Лабораторная работа №3	M3139	2022
ISA	Гришечкин Павел Аверьянович	

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

Инструментарий и требования к работе: Работа выполнена на java 17.

Описание

1. Изучить систему кодирования команд RISC-V.
2. Изучить структуру elf файла.
3. Написать программу-транслятор (дизассемблер), с помощью которой можно преобразовывать машинный код в текст программы на языке ассемблера.

Вариант

В задании необходимо декодировать команды из наборов RV32I и M (Таблица 2). Все команды даны в 32х битном формате.

Система кодирования команд RISC-V

RISC-V – это Reg-Reg ISA.

В архитектуре RISC-V имеется обязательное для реализации небольшое подмножество команд (набор инструкций I — Integer) и несколько стандартных опциональных расширений.

В базовый набор входят инструкции условной и безусловной передачи управления/ветвления, минимальный набор арифметических/битовых операций на регистрах, операций с памятью (load/store), а также небольшое число служебных инструкций.

При одинаковой кодировке инструкций в RISC-V предусмотрены реализации архитектур с 32, 64 и 128-битными регистрами общего назначения и операциями (RV32I, RV64I и RV128I соответственно).

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

Нет операций над частями регистров, нет каких-либо выделенных «регистровых пар».

Архитектура использует только модель little-endian — первый байт операнда в памяти соответствует младшим битам значений регистрового операнда.

Name	ABI Mnemonic	Meaning
x0	zero	Zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5 - x7	t0 - t2	Temporary registers
x8 - x9	s0 - s1	Callee-saved registers
x10 - x17	a0 - a7	Argument registers
x18 - x27	s2 - s11	Callee-saved registers
x28 - x31	t3 - t6	Temporary registers

Таблица 1 – Регистры RISC-V (Источник)

Сокращение	Наименование	Версия	Статус
Базовые наборы			
RVWMO	Базовая модель согласованности памяти	2.0	Ratified
RV32I	Базовый набор с целочисленными операциями, 32-битный	2.1	Ratified
RV64I	Базовый набор с целочисленными операциями, 64-битный	2.1	Ratified
RV32E	Базовый набор с целочисленными операциями для встраиваемых систем, 32-битный, 16 регистров	1.9	Draft
RV128I	Базовый набор с целочисленными операциями, 128-битный	1.7	Draft
Часть 1 Стандартные непривилигерованные наборы команд			
M	Целочисленное умножение и деление (Integer Multiplication and Division)	2.0	Ratified
A	Атомарные операции (Atomic Instructions)	2.1	Ratified
F	Арифметические операции с плавающей запятой над числами одинарной точности (Single-Precision Floating-Point)	2.2	Ratified
D	Арифметические операции с плавающей запятой над числами двойной точности (Double-Precision Floating-Point)	2.2	Ratified
Q	Арифметические операции с плавающей запятой над числами четверной точности	2.2	Ratified
C	Сокращённые имена для команд (Compressed Instructions)	2.2	Ratified
Counters	Инструкции для счетчиков производительности и таймеров -- наборы **Zicntr** и **Zihpm**	2.0	Draft
L	Арифметические операции над десятичными числами с плавающей запятой (Decimal Floating-Point)	0.0	Open
B	Битовые операции (Bit Manipulation)	0.36	Open
J	Двоичная трансляция и поддержка динамической компиляции (Dynamically Translated Languages)	0.0	Open
T	Транзакционная память (Transactional Memory)	0.0	Open
P	Короткие SIMD-операции (Packed-SIMD Instructions)	0.1	Open
V	Векторные расширения (Vector Operations)	1.0	Frozen
Zicsr	Инструкции для работы с контрольными и статусными регистрами (Control and Status Register (CSR) Instructions)	2.0	Ratified
Zifencei	Инструкции синхронизации пототоков команд и данных (Instruction-Fetch Fence)	2.0	Ratified
Zihintpause	Pause Hint	2.0	Ratified
Zihintntl	Non-Temporal Locality Hints	0.2	Draft
Zam	Расширение для смещённых атомарных операций (Extension for Misaligned Atomics)	0.1	Draft
Zfh	Extensions for Half-Precision Floating-Point	1.0	Ratified
Zfhmin	Extensions for Half-Precision Floating-Point	1.0	Ratified
Zfinx	Standard Extensions for Floating-Point in Integer Registers	1.0	Ratified
Zdinx	Standard Extensions for Floating-Point in Integer Registers	1.0	Ratified
Zhinx	Standard Extensions for Floating-Point in Integer Registers	1.0	Ratified
Zhinxmin	Standard Extensions for Floating-Point in Integer Registers	1.0	Ratified
Ztso	Расширение для модели согласованности памяти RVTSO (Extension for Total Store Ordering)	0.1	Frozen
G	**= IMAFD Zicsr Zifencei** Обобщенное/сокращённое обозначение для набора расширений	н/д	н/д
Часть 2 Стандартные наборы команд для привилегированных режимов			
Machine ISA	Инструкции аппаратного уровня	1.12	Ratified
Supervisor ISA	Инструкции уровня супервизора	1.12	Ratified
Svnapot Extension	(Extension for NAPOT Translation Contiguity)	1.0	Ratified
Svpbmt Extension	(Extension for Page-Based Memory Types)	1.0	Ratified
Svinval Extension	(Extension for Fine-Grained Address-Translation Cache Invalidation)	1.0	Ratified
Hypervisor ISA	Инструкции уровня гипервизора	1.0	Ratified

Таблица 2 – Список наборов команд RISC-V (Источник)

Format	Bit																												
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

- *opcode* (7 bits): Partially specifies which of the 6 types of *instruction formats*.
- *funct7*, and *funct3* (10 bits): These two fields, further than the *opcode* field, specify the operation to be performed.
- *rs1*, *rs2*, or *rd* (5 bits): Specifies, by index, the register, resp., containing the first operand (i.e., source register), second operand, and destination register to which the computation result will be directed.

Таблица 3 – Формат 32-битной машинной команды (Источник)

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

Таблица 4 – RV32I Base Instruction Set (Источник)

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

Таблица 5 – RV32M Standard Extension (Источник)

В задании необходимо декодировать команды из таблиц 4, 5 (кроме команды FENCE)

Структура ELF файлов

ELF Header

Position (32 bit)	Position (64 bit)	Value
0-3	0-3	Magic number - 0x7F, then 'ELF' in ASCII
4	4	1 = 32 bit, 2 = 64 bit
5	5	1 = little endian, 2 = big endian
6	6	ELF header version
7	7	OS ABI - usually 0 for System V
8-15	8-15	Unused/padding
16-17	16-17	1 = relocatable, 2 = executable, 3 = shared, 4 = core
18-19	18-19	Instruction set - see table below
20-23	20-23	ELF Version
24-27	24-31	Program entry position
28-31	32-39	Program header table position
32-35	40-47	Section header table position
36-39	48-51	Flags - architecture dependent; see note below
40-41	52-53	Header size
42-43	54-55	Size of an entry in the program header table
44-45	56-57	Number of entries in the program header table
46-47	58-59	Size of an entry in the section header table
48-49	60-61	Number of entries in the section header table
50-51	62-63	Index in section header table with the section names

Таблица 6 – Структура заголовка ELF файла (Источник)

Из таблицы 6 необходимы:

- Первые 3 строки для проверки корректности заданного файла
- Section header table position (байты 32-35)
- Количество секций в section header table (байты 48-49)
- Номер секции таблицы с именами секций (байты 50-51)

