addrSectionHeaderTable + idxSymTab *
sizeSecHedPart + 0x14, 4);

            int idxStrTab = findSectionIdx(".strtab");
            addrStrTab = getBytes(addrSectionHeaderTable + idxStrTab *
sizeSecHedPart + 0x10, 4);

            int idxText = findSectionIdx(".text");
            addrText = getBytes(addrSectionHeaderTable + idxText * sizeSecHedPart
+ 0x10, 4);
            sizeText = getBytes(addrSectionHeaderTable + idxText * sizeSecHedPart
+ 0x14, 4);
            virtualAddrText = getBytes(addrSectionHeaderTable + idxText *
sizeSecHedPart + 0x0C, 4);

            parseSymTab();

            writer.write(".text\n");
            for (int i = 0; i < sizeText / 4; i++) {
                parseTextPart(i);
            }
            for (int i = 0; i < sizeText / 4; i++) {
                writer.write(parseTextPart(i));
            }

            symbolTable.print();
        } finally {
            writer.close();
        }
    }

    private static int getBytes(int firstByte, int countBytes) {
        int bytes = 0;
        for (int i = 0; i < countBytes; i++) {
            bytes += elf.get(firstByte + i) << (8 * i);
        }
        return bytes;
    }
    private int getInLine(int line, int begin, int end) {
        int result = 0;
        for (int i = begin; i <= end; i++) {
            result |= ((line >> i) & 1) << i;
        }
        return result >> begin;
    }

    private String parseTextPart(int i) {
        int currAddr = addrText + 4 * i;
        int line = getBytes(currAddr, 4);
        int vAddr = virtualAddrText + 4 * i;
        String ans = "";
```

```
        if (symbolTable.isMark(vAddr)) {
            ans += String.format("%08x   <%s>:\n", vAddr,
symbolTable.getMark(vAddr));
        }

        String rd = getRegName(getInLine(line, 7, 11));
        String rs1 = getRegName(getInLine(line, 15, 19));
        String rs2 = getRegName(getInLine(line, 20, 24));
        int funct3 = getInLine(line, 12, 14);
        int funct7 = getInLine(line, 25, 31);
        int opcode = getInLine(line, 0, 6);
        String command = "";

        switch (opcode) {
            case 0b0110111 -> {
                command = "lui";
                int c = getInLine(line, 12, 31);
                ans += String.format(outputFormat2Hex, vAddr, getBytes(currAddr,
4), command, rd, c);
            }
            case 0b0010111 -> {
                command = "auipc";
                int c = getInLine(line, 12, 31);
                ans += String.format(outputFormat2Hex, vAddr, getBytes(currAddr,
4), command, rd, c);
            }
            case 0b1101111 -> {
                command = "jal";
                int c = getInLine(line, 31, 31) << 20;
                c |= getInLine(line,12, 19) << 12;
                c |= getInLine(line, 20, 20) << 11;
                c |= getInLine(line,21 , 30) << 1;
                c = virtualAddrText + c + i * 4;
                String mark = getMark(c);
                ans += String.format(outputFormatJarl, vAddr, getBytes(currAddr,
4), command, rd, c, mark);
            }
            case 0b1100111 -> {
                command = "jalr";
                int c = getInLine(line, 20, 31);
                ans += String.format(outputFormatLSJ, vAddr, getBytes(currAddr,
4), command, rd, c, rs1);
            }
            case 0b1100011 -> {
                switch (funct3) {
                    case 0b000 -> command = "beq";
                    case 0b001 -> command = "bne";
                    case 0b100 -> command = "blt";
                    case 0b101 -> command = "bge";
                    case 0b110 -> command = "bltu";
                    case 0b111 -> command = "bgeu";
                }
                int c = getInLine(line, 31, 31) << 12;
                c |= getInLine(line,7, 7) << 11;
                c |= getInLine(line, 25, 30) << 5;
                c |= getInLine(line,8 , 11) << 1;
```

