Лабораторная работа №4
ISA

Халили Алина Ниязовна, M3136

https://github.com/skkv-itmo2/itmo-comp-arch-2023-riscv-AlinaK12345/actions/workflows/ci.yaml

Язык: Java 19

Необходимо декодировать команды из наборов RV32I и RV32M.

Все команды даны в 32х битном формате; little endian

1) Сначала необходимо считать входной ELF файл и найти там информацию про .text (там хранятся команды, которые поданы к исполнению), .strtab (там хранятся все string в файле, в том числе и названия секций) и .symtab (описание команд, из метки и другая информация).
   ELF – файл:

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

ELF - OSDev Wiki

Из таблицы:

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

Храню данные как List<Integer> - 1 число – 1 байт.

Мы проходимся по секциям, и ищем нужные. Каждая секция содержит информацию, согласно данной структуре:

<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 colspan="2">4</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 colspan="2">4</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 colspan="2">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 colspan="2">4</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 colspan="2">4</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>

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

Интересуют:

- sh_name - Первые 4 байта – название (.text || .symtab …)
- sh_addr - 0x10 4байта – адрес начала самой информации для этой секции (например команды для .text)
- sh_size – размер секции

Размер каждой секции – 40 байт

Считывание данных для обработки каждой секции:

```
private static void checkAddressStr(int currentPos){
        int address = makeBitNumber(currentPos, 4) + SECTION_HEADER_NAMES;
// индекс где лежит значение .text или другие
        int a = elf.get(address);
        StringBuilder str = new StringBuilder();
        int i = 0;
        while (a != 0){
            str.append((char) a);
            i++;
            a = elf.get(address + i);
        }

      if (str.toString().equals(".text")){
                  TEXT_START_VIRTUAL_ADDRESS = makeBitNumber(currentPos + 0x0C, 4);
                  TEXT_ADDRESS = makeBitNumber(currentPos + 0x10, 4); // адрес на
      данные
                  LEN_TEXT_SECTION = makeBitNumber(currentPos + 0x14, 4); //
      количество секций
      }
      else if (str.toString().equals(".symtab")){
                  SYMTABLE_ADDRESS = makeBitNumber(currentPos + 0x10, 4);
                  LEN_SYMTABLE_SECTION = makeBitNumber(currentPos + 0x14, 4);
      }
      else if (str.toString().equals(".strtab")) {
                  SYMBOL_TABLE_STR_ADDRESS = makeBitNumber(currentPos + 0x10, 4);
      }
}
```

2)Обработка данных Symtab

Данные этой секции выглядят так:

```
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;
```

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

Значения bind и type можно получить из info:

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

Значение visibility можно получить из поля other:

- visibility = other & 0x3

Интерпретация значений:

(https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter679797.html)

Bind:

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

Types:

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

Visibly:

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

Shindx:

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

3)Обработка данных .text

Данные представляют собой набор команд по 32 бита.

https://github.com/riscv/riscv-isa-manual/releases/download/riscv-isa-release-056b6ff-2023-10-02/unpriv-isa-asciidoc.pdf

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

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
1000	0011	0011	00000	000	00000	0001111	FENCE.TSO
0000	0001	0000	00000	000	00000	0001111	PAUSE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

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

Обработка регистров команд(rs1, rs2, rd)