Section header table

<table>
<tr><th colspan="2">Offset</th><th colspan="2">Size (bytes)</th><th rowspan="2">Field</th><th rowspan="2">Purpose</th></tr>
<tr><th>32-bit</th><th>64-bit</th><th>32-bit</th><th>64-bit</th></tr>
<tr><td>0x00</td><td></td><td>4</td><td></td><td>sh_name</td><td>An offset to a string in the .shstrtab section that represents the name of this section.</td></tr>
<tr><td>0x04</td><td></td><td>4</td><td></td><td>sh_type</td><td>Identifies the type of this header.
<table>
<tr><th>Value</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0x0</td><td>SHT_NULL</td><td>Section header table entry unused</td></tr>
<tr><td>0x1</td><td>SHT_PROGBITS</td><td>Program data</td></tr>
<tr><td>0x2</td><td>SHT_SYMTAB</td><td>Symbol table</td></tr>
<tr><td>0x3</td><td>SHT_STRTAB</td><td>String table</td></tr>
<tr><td>0x4</td><td>SHT_RELA</td><td>Relocation entries with addends</td></tr>
<tr><td>0x5</td><td>SHT_HASH</td><td>Symbol hash table</td></tr>
<tr><td>0x6</td><td>SHT_DYNAMIC</td><td>Dynamic linking information</td></tr>
<tr><td>0x7</td><td>SHT_NOTE</td><td>Notes</td></tr>
<tr><td>0x8</td><td>SHT_NOBITS</td><td>Program space with no data (bss)</td></tr>
<tr><td>0x9</td><td>SHT_REL</td><td>Relocation entries, no addends</td></tr>
<tr><td>0x0A</td><td>SHT_SHLIB</td><td>Reserved</td></tr>
<tr><td>0x0B</td><td>SHT_DYNSYM</td><td>Dynamic linker symbol table</td></tr>
<tr><td>0x0E</td><td>SHT_INIT_ARRAY</td><td>Array of constructors</td></tr>
<tr><td>0x0F</td><td>SHT_FINI_ARRAY</td><td>Array of destructors</td></tr>
<tr><td>0x10</td><td>SHT_PREINIT_ARRAY</td><td>Array of pre-constructors</td></tr>
<tr><td>0x11</td><td>SHT_GROUP</td><td>Section group</td></tr>
<tr><td>0x12</td><td>SHT_SYMTAB_SHNDX</td><td>Extended section indices</td></tr>
<tr><td>0x13</td><td>SHT_NUM</td><td>Number of defined types.</td></tr>
<tr><td>0x60000000</td><td>SHT_LOOS</td><td>Start OS-specific.</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
</table></td></tr>
<tr><td>0x08</td><td></td><td>4</td><td>8</td><td>sh_flags</td><td>Identifies the attributes of the section.
<table>
<tr><th>Value</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0x1</td><td>SHF_WRITE</td><td>Writable</td></tr>
<tr><td>0x2</td><td>SHF_ALLOC</td><td>Occupies memory during execution</td></tr>
<tr><td>0x4</td><td>SHF_EXECINSTR</td><td>Executable</td></tr>
<tr><td>0x10</td><td>SHF_MERGE</td><td>Might be merged</td></tr>
<tr><td>0x20</td><td>SHF_STRINGS</td><td>Contains null-terminated strings</td></tr>
<tr><td>0x40</td><td>SHF_INFO_LINK</td><td>'sh_info' contains SHT index</td></tr>
<tr><td>0x80</td><td>SHF_LINK_ORDER</td><td>Preserve order after combining</td></tr>
<tr><td>0x100</td><td>SHF_OS_NONCONFORMING</td><td>Non-standard OS specific handling required</td></tr>
<tr><td>0x200</td><td>SHF_GROUP</td><td>Section is member of a group</td></tr>
<tr><td>0x400</td><td>SHF_TLS</td><td>Section hold thread-local data</td></tr>
<tr><td>0x0FF00000</td><td>SHF_MASKOS</td><td>OS-specific</td></tr>
<tr><td>0xF0000000</td><td>SHF_MASKPROC</td><td>Processor-specific</td></tr>
<tr><td>0x4000000</td><td>SHF_ORDERED</td><td>Special ordering requirement (Solaris)</td></tr>
<tr><td>0x8000000</td><td>SHF_EXCLUDE</td><td>Section is excluded unless referenced or allocated (Solaris)</td></tr>
</table></td></tr>
<tr><td>0x0C</td><td>0x10</td><td>4</td><td>8</td><td>sh_addr</td><td>Virtual address of the section in memory, for sections that are loaded.</td></tr>
<tr><td>0x10</td><td>0x18</td><td>4</td><td>8</td><td>sh_offset</td><td>Offset of the section in the file image.</td></tr>
<tr><td>0x14</td><td>0x20</td><td>4</td><td>8</td><td>sh_size</td><td>Size in bytes of the section in the file image. May be 0.</td></tr>
<tr><td>0x18</td><td>0x28</td><td>4</td><td></td><td>sh_link</td><td>Contains the section index of an associated section. This field is used for several purposes, depending on the type of section.</td></tr>
<tr><td>0x1C</td><td>0x2C</td><td>4</td><td></td><td>sh_info</td><td>Contains extra information about the section. This field is used for several purposes, depending on the type of section.</td></tr>
<tr><td>0x20</td><td>0x30</td><td>4</td><td>8</td><td>sh_addralign</td><td>Contains the required alignment of the section. This field must be a power of two.</td></tr>
<tr><td>0x24</td><td>0x38</td><td>4</td><td>8</td><td>sh_entsize</td><td>Contains the size, in bytes, of each entry, for sections that contain fixed-size entries. Otherwise, this field contains zero.</td></tr>
<tr><td>0x28</td><td>0x40</td><td colspan="3"></td><td>End of Section Header (size).</td></tr>
</table>

Таблица 7 – Описание section header-a (Источник)

Необходимы поля (Таблица 7):

- sh_name – индекс начала имени section-header-a в string table. Считывание происходит до первого 0 (Блок из таблицы 6)
- sh_addr – для секции .text – это будет виртуальный адрес первой команды
- sh_offset – указатель на начало секции в файле.
- sh_size – размер секции в байтах

Размер каждой секции – 40 байт

Symbol table

Секцию можно найти по имени .symtab.

Symbol table состоит из нескольких блоков, расположенных подряд.

```
typedef struct {
        Elf32_Word      st_name;
        Elf32_Addr      st_value;
        Elf32_Word      st_size;
        unsigned char   st_info;
        unsigned char   st_other;
        Elf32_Half      st_shndx;
} Elf32_Sym;
```

Таблица 8 – Структура блока symbol table-a (Источник)

Размер первых 3х полей – 4 байта. Размер следующих 2х – 1 байт. Размер последнего – 2 байта.

Размер одного блока – 16 байт

Поля bind и type можно получить из info:

- bind = info >> 4
- type = info & 0xf

Поле visibility можно получить из поля other:

1. visibility = other & 0x3

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

Таблица 9 – Перевод bind из числа в строку (Источник)

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

Таблица 10 – Перевод type из числа в строку (Источник)

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

Таблица 11 – Перевод visibility из числа в строку (Источник)

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

Таблица 12 – Перевод shndx из числа в строку (Источник)

st_name – индекс начала строки в .strtab (секцию можно найти в section header table)

st_value – виртуальный адрес блока.