```
                String mark = getMark(vAddr + c);
                ans += String.format(outputFormatB, vAddr, getBytes(currAddr, 4),
command, rs1, rs2, vAddr + c, mark);
            }
            case 0b0000011 -> {
                switch (funct3) {
                    case 0b000 -> command = "lb";
                    case 0b001 -> command = "lh";
                    case 0b010 -> command = "lw";
                    case 0b100 -> command = "lbu";
                    case 0b101 -> command = "lhu";
                }
                int c = getInLine(line, 20, 31);
                ans += String.format(outputFormatLSJ, vAddr, getBytes(currAddr,
4), command, rd, c, rs1);
            }
            case 0b0100011 -> {
                switch (funct3) {
                    case 0b000 -> command = "sb";
                    case 0b001 -> command = "sh";
                    case 0b010 -> command = "sw";
                }
                int c = (getInLine(line, 25, 31) << 5) + getInLine(line, 7, 11);
                ans += String.format(outputFormatLSJ, vAddr, getBytes(currAddr,
4), command, rs2, c, rs1);
            }
            case 0b0010011 -> {
                switch (funct3) {
                    case 0b001, 0b101 -> {
                        switch (funct3) {
                            case 0b001 -> command = "slli";
                            case 0b101 -> {
                                switch (funct7) {
                                    case 0b0000000 -> command = "srli";
                                    case 0b0100000 -> command = "srai";
                                }
                            }
                        }
                        int c = getInLine(line, 20, 24);
                        return String.format(outputFormat3, vAddr,
getBytes(currAddr, 4), command, rd, rs1, c);
                    }
                    default -> {
                        switch (funct3) {
                            case 0b000 -> command = "addi";
                            case 0b010 -> command = "slti";
                            case 0b011 -> command = "sltiu";
                            case 0b100 -> command = "xori";
                            case 0b110 -> command = "ori";
                            case 0b111 -> command = "andi";
                        }
                        int c = getInLine(line, 20, 31);
                        ans += String.format(outputFormat3, vAddr,
getBytes(currAddr, 4), command, rd, rs1, c);
                    }
                }
```

```
            }
            case 0b0110011 -> {
                switch (funct7) {
                    case 0b0000001 -> {
                        switch (funct3) {
                            case 0b000 -> command = "mul";
                            case 0b001 -> command = "mulh";
                            case 0b010 -> command = "mulhsu";
                            case 0b011 -> command = "mulhu";
                            case 0b100 -> command = "div";
                            case 0b101 -> command = "divu";
                            case 0b110 -> command = "rem";
                            case 0b111 -> command = "remu";
                        }
                    }
                    case 0b0000000 -> {
                        switch (funct3) {
                            case 0b000 -> command = "add";
                            case 0b001 -> command = "sll";
                            case 0b010 -> command = "slt";
                            case 0b011 -> command = "sltu";
                            case 0b100 -> command = "xor";
                            case 0b101 -> command = "srl";
                            case 0b110 -> command = "or";
                            case 0b111 -> command = "and";
                        }
                    }
                    case 0b0100000 -> {
                        switch (funct3) {
                            case 0b000 -> command = "sub";
                            case 0b101 -> command = "sra";
                        }
                    }
                }
                ans += String.format(outputFormat3, vAddr, getBytes(currAddr, 4),
command, rd, rs1, rs2);
            }
            case 0b1110011 -> {
                switch (getBytes(currAddr + 2, 1) >> 4) {
                    case 0b0000 -> command = "ecall";
                    case 0b0001 -> command = "ebreak";
                }
                ans += String.format(outputFormatEF, vAddr, getBytes(currAddr, 4),
command);
            }
            case 0b0001111 -> {
                command = "fence";
                ans += String.format(outputFormatEF, vAddr, getBytes(currAddr, 4),
command);
            }
            default -> ans += String.format(outputFormatEF, vAddr,
getBytes(currAddr, 4), "unknown_instruction");
        }
        return ans;
    }
```

```
    private void parseSymTab() {
        symbolTable = new SymbolTable();
        for (int i = 0; i < sizeSymTab / 16; i++) {
            int nameInt = getBytes(addrSymTab + i * 16, 4);
            int value = getBytes(addrSymTab + i * 16 + 4, 4);
            int size = getBytes(addrSymTab + i * 16 + 8, 4);
            int info = getBytes(addrSymTab + i * 16 + 12, 1);
            int other = getBytes(addrSymTab + i * 16 + 13, 1);
            int shndx = getBytes(addrSymTab + i * 16 + 14, 2);
            SymbolTablePart part = new SymbolTablePart(List.of(i, value, size,
info & 15, info >> 4, other & 3, shndx, nameInt));
            part.addInSymTab(symbolTable);
        }
    }

    private int findSectionIdx(String name) {
        int i = 0;
        while (i < countSectionHeaders) {
            int firstByteOfSectionHeader = addrSectionHeaderTable + sizeSecHedPart
* i;
            int addrInShSrtTab = addrShStrTab + getBytes(firstByteOfSectionHeader,
4);
            String nameOfSection = readString(addrInShSrtTab);
            if (nameOfSection.equals(name)) {
                return i;
            }
            i++;
        }
        throw new DisassemblerException("There is no " + name + " header in
file.");
    }

    public static void print(String str) throws IOException {
        writer.write(str);
    }

    public static String readName(int num) {
        return readString(addrStrTab + num);
    }
    public static String readString(int addr) {
        int j = 1;
        char chr = (char) getBytes(addr, 1);
        StringBuilder str = new StringBuilder();
        while ((int) chr != 0) {
            str.append(chr);
            chr = (char) getBytes(addr + j, 1);
            j++;
        }
        return str.toString();
    }
    private String getMark(int addr) {
        if (symbolTable.isMark(addr)) {
            return symbolTable.getMark(addr);
        }
        String newMark = "L" + numForL++;
        symbolTable.addInMarks(addr, newMark);
```