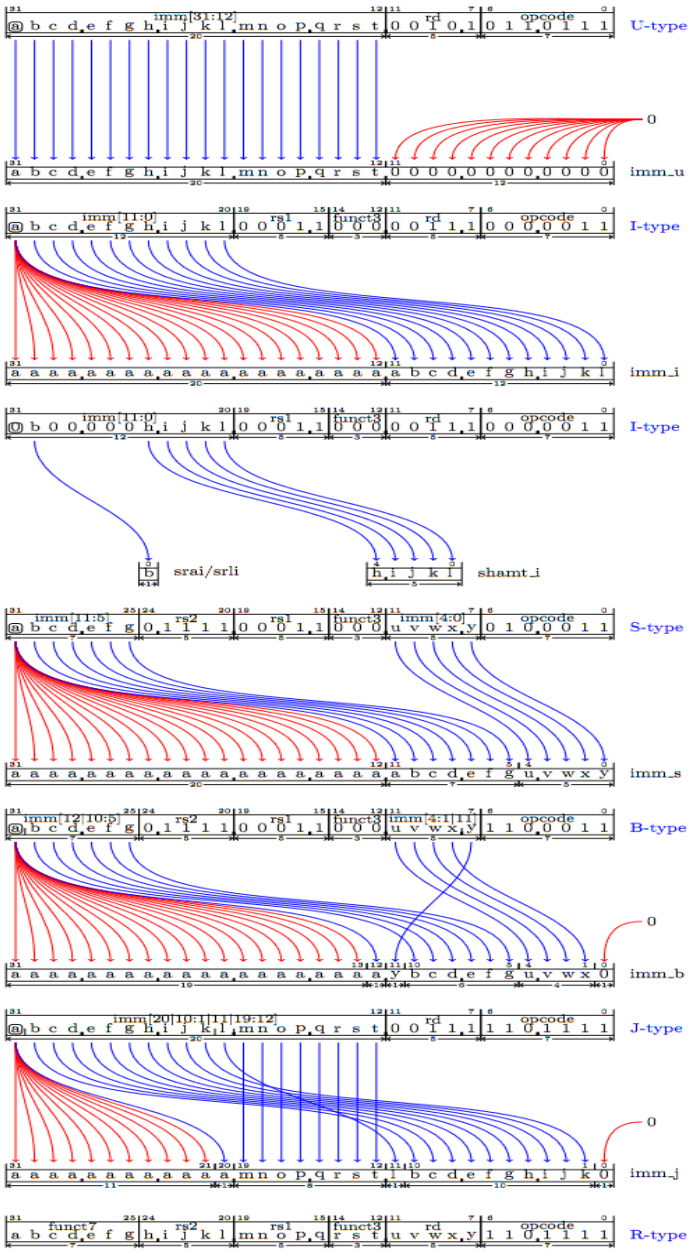
https://github.com/riscv-non-isa/riscv-elf-psabi-doc/blob/master/riscv-cc.adoc

Name	ABI Mnemonic	Meaning	Preserved across calls?
x0	zero	Zero	— (Immutable)
x1	ra	Return address	No

Name	ABI Mnemonic	Meaning	Preserved across calls?
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	— (Unallocatable)
x4	tp	Thread pointer	— (Unallocatable)
x5 - x7	t0 - t2	Temporary registers	No
x8 - x9	s0 - s1	Callee-saved registers	Yes
x10 - x17	a0 - a7	Argument registers	No
x18 - x27	s2 - s11	Callee-saved registers	Yes
x28 - x31	t3 - t6	Temporary registers	No

Обработка констант(immanuate)

(формат данных – дополнение до 2)



Таким образом я получила константы, регистры, адрес команд. Некоторые команды ставят еще lable на другие. Я храню Map<> с ними. – `readText()`

Сначала мне нужно один раз пройтись что бы расставить все метки, потом я прохожусь и вывожу .text