Структура .text

.text можно найти в section header table по имени. Виртуальный адрес первой команды находится там же (sh_addr).

Команды в .text расположены подряд. Каждая команда занимает 4 байта (примеры команд можно найти в таблицах 4 и 5)

Описание работы кода

```
try (InputStream reader = new FileInputStream(args[0])) {
    int read = reader.read();
    while (read != -1) {
        file.add(read);
        read = reader.read();
    }

    ELFParser parser = new ELFParser(file);

    try (BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(
                    new FileOutputStream(args[1]), StandardCharsets.UTF_8)))
    {
        writeInFile(parser, writer);
    } catch (IOException e) {
        System.out.println("Output error, i give up! " + e.getMessage());
    }
} catch (IOException e) {
    System.out.println("Input error, i give up! " + e.getMessage());
}
```

Листинг кода 1 – Ввод/вывод данных (файл Disassembler.java)

В классе происходит стандартная работа по считыванию/записи данных. Список file будет содержать все байты ELF файла подряд.

ELFParser – класс, который парсит ELF файлы (только те, которые были в тз)

Метод writeInFile записывает результат работы программы в файл (принцип его работы будет объяснен позже)

```
private final List<Integer> file;
private final SymbolTable symbolTable;
public final int SECTION_HEADER_TABLE_POSITION;
public static final int SECTION_HEADER_SEGMENT_SIZE = 40;
public final int STRING_TABLE_HEADER_POSITION;
public final int STRING_TABLE_POSITION;
public final int SYMTAB_STRING_TABLE_POSITION;
public final int SYMTAB_STRING_TABLE_SIZE;
public final int SECTION_COUNT;
public final int TEXT_POSITION;
public final int TEXT_SIZE;
public final int TEXT_VIRTUAL_ADDRESS;
public final int SYMBOL_TABLE_POSITION;
public final int SYMBOL_TABLE_SIZE;
public final int COMMAND_COUNT;
public static final int SYMBOL_TABLE_SECTION_SIZE = 16;
public final int SYMBOL_TABLE_SECTION_COUNT;
```

Листинг кода 2 – Константы ELFParser-а (файл ELFParser.java)

Название поля	Значение
file	Байты ELF файла
symbolTable	Symbol table
SECTION_HEADER_TABLE_POSITION	Позиция начала section header table
SECTION_HEADER_SEGMENT_SIZE	Размер сегмента section header table
STRING_TABLE_HEADER_POSITION	Номер section header-а для string table-а
STRING_TABLE_POSITION	Индекс начала string table в file
SYMTAB_STRING_TABLE_POSITION	Позиция .strtab в file
SYMTAB_STRING_TABLE_SIZE	Размер .strtab
SECTION_COUNT	Количество секций в section header-е
TEXT_POSITION	Позиция начала .text в file
TEXT_SIZE	Размер .text
TEXT_VIRTUAL_ADDRESS	Виртуальный адрес первой команды из .text
SYMBOL_TABLE_POSITION	Позиция начала .symtab в file
SYMBOL_TABLE_SIZE	Размер .symtab
COMMAND_COUNT	Количество команд из .text
SYMBOL_TABLE_SECTION_SIZE	Размер одной секции в .symtab
SYMBOL_TABLE_SECTION_COUNT	Количество секций в .symtab

```
this.file = new ArrayList<>(file);

if (file.get(0) != 0x7f || file.get(1) != 0x45 || file.get(2) != 0x4c ||
file.get(3) != 0x46) {
    throw new UnsupportedOperationException("Unsupported file format");
}
if (file.get(4) != 1) {
    throw new UnsupportedOperationException("Supports only 32 bits file");
}
if (file.get(5) != 1) {
    throw new UnsupportedOperationException("Supports only little-endian file");
}
```

Листинг кода 3 – Часть конструктора ELFParser-а (файл ELFParser.java)

На вход подается список из байтов ELF файла.

В начале проверяется корректность файла (если на вход дан не 32х битный ELF файл с little-endian, то будет брошен exception)

```
SECTION_COUNT = getByte(48, 49);
SECTION_HEADER_TABLE_POSITION = getByte(32, 35);
STRING_TABLE_HEADER_POSITION = 40 * getByte(50, 51) +
SECTION_HEADER_TABLE_POSITION;
STRING_TABLE_POSITION = getByte(STRING_TABLE_HEADER_POSITION + 0x10,
STRING_TABLE_HEADER_POSITION + 0x10 + 4);
```

Листинг кода 4 – Поиск section header table/количества секций/секции с именами секций (файл ELFParser.java)

```
public int getByte(int start, int end) {
    int result = 0;
    for (int i = end; i >= start; i--) {
        result = (result << 8) + file.get(i);
    }
    return result;
}
```

Листинг кода 5 – Метод getByte (файл ELFParser.java)

getByte считывает байты в заданном отрезке и объединяет их в одно число (порядок байтов little-endian).

В условиях задания достаточно возвращать int.

Поиск в листинге кода 4 происходит по таблице 6.

```
int symbolTablePosition = 0;
int symbolTableSize = 0;

int textPosition = 0;
int textSize = 0;
int textVirtualAddress = 0;

int symtabStringTablePosition = 0;
int symtabStringTableSize = 0;

for (int i = 0; i < SECTION_COUNT; i++) {
    if (getSectionName(i).equals(".symtab")) {
        symbolTablePosition = getSectionOffset(i);
        symbolTableSize = getSectionSize(i);
    }
    if (getSectionName(i).equals(".text")) {
        textPosition = getSectionOffset(i);
        textSize = getSectionSize(i);
        textVirtualAddress = getSectionVirtualAddress(i);
    }
    if (getSectionName(i).equals(".strtab")) {
        symtabStringTablePosition = getSectionOffset(i);
        symtabStringTableSize = getSectionSize(i);
    }
}
```

Листинг кода 6 – Поиск необходимых секций (файл ELFParser.java)

```
public String getSectionName(int sectionPosition) {
    StringBuilder result = new StringBuilder();
    int stringTablePos = getSectionNamePosition(sectionPosition);
    while (getStringTableByte(stringTablePos) != 0) {
        result.append((char) getStringTableByte(stringTablePos++));
    }
    return result.toString();
}
```

Листинг кода 7 – Метод getSectionName (файл ELFParser.java)

```
public int getSectionHeaderBytes(int section, int start, int end) {
    start += SECTION_HEADER_TABLE_POSITION + section *
SECTION_HEADER_SEGMENT_SIZE;
    end += SECTION_HEADER_TABLE_POSITION + section *
SECTION_HEADER_SEGMENT_SIZE;
    return getByte(start, end);
}
```

Листинг кода 8 – Метод getSectionHeaderBytes (файл ELFParser.java)

```
public int getSectionNamePosition(int section) {
    return getSectionHeaderBytes(section, 0, 3);
}

public int getSectionOffset(int section) {
    return getSectionHeaderBytes(section, 0x10, 0x10 + 4);
}

public int getSectionSize(int section) {
    return getSectionHeaderBytes(section, 0x14, 0x14 + 4);
}

public int getSectionVirtualAddress(int section) {
    return getSectionHeaderBytes(section, 0x0c, 0x0c + 3);
}
```

Листинг кода 9 – Методы получения информации из section header table (файл ELFParser.java)

```
public int getStringTableByte(int pos) {
    return getByte(STRING_TABLE_POSITION + pos);
}
```

Листинг кода 10 – Получение байта из string table (файл ELFParser.java)

В листинге кода 8 описано получение отрезка байтов из указанной секции

В листинге кода 9 получена информация по таблице 7.