```
        return newMark;
    }
    private void checkFile() {
        if (getBytes(0x04, 1) != 1) {
            throw new DisassemblerException("Not 32-bit format. Such file format
not supported.");
        }
        if (getBytes(0x05, 1) != 1) {
            throw new DisassemblerException("Not little endian. Such file format
not supported.");
        }
        if (getBytes(0x12, 2) != 0xF3) {
            throw new DisassemblerException("Not RISC-V. Such file format not
supported.");
        }
    }
    private String getRegName(int num) {
        return switch (num) {
            case 0 -> "zero";
            case 1 -> "ra";
            case 2 -> "sp";
            case 3 -> "gp";
            case 4 -> "tp";
            case 5, 6, 7 -> "t" + (num - 5);
            case 8, 9 -> "s" + (num - 8);
            case 10, 11, 12, 13, 14, 15, 16, 17 -> "a" + (num - 10);
            case 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 -> "s" + (num - 16);
            case 28, 29, 30, 31 -> "t" + (num - 25);
            default -> null;
        };
    }
}
```

SymbolTable.java

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class SymbolTable {
    private ArrayList<SymbolTablePart> symTab;
    private final Map<Integer, String> marks;

    SymbolTable() {
        symTab = new ArrayList<>();
        marks = new HashMap<>();
    }

    public void add(SymbolTablePart part) {
        symTab.add(part);
    }
    public void add(SymbolTablePart part, int value, String name) {
        symTab.add(part);
        marks.put(value, name);
    }
```

```
    public boolean isMark(int value) {
        return marks.containsKey(value);
    }
    public String getMark(int value) {
        return marks.get(value);
    }
    public void addInMarks(int value, String mark) {
        marksL.put(value, mark);
    }
    public void print() throws IOException {
        Disassembler.print("\n.symtab\n");
        Disassembler.print("Symbol Value              Size Type    Bind    Vis
Index Name\n");
        for (SymbolTablePart part : symTab) {
            Disassembler.print(part.toFormatString());
        }
    }
}
```

SymbolTablePart.java

```
import java.util.List;

public class SymbolTablePart {
    private int num;
    private int value;
    private int size;
    private String type;
    private String bind;
    private String vis;
    private String index;
    private String name;
    private final String format = "[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s\n";

    SymbolTablePart(List<Integer> inf) {
        num = inf.get(0);
        value = inf.get(1);
        size = inf.get(2);
        type = getType(inf.get(3));
        bind = getBind(inf.get(4));
        vis = getVis(inf.get(5));
        index = getIndex(inf.get(6));
        name = getName(inf.get(7));
    }

    public String toFormatString() {
        return String.format(format, num, value, size, type, bind, vis, index,
name);
    }

    public void addInSymTab(SymbolTable symTab) {
        if (type.equals("FUNC")) {
            symTab.add(this, value, name);
        } else {
            symTab.add(this);
        }
    }
```

```
    private String getType(int num) {
        return switch (num) {
            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 -> throw new MyException("There is unsupported type in
file.");
        };
    }

    private String getBind(int num) {
        return switch (num) {
            case 0 -> "LOCAL";
            case 1 -> "GLOBAL";
            case 2 -> "WEAK";
            case 10 -> "LOOS";
            case 12 -> "HIOS";
            case 13 -> "LOPROC";
            case 15 -> "HIPROC";
            default -> throw new MyException("There is unsupported bind in
file.");
        };
    }

    private String getVis(int num) {
        return switch (num) {
            case 0 -> "DEFAULT";
            case 1 -> "INTERNAL";
            case 2 -> "HIDDEN";
            case 3 -> "PROTECTED";
            case 4 -> "EXPORTED";
            case 5 -> "SINGLETON";
            case 6 -> "ELIMINATE";
            default -> throw new MyException("There is unsupported visibility in
file.");
        };
    }

    private String getIndex(int num) {
        return switch (num) {
            case 0 -> "UND";
            case 0xff00 -> "BEFORE";
            case 0xff01 -> "AFTER";
            case 0xff02 -> "AMD64_LCOMMON";
            case 0xff1f -> "HIPROC";
            case 0xff20 -> "LOOS";
            case 0xff3f -> "HIOS";
            case 0xfff1 -> "ABS";
```

```
            case 0xfff2 -> "COMMON";
            case 0xffff -> "HIRESERVE";
            default -> Integer.toString(num);
        };
    }

    private String getName(int num) {
        if (num == 0) {
            return "";
        } else {
            return Disassembler.readName(num);
        }
    }

}
```

Листинг 14. Весь код