Затем вывожу .symtab

В работе полностью реализовано 32i, 32m:
Вот результат вывода, тесты прошли:
(так же результат сходится на всех дополнительных файлах test_elf, test2.elf)

```
.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 iorw, iorw
   100a8:       00000073         ecall

000100ac        <mmul>:
   100ac:       00011f37           lui t5, 0x11
   100b0:       124f0513          addi a0, t5, 292
   100b4:       65450513          addi a0, a0, 1620
   100b8:       124f0f13          addi t5, t5, 292
   100bc:       e4018293          addi t0, gp, -448
   100c0:       fd018f93          addi t6, gp, -48
   100c4:       02800e93          addi t4, zero, 40

000100c8        <L2>:
   100c8:       fec50e13          addi t3, a0, -20
   100cc:       000f0313          addi t1, t5, 0
   100d0:       000f8893          addi a7, t6, 0
   100d4:       00000813          addi a6, zero, 0

000100d8        <L1>:
   100d8:       00088693          addi a3, a7, 0
   100dc:       000e0793          addi a5, t3, 0
   100e0:       00000613          addi a2, zero, 0

000100e4        <L0>:
   100e4:       00078703            lb a4, 0(a5)
   100e8:       00069583            lh a1, 0(a3)
   100ec:       00178793          addi a5, a5, 1
   100f0:       02868693          addi a3, a3, 40
   100f4:       02b70733           mul a4, a4, a1
   100f8:       00e60633           add a2, a2, a4
   100fc:       fea794e3           bne a5, a0, 0x100e4, <L0>
   10100:       00c32023            sw a2, 0(t1)
   10104:       00280813          addi a6, a6, 2
```

```
   10108:       00430313          addi t1, t1, 4
   1010c:       00288893          addi a7, a7, 2
   10110:       fdd814e3           bne a6, t4, 0x100d8, <L1>
   10114:       050f0f13          addi t5, t5, 80
   10118:       01478513          addi a0, a5, 20
   1011c:       fa5f16e3           bne t5, t0, 0x100c8, <L2>
   10120:       00008067          jalr zero, 0(ra)
```

```
.symtab

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

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class Main {

    static List<Integer> elf = new ArrayList<>();
    static StringBuilder symtable = new StringBuilder("\nSymbol Value
Size Type     Bind     Vis       Index Name\n");
    static StringBuilder text = new StringBuilder();
    static HashMap<Integer, String> label= new HashMap<>();
    static int lastLabel = 0;
    private static int SECTION_HEADER_POSITION = 32; // 32-35
    private static int NUMBER_OF_SECTIONS; // 48-49
    private static int LEN_SECTION = 40; // 1 секция -  40 бит
    private static int sectionHeadPosition = 0;
    private static int SECTION_HEADER_NAMES; // 50-51
    private static int TEXT_ADDRESS;
    private static int LEN_TEXT_SECTION;
    private static int TEXT_START_VIRTUAL_ADDRESS;
    private static int LEN_SYMTABLE_SECTION;
    private static int SYMTABLE_ADDRESS;
    private static int SYMBOL_TABLE_STR_ADDRESS;

    public static void main(String[] args) {
        String filenameIn = args[0];
        String filenameOut = args[1];
        try{
            InputStream in =
                    new FileInputStream(filenameIn);
            try{
                solution(in);

            } finally {
                in.close();

            }
            OutputStreamWriter out = new OutputStreamWriter(
                    new FileOutputStream(filenameOut), "UTF-8");
            try {
                out.write(".text\n");
                out.write(text.toString());
                out.write("\n\n");
                out.write(".symtab\n");
                out.write(symtable.toString());
            } finally {
                out.close();
            }

        } catch (FileNotFoundException e){
            System.out.println("No Input/Output file");
        } catch (IOException e) {
            System.out.println("IOExeption");
        }
    }

    private static void checkIsCorrect(){
        if (elf.get(0) != 0x7f || elf.get(1) != 0x45 || elf.get(2) != 0x4c ||
elf.get(3) != 0x46) {
            throw new UnsupportedOperationException("Unsupported file
format");
        }
        if (elf.get(4) != 1) {
```

```
            throw new UnsupportedOperationException("Not 32 bits file");
        }
        if (elf.get(5) != 1) {
            throw new UnsupportedOperationException("Not little-endian
file");
        }
    }

    private static int makeBitNumber(int pos, int len){
        int ans = 0;
        for (int i = len-1; i >=0; i--){
            ans = ans << 8;
            ans += elf.get(pos+i);
        }
        return ans;
    }

    private static void checkAddressStr(int currentPos){
        int address = makeBitNumber(currentPos, 4) + SECTION_HEADER_NAMES; //
индекс где лежит значение .text или другие
        int a = elf.get(address);
        StringBuilder str = new StringBuilder();
        int i = 0;
        while (a != 0){
            str.append((char) a);
            i++;
            a = elf.get(address + i);
        }
        if (str.toString().equals(".text")){