При получении имени секции (листинг кода 7) считывается индекс первого байта в string table. К ответу последовательно приписываются символы до первого 0.

В листинге кода 6 считывается информация для необходимых секций.

```
if (symbolTablePosition == 0) {
    throw new AssertionError("Symbol table not found");
}
SYMBOL_TABLE_POSITION = symbolTablePosition;
SYMBOL_TABLE_SIZE = symbolTableSize;
SYMBOL_TABLE_SECTION_COUNT = SYMBOL_TABLE_SIZE / SYMBOL_TABLE_SECTION_SIZE;

if (textPosition == 0) {
    throw new AssertionError("Text not found");
}
TEXT_POSITION = textPosition;
TEXT_SIZE = textSize;
COMMAND_COUNT = TEXT_SIZE / 4;
TEXT_VIRTUAL_ADDRESS = textVirtualAddress;

if (symtabStringTablePosition == 0) {
    throw new AssertionError("String table for symbol table not found");
}
SYMTAB_STRING_TABLE_POSITION = symtabStringTablePosition;
SYMTAB_STRING_TABLE_SIZE = symtabStringTableSize;
List<Integer> symtabStringTable = file.subList(SYMTAB_STRING_TABLE_POSITION,
        SYMTAB_STRING_TABLE_POSITION + SYMTAB_STRING_TABLE_SIZE);
```

Листинг кода 11 – Присвоение констант (файл ELFParser.java)

В листинге кода 11 проверяется корректность данных, полученных из листинга кода 6.

```
public SymtabSegment(int name, int value, int size, int info, int other, int
shndx, List<Integer> stringTable) {
    this.name = name;
    this.value = value;
    this.size = size;
    this.info = info;
    this.other = other;
    this.shndx = shndx;
    this.stringName = findStringName(stringTable);

    this.bind = this.info >> 4;
    this.type = this.info & 0xf;
    this.visibility = this.other & 0x3;
}

public String findStringName(List<Integer> stringTable) {
    int pos = name;
    StringBuilder result = new StringBuilder();
    while (stringTable.get(pos) != 0) {
        result.append((char)(int)stringTable.get(pos++));
    }
    return result.toString();
}
```

Листинг кода 12 – Конструктор SymtabSegment (файл SymtabSegment.java)

Листинг кода 12 основан на таблице 8 и комментариях к ней.

Получение имени аналогично получению имени секции.

```
private final SymtabSegment[] symbolTable;
private final Map<Integer, String> labelAddress;
private int lastLabel = 0;
public SymbolTable(SymtabSegment[] symbolTable) {
    this.symbolTable = symbolTable;

    labelAddress = new HashMap<>();
    for (SymtabSegment : symbolTable) {
        if (symtabSegment.getType() == 2) {
            labelAddress.put(symtabSegment.getValue(),
symtabSegment.getStringName());
        }
    }
}
```

Листинг кода 13 – Конструктор SymbolTable (файл SymbolTable.java)

labelAddress – map в котором ключ – адрес, значение – имя/лэйбл, назначенный этому адресу.

lastLabel – минимальный свободный лэйбл.

```
SymtabSegment[] symbolTableSegments = new
SymtabSegment[SYMBOL_TABLE_SECTION_COUNT];
for (int i = 0; i < SYMBOL_TABLE_SECTION_COUNT; i++) {
    symbolTableSegments[i] = new SymtabSegment(
     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE,
         SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 3),

     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 4,
         SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 4 + 3),

     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 8,
         SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 8 + 3),

     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 12),

     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 13),

     getByte(SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 14,
         SYMBOL_TABLE_POSITION + i * SYMBOL_TABLE_SECTION_SIZE + 14 + 1),

     symtabStringTable
    );
}
symbolTable = new SymbolTable(symbolTableSegments);
```

Листинг кода 14 – Построение symbol table (файл ELFParser.java)

Листинг кода 14 основан на таблице 8 и комментариях к ней.

```
for (int i = 0; i < COMMAND_COUNT; i++) {
    getCommandString(i);
}
```

Листинг кода 15 – первый проход по .text (файл ELFParser.java)

getCommandString – Метод, который возвращает строковое представление ій инструкции.

Проход по .text заранее необходим для предварительной установки лейблов на некоторые адреса.

```
public int getCommand(int number) {
      return getByte(number * 4 + TEXT_POSITION, number * 4 + TEXT_POSITION +
3);
}
public int getOpcode(int command) { return command & 0b1111111; }
public int getFunct3(int command) { return (command >> 12) & 0b111; }
public int getFunct7(int command) { return command >> 25; }
public String getRegisterName(int reg) {
    return switch (reg) {
        case 0 -> "zero";
        case 1 -> "ra";
        case 2 -> "sp";
        case 3 -> "gp";
        case 4 -> "tp";
        case 5 -> "t0";
        case 6 -> "t1";
        case 7 -> "t2";
        case 8 -> "s0";
        case 9 -> "s1";
        case 10 -> "a0";
        case 11 -> "a1";
        case 12 -> "a2";
        case 13 -> "a3";
        case 14 -> "a4";
        case 15 -> "a5";
        case 16 -> "a6";
        case 17 -> "a7";
        case 18 -> "s2";
        case 19 -> "s3";
        case 20 -> "s4";
        case 21 -> "s5";
        case 22 -> "s6";
        case 23 -> "s7";
        case 24 -> "s8";
        case 25 -> "s9";
        case 26 -> "s10";
        case 27 -> "s11";
        case 28 -> "t3";
        case 29 -> "t4";
        case 30 -> "t5";
        case 31 -> "t6";
        default -> throw new UnsupportedOperationException("Unsupported
register: " + "\"" + reg + "\"");
    };
}
```

Листинг кода 16 – Геттеры для getCommandString (файл ELFParser.java)

Листинг кода 16 основан на таблицах 1 и 3.

```
public int setBits(int number, int bits, int begin, int end) {
    return number % (1 << begin) + ((number >> end) << end) + (bits << begin);
}
public int getBits(int number, int begin, int end) {
    int result = 0;
    for (int i = begin; i <= end; i++) {
        if (((number >> i) & 1) == 1) {
            result |= (1 << i);
        }
    }
    return result >> begin;
}
```

Листинг кода 17 – Установка и получение битов из числа

(файл ELFParser.java)

```
public String getAddressLabel(int address) {
    if (!labelAddress.containsKey(address)) {
        labelAddress.put(address, "L" + lastLabel++);
    }
    return labelAddress.get(address);
}
```

Листинг кода 18 – Получение лейбла адреса (файл SymbolTable.java)

Метод возвращает имя/лейбл адреса, если он уже присвоен. Если у адреса нет имени, то присваивает минимальный не занятый лейбл.

```
public String getCommandString(int number, int addr) {
    int command = getCommand(number);
    StringBuilder parameters = new StringBuilder();
    String commandName = "";
    switch (getOpcode(command)) {
        case 0b0110111 -> {
            commandName = "lui";
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append("0x").append(Integer.toHexString(getBits(command,
12, 31)));
        }
        case 0b0010111 -> {
            commandName = "auipc";
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append("0x").append(Integer.toHexString(getBits(command,
12, 31)));
        }
        case 0b1101111 -> {
            commandName = "jal";
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            int current = setBits(0, getBits(command, 12, 19), 12, 19);
            current = setBits(current, getBits(command, 20, 20), 11, 11);
            current = setBits(current, getBits(command, 21, 30), 1, 10);
            current = setBits(current, getBits(command, 31, 31), 20, 20);
            parameters.append("0x").append(Integer.toHexString(addr + current));
            parameters.append(" <").append(getAddressLabel(addr +
current)).append(">");
```

```
        }
        case 0b1100111 -> {
            commandName = "jalr";
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append(getBits(command, 20, 31));
            parameters.append("(");
            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(")");
        }
        case 0b1100011 -> {
            switch (getFunct3(command)) {
                case 0b000 -> commandName = "beq";
                case 0b001 -> commandName = "bne";
                case 0b100 -> commandName = "blt";
                case 0b101 -> commandName = "bge";
                case 0b110 -> commandName = "bltu";
                case 0b111 -> commandName = "bgeu";
                default -> commandName = "unknown_instruction";
            }

            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(",");
            parameters.append(getRegisterName(getBits(command, 20, 24)));
            parameters.append(",");
            int current = setBits(0, getBits(command, 7, 7), 11, 11);
            current = setBits(current, getBits(command, 8, 11), 1, 4);
            current = setBits(current, getBits(command, 25, 30), 5, 10);
            current = setBits(current, getBits(command, 31, 31), 12, 12);
            parameters.append("0x").append(Integer.toHexString(addr + current));
            parameters.append(" <").append(getAddressLabel(addr +
current)).append(">");
        }
        case 0b0000011 -> {
            switch (getFunct3(command)) {
                case 0b000 -> commandName = "lb";
                case 0b001 -> commandName = "lh";
                case 0b010 -> commandName = "lw";
                case 0b100 -> commandName = "lbu";
                case 0b101 -> commandName = "lhu";
                default -> commandName = "unknown_instruction";
            }
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append(getBits(command, 20, 31));
            parameters.append("(");
            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(")");
        }
        case 0b0100011 -> {
            switch (getFunct3(command)) {
                case 0b000 -> commandName = "sb";
                case 0b001 -> commandName = "sh";
                case 0b010 -> commandName = "sw";
                default -> commandName = "unknown_instruction";
            }
            int current = getBits(command, 7, 11);
            current = setBits(current, getBits(command, 25, 31), 5, 11);
            parameters.append(getRegisterName(getBits(command, 20, 24)));
```

```
            parameters.append(",");
            parameters.append(current);
            parameters.append("(");
            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(")");
        }
        case 0b0010011 -> {
            switch (getFunct3(command)) {
                case 0b000 -> commandName = "addi";
                case 0b010 -> commandName = "slti";
                case 0b011 -> commandName = "sltiu";
                case 0b100 -> commandName = "xori";
                case 0b110 -> commandName = "ori";
                case 0b111 -> commandName = "andi";
                case 0b001 -> commandName = "slli";
                case 0b101 -> {
                    switch (getFunct7(command)) {
                        case 0b0000000 -> commandName = "srli";
                        case 0b0100000 -> commandName = "srai";
                        default -> commandName = "unknown_instruction";
                    }
                }
                default -> commandName = "unknown_instruction";
            }
            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(",");
            parameters.append(getBits(command, 20, 31));
        }
        case 0b0110011 -> {
            switch (getFunct7(command)) {
                case 0b0000000 -> {
                    switch (getFunct3(command)) {
                        case 0b000 -> commandName = "add";
                        case 0b001 -> commandName = "sll";
                        case 0b010 -> commandName = "slt";
                        case 0b011 -> commandName = "sltu";
                        case 0b100 -> commandName = "xor";
                        case 0b101 -> commandName = "srl";
                        case 0b110 -> commandName = "or";
                        case 0b111 -> commandName = "and";
                    }
                }
                case 0b0100000 -> {
                    switch (getFunct3(command)) {
                        case 0b000 -> commandName = "sub";
                        case 0b101 -> commandName = "sra";
                    }
                }
                case 0b0000001 -> {
                    switch (getFunct3(command)) {
                        case 0b000 -> commandName = "mul";
                        case 0b001 -> commandName = "mulh";
                        case 0b010 -> commandName = "mulhsu";
                        case 0b011 -> commandName = "mulhu";
                        case 0b100 -> commandName = "div";
                        case 0b101 -> commandName = "divu";
                        case 0b110 -> commandName = "rem";
```

```
                        case 0b111 -> commandName = "remu";
                    }
                }
                default -> commandName = "unknown_instruction";
            }

            parameters.append(getRegisterName(getBits(command, 7, 11)));
            parameters.append(",");
            parameters.append(getRegisterName(getBits(command, 15, 19)));
            parameters.append(",");
            parameters.append(getRegisterName(getBits(command, 20, 24)));
        }
        case 0b1110011 -> {
            switch (command >> 20) {
                case 0b000000000000 -> commandName = "ecall";
                case 0b000000000001 -> commandName = "ebreak";
                default -> commandName = "unknown_instruction";
            }
        }
        case 0b0001111 -> commandName = "fence";
        default -> commandName = "unknown_instruction";
    }

    return String.format("    %05x:   %08x      %5s %s", addr, command,
commandName, parameters);
}
```

Листинг кода 19 – Получение строкового представления команды (файл ELFParser.java)

Обрабатываются команды из таблиц 4, 5.

Метод принимает номер команды в .text и возвращает строковое представление команды.

```
public String getStringType() {
    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 -> {
            throw new UnsupportedOperationException("Unsupported symtab segment
type");
        }
    };
}
```

```
public String getStringBind() {
    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 -> {
            throw new UnsupportedOperationException("Unsupported symtab segment
bind");
        }
    };
}

public String getStringVisibility() {
    return switch(visibility) {
        case 0 -> "DEFAULT";
        case 1 -> "INTERNAL";
        case 2 -> "HIDDEN";
        case 3 -> "PROTECTED";
        case 4 -> "EXPORTED";
        case 5 -> "SINGLETON";
        case 6 -> "ELIMINATE";
        default -> {
            throw new UnsupportedOperationException("Unsupported symtab segment
visibility");
        }
    };
}

public String getStringShndx() {
    return switch(shndx) {
        case 0 -> "UNDEF";
        case 0xff00 -> "LORESERVE";
        case 0xff01 -> "AFTER";
        case 0xff02 -> "AMD64_LCOMMON";
        case 0xff1f -> "HIPROC";
        case 0xff20 -> "LOOS";
        case 0xff3f -> "HIOS";
        case 0xfff1 -> "ABS";
        case 0xfff2 -> "COMMON";
        case 0xffff -> "XINDEX";
        default -> Integer.toString(shndx);
    };
}
```

Листинг кода 20 – Получение строкового представления полей symtabSegment (файл SymtabSegment.java)

Листинг кода 20 основан на таблицах 9-12.

```
@Override
public String toString() {
    StringBuilder result = new StringBuilder();
    result.append("Symbol Value              Size Type     Bind    Vis
Index Name\n");
    for (int i = 0; i < symbolTable.length; i++) {
        result.append(String.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s\n",
                i,
                symbolTable[i].getValue(),
                symbolTable[i].getSize(),
                symbolTable[i].getStringType(),
                symbolTable[i].getStringBind(),
                symbolTable[i].getStringVisibility(),
                symbolTable[i].getStringShndx(),
                symbolTable[i].getStringName()
        ));
    }
    return result.toString();
}
```

Листинг кода 21 – Вывод symbol table (файл SymbolTable.java)

Переопределение метода toString для вывода symbol table.