            TEXT_START_VIRTUAL_ADDRESS = makeBitNumber(currentPos + 0x0C, 4);
            TEXT_ADDRESS = makeBitNumber(currentPos + 0x10, 4); // адрес на
данные
            LEN_TEXT_SECTION = makeBitNumber(currentPos + 0x14, 4); //
количество секций

        }
        else if (str.toString().equals(".symtab")){
            SYMTABLE_ADDRESS = makeBitNumber(currentPos + 0x10, 4);
            LEN_SYMTABLE_SECTION = makeBitNumber(currentPos + 0x14, 4);
        }
        else if (str.toString().equals(".strtab")) {
            SYMBOL_TABLE_STR_ADDRESS = makeBitNumber(currentPos + 0x10, 4);
        }
    }

    private static int getBitNumber(int number, int begin, int end){
        int ans = 0;
        int a = 0;
        for (int i = end; i <= begin; i++) {
            if (((number >> i) & 1) == 1) {
                ans |= (1 << i);
            }
        }

        return ans >> end;
    }

    private static int putBits(int number, int begin, int end, int bits){
        int ans = number % (1 << end); // конец справа
        ans += (number >> begin) << begin;
        ans += bits << end;
        return ans;
    }
```

```
    private static void readText(int i, int parametr){
        // одна команда - 32 бита - 4 байта
        //litle-endian ...? - первые 6 бит?
        int address = i + TEXT_ADDRESS; // начало команды
        int virtAddress = TEXT_START_VIRTUAL_ADDRESS + i;
        StringBuilder command = new StringBuilder();
        int number = makeBitNumber(address, 4);
        int opcode = getBitNumber(number, 6, 0);


        String name;

        int funct3 = getBitNumber(number, 14, 12);
        int funct7 = getBitNumber(number, 31, 25);

        int rs1 = getBitNumber(number, 19, 15);
        int rs2 = getBitNumber(number, 24, 20);
        int rd = getBitNumber(number, 11, 7);

        List<Integer> registrs = new ArrayList<>();
        registrs.add(rd);
        registrs.add(rs1);

        int immediate = -1;

        switch (opcode){
            case 0b0110111 -> {
                name = "lui";
                registrs.remove(1);
                immediate = getBitNumber(number, 31, 12);
            }
            case 0b0010111 -> {
                name = "auipc";
                registrs.remove(1);

                immediate = getBitNumber(number, 31, 12);
            }
            case 0b1101111 -> {
                name = "jal";
                registrs.remove(1);

                immediate = 0;
                immediate = putBits(immediate, 19, 12, getBitNumber(number,
19, 12));
                immediate = putBits(immediate, 11, 11, getBitNumber(number,
20, 20));
                immediate = putBits(immediate, 10, 1, getBitNumber(number,
30, 21));
                immediate = putBits(immediate, 20, 20, getBitNumber(number,
31, 31));
//
                immediate += virtAddress;
                makeLabel(immediate);

            }
            case 0b1100111 -> {
                name = "jalr";
                immediate = getBitNumber(number, 31, 20);

            }
            case 0b1100011 ->{
                switch (funct3){
```

```
                    case 0b000 -> name = "beq";
                    case 0b001 -> name = "bne";
                    case 0b100 -> name = "blt";
                    case 0b101 -> name = "bge";
                    case 0b110 -> name = "bltu";
                    case 0b111 -> name = "bgeu";
                    default -> name = "invalid_instruction";
                }
                registrs.remove(0);
                registrs.add(rs2);

                immediate = 0;
                immediate = putBits(immediate, 12, 12, getBitNumber(number,
31, 31));
                immediate = putBits(immediate, 11, 11, getBitNumber(number,
7, 7));
                immediate = putBits(immediate, 10, 5, getBitNumber(number,
30, 25));
                immediate = putBits(immediate, 4, 1, getBitNumber(number, 11,
8));


                immediate += virtAddress;
                makeLabel(immediate);
            }
            case 0b0000011 ->{
                switch (funct3){
                    case 0b000 -> name = "lb";
                    case 0b001 -> name = "lh";
                    case 0b010 -> name = "lw";
                    case 0b100 -> name = "lbu";
                    case 0b101 -> name = "lhu";
                    default -> name = "invalid_instruction";
                }
//
                immediate = getBitNumber(number, 31, 20);
            }
            case 0b0100011 -> {
                switch (funct3){
                    case 0b000 -> name = "sb";
                    case 0b001 -> name = "sh";
                    case 0b010 -> name = "sw";
                    default -> name = "invalid_instruction";
                }
                registrs.set(0, rs2);
                immediate = getBitNumber(number, 31, 25);
                immediate = immediate << 5;
                immediate += getBitNumber(number, 11, 7);
            }
            case 0b0010011 -> {
                switch (funct3) {