Используются методы из листинга кода 20.

```
public String getSymbolTableString() {
    return symbolTable.toString();
}

public String getAddressLabel(int address) {
    return symbolTable.getAddressLabel(address);
}

public String getAddressName(int address) {
    return symbolTable.getAddressName(address);
}
```

Листинг кода 22 – Методы для работы с symbol table (файл ELFParser.java)

В листинге кода 22 представлены методы для работы с .symtab из класса ELFParser.

```
public String getAddressName(int address) {
    return labelAddress.getOrDefault(address, "");
}
```

Листинг кода 23 – Получение имени адреса (файл SymbolTable.java)

```java
private static void writeInFile(ELFParser parser, BufferedWriter writer) throws
IOException {
    writer.write("Disassembly of section .text:");
    writer.newLine();
    int addr = parser.TEXT_VIRTUAL_ADDRESS;
    for (int i = 0; i < parser.COMMAND_COUNT; i++) {
        String addressName = parser.getAddressName(addr);
        if (!addressName.isEmpty()) {
            writer.newLine();
            writer.write(String.format("%08x   <%s>:\n", addr, addressName));
        }
        writer.write(parser.getCommandString(i));
        writer.newLine();
        addr += 4;
    }

    writer.newLine();
    writer.write("SYMBOL TABLE:");
    writer.newLine();
    writer.write(parser.getSymbolTableString());
}
```

Листинг кода 24 – Вывод итогового результата в файл

(файл Disassembler.java)

1. На каждой итерации фора считывается имя адреса.
2. Если у адреса есть имя, то оно выводится в формате данном в тз.
3. Выводится строковое представление ій команды
4. В конце выводится symbol table

Результат работы программы

```
Disassembly of section .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)

SYMBOL TABLE:
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
```

Результат работы программы (файл test_elf_diasm)

Список источников

https://wiki.osdev.org/ELF

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-tbl-21

https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#Section_header

https://ru.wikipedia.org/wiki/RISC-V

https://riscv.org/technical/specifications/

Листинг кода

Disassembler.java

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

public class Disassembler {
    private static final List<Integer> file = new ArrayList<>();
    public static void main(String[] args) {
        try (InputStream reader = new FileInputStream(args[0])) {
            int read = reader.read();
            while (read != -1) {
                file.add(read);
                read = reader.read();
            }

            ELFParser parser = new ELFParser(file);

            try (BufferedWriter writer = new BufferedWriter(
                    new OutputStreamWriter(
                            new FileOutputStream(args[1]),
StandardCharsets.UTF_8)))
            {
                writeInFile(parser, writer);
            } catch (IOException e) {
                System.out.println("Output error, i give up! " +
e.getMessage());
            }
        } catch (IOException e) {
            System.out.println("Input error, i give up! " + e.getMessage());
        }
    }

    private static void writeInFile(ELFParser parser, BufferedWriter writer)
throws IOException {
        writer.write("Disassembly of section .text:");
        writer.newLine();
        int addr = parser.TEXT_VIRTUAL_ADDRESS;
        for (int i = 0; i < parser.COMMAND_COUNT; i++) {
```

```
            String addressName = parser.getAddressName(addr);
            if (!addressName.isEmpty()) {
                writer.newLine();
                writer.write(String.format("%08x   <%s>:\n", addr,
addressName));
            }
            writer.write(parser.getCommandString(i));
            writer.newLine();
            addr += 4;
        }

        writer.newLine();
        writer.write("SYMBOL TABLE:");
        writer.newLine();
        writer.write(parser.getSymbolTableString());
    }
}
```

ELFParser.java

```
import java.util.ArrayList;
import java.util.List;

@SuppressWarnings("DuplicatedCode")
public class ELFParser {
    private final List<Integer> file;
    private final SymbolTable symbolTable;
    public final int SECTION_HEADER_TABLE_POSITION;
    public static final int SECTION_HEADER_SEGMENT_SIZE = 40;
    public final int STRING_TABLE_HEADER_POSITION;
    public final int STRING_TABLE_POSITION;
    public final int SYMTAB_STRING_TABLE_POSITION;
    public final int SYMTAB_STRING_TABLE_SIZE;
    public final int SECTION_COUNT;
    public final int TEXT_POSITION;
    public final int TEXT_SIZE;
    public final int TEXT_VIRTUAL_ADDRESS;
    public final int SYMBOL_TABLE_POSITION;
    public final int SYMBOL_TABLE_SIZE;
    public final int COMMAND_COUNT;
    public static final int SYMBOL_TABLE_SECTION_SIZE = 16;
    public final int SYMBOL_TABLE_SECTION_COUNT;

    public ELFParser(final List<Integer> file) {
        this.file = new ArrayList<>(file);

        if (file.get(0) != 0x7f || file.get(1) != 0x45 || file.get(2) != 0x4c ||
file.get(3) != 0x46) {
            throw new UnsupportedOperationException("Unsupported file format");
        }
        if (file.get(4) != 1) {
            throw new UnsupportedOperationException("Supports only 32 bits
file");
        }
        if (file.get(5) != 1) {
            throw new UnsupportedOperationException("Supports only little-endian
file");
        }
```

```
        SECTION_COUNT = getByte(48, 49);
        SECTION_HEADER_TABLE_POSITION = getByte(32, 35);
        STRING_TABLE_HEADER_POSITION = 40 * getByte(50, 51) +
SECTION_HEADER_TABLE_POSITION;
        STRING_TABLE_POSITION = getByte(STRING_TABLE_HEADER_POSITION + 0x10,
STRING_TABLE_HEADER_POSITION + 0x10 + 4);

        int symbolTablePosition = 0;
        int symbolTableSize = 0;

        int textPosition = 0;
        int textSize = 0;
        int textVirtualAddress = 0;

        int symtabStringTablePosition = 0;
        int symtabStringTableSize = 0;

        for (int i = 0; i < SECTION_COUNT; i++) {
            if (getSectionName(i).equals(".symtab")) {
                symbolTablePosition = getSectionOffset(i);
                symbolTableSize = getSectionSize(i);
            }
            if (getSectionName(i).equals(".text")) {
                textPosition = getSectionOffset(i);
                textSize = getSectionSize(i);
                textVirtualAddress = getSectionVirtualAddress(i);
            }
            if (getSectionName(i).equals(".strtab")) {
                symtabStringTablePosition = getSectionOffset(i);
                symtabStringTableSize = getSectionSize(i);
            }
        }

        if (symbolTablePosition == 0) {
            throw new AssertionError("Symbol table not found");
        }
        SYMBOL_TABLE_POSITION = symbolTablePosition;
        SYMBOL_TABLE_SIZE = symbolTableSize;
        SYMBOL_TABLE_SECTION_COUNT = SYMBOL_TABLE_SIZE /
SYMBOL_TABLE_SECTION_SIZE;

        if (textPosition == 0) {
            throw new AssertionError("Text not found");
        }
        TEXT_POSITION = textPosition;
        TEXT_SIZE = textSize;
        COMMAND_COUNT = TEXT_SIZE / 4;
        TEXT_VIRTUAL_ADDRESS = textVirtualAddress;

        if (symtabStringTablePosition == 0) {
            throw new AssertionError("String table for symbol table not found");
        }
        SYMTAB_STRING_TABLE_POSITION = symtabStringTablePosition;
        SYMTAB_STRING_TABLE_SIZE = symtabStringTableSize;
        List<Integer> symtabStringTable =
file.subList(SYMTAB_STRING_TABLE_POSITION,
                SYMTAB_STRING_TABLE_POSITION + SYMTAB_STRING_TABLE_SIZE);
```

```
        SymtabSegment[] symbolTableSegments = new
SymtabSegment[SYMBOL_TABLE_SECTION_COUNT];
        for (int i = 0; i < SYMBOL_TABLE_SECTION_COUNT; i++) {
            symbolTableSegments[i] = new SymtabSegment(
                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE,
                            SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 3),

                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 4,
                            SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 4 + 3),

                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 8,
                            SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 8 + 3),

                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 12),

                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 13),

                    getByte(SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 14,
                            SYMBOL_TABLE_POSITION + i *
SYMBOL_TABLE_SECTION_SIZE + 14 + 1),

                    symtabStringTable
            );
        }
        symbolTable = new SymbolTable(symbolTableSegments);

        for (int i = 0; i < COMMAND_COUNT; i++) {
            getCommandString(i);
        }
    }

    public int getByte(int pos) {
        return file.get(pos);
    }

    public int getByte(int start, int end) {
        int result = 0;
        for (int i = end; i >= start; i--) {
            result = (result << 8) + file.get(i);
        }
        return result;
    }

    public int getSectionHeaderBytes(int section, int start, int end) {
        start += SECTION_HEADER_TABLE_POSITION + section *
SECTION_HEADER_SEGMENT_SIZE;
        end += SECTION_HEADER_TABLE_POSITION + section *
SECTION_HEADER_SEGMENT_SIZE;
        return getByte(start, end);
    }
```

```
    public int getSectionNamePosition(int section) {
        return getSectionHeaderBytes(section, 0, 3);
    }

    public int getSectionOffset(int section) {
        return getSectionHeaderBytes(section, 0x10, 0x10 + 4);
    }

    public int getSectionSize(int section) {
        return getSectionHeaderBytes(section, 0x14, 0x14 + 4);
    }

    public int getSectionVirtualAddress(int section) {
        return getSectionHeaderBytes(section, 0x0c, 0x0c + 3);
    }

    public int getStringTableByte(int pos) {
        return getByte(STRING_TABLE_POSITION + pos);
    }

    public String getSectionName(int sectionPosition) {
        StringBuilder result = new StringBuilder();
        int stringTablePos = getSectionNamePosition(sectionPosition);
        while (getStringTableByte(stringTablePos) != 0) {
            result.append((char) getStringTableByte(stringTablePos++));
        }
        return result.toString();
    }

    public int getCommand(int number) {
        return getByte(number * 4 + TEXT_POSITION, number * 4 + TEXT_POSITION +
3);
    }

    public int getOpcode(int command) {
        return command & 0b1111111;
    }

    public int getFunct3(int command) {
        return (command >> 12) & 0b111;
    }

    public int getFunct7(int command) {
        return command >> 25;
    }

    public String getRegisterName(int reg) {
        return switch (reg) {
            case 0 -> "zero";
            case 1 -> "ra";
            case 2 -> "sp";
            case 3 -> "gp";
            case 4 -> "tp";
            case 5 -> "t0";
            case 6 -> "t1";
            case 7 -> "t2";
            case 8 -> "s0";
            case 9 -> "s1";
```

```
            case 10 -> "a0";
            case 11 -> "a1";
            case 12 -> "a2";
            case 13 -> "a3";
            case 14 -> "a4";
            case 15 -> "a5";
            case 16 -> "a6";
            case 17 -> "a7";
            case 18 -> "s2";
            case 19 -> "s3";
            case 20 -> "s4";
            case 21 -> "s5";
            case 22 -> "s6";
            case 23 -> "s7";
            case 24 -> "s8";
            case 25 -> "s9";
            case 26 -> "s10";
            case 27 -> "s11";
            case 28 -> "t3";
            case 29 -> "t4";
            case 30 -> "t5";
            case 31 -> "t6";
            default -> throw new UnsupportedOperationException("Unsupported
register: " + "\"" + reg + "\"");
        };
    }

    public int setBits(int number, int bits, int begin, int end) {
        return number % (1 << begin) + ((number >> end) << end) + (bits <<
begin);
    }

    public int getBits(int number, int begin, int end) {
        int result = 0;
        for (int i = begin; i <= end; i++) {
            if (((number >> i) & 1) == 1) {
                result |= (1 << i);
            }
        }
        return result >> begin;
    }

    public String getSymbolTableString() {
        return symbolTable.toString();
    }

    public String getAddressLabel(int address) {
        return symbolTable.getAddressLabel(address);
    }

    public String getAddressName(int address) {
        return symbolTable.getAddressName(address);
    }

    public String getCommandString(int number) {
        int addr = TEXT_VIRTUAL_ADDRESS + number * 4;
        int command = getCommand(number);
        StringBuilder parameters = new StringBuilder();
        String commandName = "";
```

```
        switch (getOpcode(command)) {
            case 0b0110111 -> {
                commandName = "lui";
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");

parameters.append("0x").append(Integer.toHexString(getBits(command, 12, 31)));
            }
            case 0b0010111 -> {
                commandName = "auipc";
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");

parameters.append("0x").append(Integer.toHexString(getBits(command, 12, 31)));
            }
            case 0b1101111 -> {
                commandName = "jal";
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");
                int current = setBits(0, getBits(command, 12, 19), 12, 19);
                current = setBits(current, getBits(command, 20, 20), 11, 11);
                current = setBits(current, getBits(command, 21, 30), 1, 10);
                current = setBits(current, getBits(command, 31, 31), 20, 20);
                parameters.append("0x").append(Integer.toHexString(addr +
current));
                parameters.append(" <").append(getAddressLabel(addr +
current)).append(">");
            }
            case 0b1100111 -> {
                commandName = "jalr";
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");
                parameters.append(getBits(command, 20, 31));
                parameters.append("(");
                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(")");
            }
            case 0b1100011 -> {
                switch (getFunct3(command)) {
                    case 0b000 -> commandName = "beq";
                    case 0b001 -> commandName = "bne";
                    case 0b100 -> commandName = "blt";
                    case 0b101 -> commandName = "bge";
                    case 0b110 -> commandName = "bltu";
                    case 0b111 -> commandName = "bgeu";
                    default -> commandName = "unknown_instruction";
                }

                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(",");
                parameters.append(getRegisterName(getBits(command, 20, 24)));
                parameters.append(",");
                int current = setBits(0, getBits(command, 7, 7), 11, 11);
                current = setBits(current, getBits(command, 8, 11), 1, 4);
                current = setBits(current, getBits(command, 25, 30), 5, 10);
                current = setBits(current, getBits(command, 31, 31), 12, 12);
                parameters.append("0x").append(Integer.toHexString(addr +
current));
                parameters.append(" <").append(getAddressLabel(addr +
```