                    case 0b000 -> name = "addi";
                    case 0b010 -> name = "slti";
                    case 0b011 -> name = "sltiu";
                    case 0b100 -> name = "xori";
                    case 0b110 -> name = "ori";
                    case 0b111 -> name = "andi";
                    case 0b001 -> {
                        name = "slli";
                        int shamt = getBitNumber(number, 24,20);

                    }
                    case 0b101 -> {
                        switch (funct7) {
```

```
                    case 0b0000000 -> name = "srli";
                    case 0b0100000 -> name = "srai";
                    default -> name = "invalid_instruction";
                }
                int shamt = getBitNumber(number, 24,20);
            }
            default -> name = "invalid_instruction";
        }
        immediate = getBitNumber(number, 31, 20);
       if (funct3 ==0b101 ){
           immediate = getBitNumber(number, 24,20); // shamt
       }

    }
    case 0b0110011 -> {

        if (funct7 == 0b0000001){    /// RV32M
            switch (funct3) {
                case 0b000 -> name = "mul";
                case 0b001 -> name = "mulh";
                case 0b010 -> name = "mulhsu";
                case 0b011 -> name = "mulhu";
                case 0b100 -> name = "div";
                case 0b101 -> name = "divu";
                case 0b110 -> name = "rem";
                case 0b111 -> name = "remu";
                default -> name = "invalid_instruction";
            }
        }
        else {
            switch (funct3) {
                case 0b000 -> {
                    switch (funct7) {
                        case 0b0000000 -> name = "add";
                        case 0b0100000 -> name = "sub";
                        default -> name = "invalid_instruction";
                    }
                }
                case 0b001 -> name = "sll";
                case 0b010 -> name = "slt";
                case 0b011 -> name = "sltu";
                case 0b100 -> name = "xor";
                case 0b101 -> {
                    switch (funct7) {
                        case 0b0000000 -> name = "srl";
                        case 0b0100000 -> name = "sra";
                        default -> name = "invalid_instruction";
                    }
                }
                case 0b110 -> name = "or";
                case 0b111 -> name = "and";
                default -> name = "invalid_instruction";
            }
        }
        immediate = -1;
        registrs.add(rs2);
    }
    case 0b0001111 -> {
        switch (getBitNumber(number, 31, 28)) {
            case 0b1000 -> {
                name = "fence.tso";
                registrs.remove(1);
                registrs.remove(0);
            }
```

```
                    case 0b0000 -> {
                        name = "fence";
                        registrs.remove(1);
                        registrs.remove(0);
                    }
                    default -> name = "invalid_instruction";
                }
                immediate = -1;
            }
            case 0b1110011 -> {
                switch (getBitNumber(number, 31, 20)){
                    case 0b000000000000 -> {
                        name = "ecall";
                        registrs.remove(1);
                        registrs.remove(0);
                    }
                    case 0b000000000001 -> {
                        name = "ebreak";
                        registrs.remove(1);
                        registrs.remove(0);
                    }
                    default -> name = "invalid_instruction";
                }
                immediate = -1;
            }
            default ->  name = "invalid_instruction";
        }
        if (parametr == 1) {
            makeStringText(virtAddress, number, name, registrs, immediate);
        }
    }

    private static void makeLabel(int virtAddr){
        if (!label.containsKey(virtAddr)){
            label.put(virtAddr, "L" + lastLabel);
            lastLabel++;
        }
    }

    private static void makeStringText(int virtAddr, int number, String name,
List<Integer> registrs, int immediate){ ////////лучше сразу записывать в out
- file
        if (name.equals("invalid_instruction")){
            text.append(String.format("   %05x:\t%08x\t%-7s\n", virtAddr,
number, name)); // + аргументы ////////////////////
        }
        else{
            if (label.containsKey(virtAddr)) {
                text.append(String.format("\n%08x \t<%s>:\n", virtAddr,
label.get(virtAddr)));
                if (label.get(virtAddr).charAt(0) != 'L'){
                }
            }
            if (name.equals("fence")){
                text.append("fence\tiorw, iorw");
            }
            if (name.equals("jal")){
                text.append(String.format("   %05x:\t%08x\t%7s\t%s, 0x%x
<%s>\n", virtAddr,number,
                        name, registerName(registrs.get(0)), immediate,
label.get(immediate)));