```
current)).append(">");
            }
            case 0b0000011 -> {
                switch (getFunct3(command)) {
                    case 0b000 -> commandName = "lb";
                    case 0b001 -> commandName = "lh";
                    case 0b010 -> commandName = "lw";
                    case 0b100 -> commandName = "lbu";
                    case 0b101 -> commandName = "lhu";
                    default -> commandName = "unknown_instruction";
                }
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");
                parameters.append(getBits(command, 20, 31));
                parameters.append("(");
                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(")");
            }
            case 0b0100011 -> {
                switch (getFunct3(command)) {
                    case 0b000 -> commandName = "sb";
                    case 0b001 -> commandName = "sh";
                    case 0b010 -> commandName = "sw";
                    default -> commandName = "unknown_instruction";
                }
                int current = getBits(command, 7, 11);
                current = setBits(current, getBits(command, 25, 31), 5, 11);
                parameters.append(getRegisterName(getBits(command, 20, 24)));
                parameters.append(",");
                parameters.append(current);
                parameters.append("(");
                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(")");
            }
            case 0b0010011 -> {
                switch (getFunct3(command)) {
                    case 0b000 -> commandName = "addi";
                    case 0b010 -> commandName = "slti";
                    case 0b011 -> commandName = "sltiu";
                    case 0b100 -> commandName = "xori";
                    case 0b110 -> commandName = "ori";
                    case 0b111 -> commandName = "andi";
                    case 0b001 -> commandName = "slli";
                    case 0b101 -> {
                        switch (getFunct7(command)) {
                            case 0b0000000 -> commandName = "srli";
                            case 0b0100000 -> commandName = "srai";
                            default -> commandName = "unknown_instruction";
                        }
                    }
                    default -> commandName = "unknown_instruction";
                }
                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");
                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(",");
                parameters.append(getBits(command, 20, 31));
            }
            case 0b0110011 -> {
```

```
                switch (getFunct7(command)) {
                    case 0b0000000 -> {
                        switch (getFunct3(command)) {
                            case 0b000 -> commandName = "add";
                            case 0b001 -> commandName = "sll";
                            case 0b010 -> commandName = "slt";
                            case 0b011 -> commandName = "sltu";
                            case 0b100 -> commandName = "xor";
                            case 0b101 -> commandName = "srl";
                            case 0b110 -> commandName = "or";
                            case 0b111 -> commandName = "and";
                        }
                    }
                    case 0b0100000 -> {
                        switch (getFunct3(command)) {
                            case 0b000 -> commandName = "sub";
                            case 0b101 -> commandName = "sra";
                        }
                    }
                    case 0b0000001 -> {
                        switch (getFunct3(command)) {
                            case 0b000 -> commandName = "mul";
                            case 0b001 -> commandName = "mulh";
                            case 0b010 -> commandName = "mulhsu";
                            case 0b011 -> commandName = "mulhu";
                            case 0b100 -> commandName = "div";
                            case 0b101 -> commandName = "divu";
                            case 0b110 -> commandName = "rem";
                            case 0b111 -> commandName = "remu";
                        }
                    }
                    default -> commandName = "unknown_instruction";
                }

                parameters.append(getRegisterName(getBits(command, 7, 11)));
                parameters.append(",");
                parameters.append(getRegisterName(getBits(command, 15, 19)));
                parameters.append(",");
                parameters.append(getRegisterName(getBits(command, 20, 24)));
            }
            case 0b1110011 -> {
                switch (command >> 20) {
                    case 0b000000000000 -> commandName = "ecall";
                    case 0b000000000001 -> commandName = "ebreak";
                    default -> commandName = "unknown_instruction";
                }
            }
            case 0b0001111 -> commandName = "fence";
            default -> commandName = "unknown_instruction";
        }

        return String.format("    %05x:   %08x      %5s %s", addr, command,
commandName, parameters);
    }
}
```

SymbolTable.java

```
import java.util.HashMap;
import java.util.Map;

public class SymbolTable {
    private final SymtabSegment[] symbolTable;
    private final Map<Integer, String> labelAddress;
    private int lastLabel = 0;
    public SymbolTable(SymtabSegment[] symbolTable) {
        this.symbolTable = symbolTable;

        labelAddress = new HashMap<>();
        for (SymtabSegment symtabSegment : symbolTable) {
            if (symtabSegment.getType() == 2) {
                labelAddress.put(symtabSegment.getValue(),
symtabSegment.getStringName());
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        result.append("Symbol Value              Size Type     Bind    Vis
Index Name\n");
        for (int i = 0; i < symbolTable.length; i++) {
            result.append(String.format("[%4d] 0x%-15X %5d %-8s %-8s %-
8s %6s %s\n",
                    i,
                    symbolTable[i].getValue(),
                    symbolTable[i].getSize(),
                    symbolTable[i].getStringType(),
                    symbolTable[i].getStringBind(),
                    symbolTable[i].getStringVisibility(),
                    symbolTable[i].getStringShndx(),
                    symbolTable[i].getStringName()
            ));
        }
        return result.toString();
    }

    public String getAddressLabel(int address) {
        if (!labelAddress.containsKey(address)) {
            labelAddress.put(address, "L" + lastLabel++);
        }
        return labelAddress.get(address);
    }

    public String getAddressName(int address) {
        return labelAddress.getOrDefault(address, "");
    }
}
```

SymtabSegment.java

```
import java.util.List;

public class SymtabSegment {
    private final String stringName;
    private final int name;
    private final int value;
    private final int size;
    private final int info;
    private final int other;
    private final int shndx;
    private final int type;
    private final int bind;
    private final int visibility;

    public SymtabSegment(int name, int value, int size, int info, int other, int
shndx, List<Integer> stringTable) {
        this.name = name;
        this.value = value;
        this.size = size;
        this.info = info;
        this.other = other;
        this.shndx = shndx;
        this.stringName = findStringName(stringTable);

        this.bind = this.info >> 4;
        this.type = this.info & 0xf;
        this.visibility = this.other & 0x3;
    }

    public int getName() {
        return name;
    }

    public int getValue() {
        return value;
    }

    public int getSize() {
        return size;
    }

    public int getInfo() {
        return info;
    }

    public int getOther() {
        return other;
    }

    public int getShndx() {
        return shndx;
    }

    public int getType() {
        return type;
    }

    public int getBind() {
```

```
        return bind;
    }

    public int getVisibility() {
        return visibility;
    }

    public String findStringName(List<Integer> stringTable) {
        int pos = name;
        StringBuilder result = new StringBuilder();
        while (stringTable.get(pos) != 0) {
            result.append((char)(int)stringTable.get(pos++));
        }
        return result.toString();
    }

    public String getStringType() {
        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 -> {
                throw new UnsupportedOperationException("Unsupported symtab
segment type");
            }
        };
    }

    public String getStringBind() {
        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 -> {
                throw new UnsupportedOperationException("Unsupported symtab
segment bind");
            }
        };
    }

    public String getStringVisibility() {
        return switch(visibility) {
            case 0 -> "DEFAULT";
            case 1 -> "INTERNAL";
            case 2 -> "HIDDEN";
            case 3 -> "PROTECTED";
            case 4 -> "EXPORTED";
```

```
            case 5 -> "SINGLETON";
            case 6 -> "ELIMINATE";
            default -> {
                throw new UnsupportedOperationException("Unsupported symtab
segment visibility");
            }
        };
    }

    public String getStringShndx() {
        return switch(shndx) {
            case 0 -> "UNDEF";
            case 0xff00 -> "LORESERVE";
            case 0xff01 -> "AFTER";
            case 0xff02 -> "AMD64_LCOMMON";
            case 0xff1f -> "HIPROC";
            case 0xff20 -> "LOOS";
            case 0xff3f -> "HIOS";
            case 0xfff1 -> "ABS";
            case 0xfff2 -> "COMMON";
            case 0xffff -> "XINDEX";
            default -> Integer.toString(shndx);
        };
    }

    public String getStringName() {
        return stringName;
    }
}
```