            }
            else if (name.equals("jalr") || name.length() == 2
&&(name.charAt(0) == 's' || name.charAt(0) == 'l') || name.equals("lbu") ||
```

```
name.equals("lhu")){
                text.append(String.format("   %05x:\t%08x\t%7s\t%s,
%d(%s)\n",
                        virtAddr, number, name,
registerName(registrs.get(0)), immediate, registerName(registrs.get(1))));
            }
            else if ( name.charAt(0) == 'b'){
                text.append(String.format("   %05x:\t%08x\t%7s\t%s, %s, 0x%x,
<%s>\n", virtAddr,number,
                        name,
registerName(registrs.get(0)),registerName(registrs.get(1)), immediate,
label.get(immediate)));
            }
            else if (registrs.size() == 3){
                text.append(String.format("   %05x:\t%08x\t%7s\t%s, %s,
%s\n",
                        virtAddr, number, name,
registerName(registrs.get(0)), registerName(registrs.get(1)),
registerName(registrs.get(2))));
            }
            else if (registrs.size() == 2){
                text.append(String.format("   %05x:\t%08x\t%7s\t%s, %s,
%s\n",
                        virtAddr, number, name,
registerName(registrs.get(0)),registerName(registrs.get(1)), immediate));
            }
            else if (registrs.size() == 1){
                if (name.equals("lui") || name.equals("auipc")){
                    text.append(String.format("   %05x:\t%08x\t%7s\t%s,
%s\n",
                            virtAddr, number, name,
registerName(registrs.get(0)), "0x" + Integer.toHexString(immediate)));
                }
                else {
                    text.append(String.format("   %05x:\t%08x\t%7s\t%s,
%s\n",
                            virtAddr, number, name,
registerName(registrs.get(0)), immediate));
                }
            }
            else if (registrs.size() == 0){
                text.append(String.format("   %05x:\t%08x\t%7s\n",
                        virtAddr, number, name));
            }
        }
    }



    private static String registerName(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("not correct
register");
        };
    }

    private static void readSymtable(int i){
        int address = SYMTABLE_ADDRESS + i;

        int st_name = makeBitNumber(address, 4);
        int st_value = makeBitNumber(address + 4, 4);
        int st_size = makeBitNumber(address + 8, 4);
        int st_info = makeBitNumber(address + 12, 1);
        int st_other = makeBitNumber(address + 13, 1);
        int st_shndx = makeBitNumber(address + 14, 2);

        address = SYMBOL_TABLE_STR_ADDRESS + st_name; // типа это сдвиг
        int a = elf.get(address);
        StringBuilder str = new StringBuilder();
        int j = 0;
        while (a != 0){ //
            str.append((char) a);
            j++;
            a = elf.get(address + j);
        }
        String name = str.toString();

        int value = st_value; // виртуальный адрес функции;
        int bind = ((st_info) >> 4);
        int type = ((st_info) & 0xf);
        int vis = st_other & 0x3;

        if (!name.isEmpty()) { /////////// типа что строка не поустая
            label.put(value, name);
        }
        makeStringSymtable(i, value, st_size, type, bind, vis, st_shndx,
name);

    }

    private static void makeStringSymtable(int i, int value, int size, int
type, int bind, int vis, int index, String name){
        String line = String.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s
%s\n",i/16, value,
                size, getType(type), getBind(bind),
                getVisiable(vis), getIndex(index), name);
```

```
            symtable.append(line);
        }



        private static String getIndex(int ind){
            return switch (ind) {
                case 0 -> "UNDEF";
                case 0xff00 -> "LORESERVE"; //
                case 0xff01 -> "AFTER";
                case 0xff02 -> "AMD64_LCOMMON";
                case 0xff1f -> "HIPROC";
                case 0xff20 -> "LOOS";
                case 0xff3f -> "LOSUNW"; //
                case 0xfff1 -> "ABS";
                case 0xfff2 -> "COMMON";
                case 0xffff -> "XINDEX"; //
                default -> Integer.toString(ind);
            };
        }

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

        private static String getType(int type){
            return switch(type) {
                case 0 -> "NOTYPE";
                case 1 -> "OBJECT";
                case 2 -> "FUNC";
                case 3 -> "SECTION";
                case 4 -> "FILE";
                case 5 -> "COMMON";
                case 6 -> "TLS";
                case 10 -> "LOOS";
                case 12 -> "HIOS";
                case 13 -> "LOPROC";
                case 15 -> "HIPROC";
                default -> {
                    throw new UnsupportedOperationException();
                }
            };
        }
        private static String getBind(int bind){
            return switch(bind) {
                case 0 -> "LOCAL";
                case 1 -> "GLOBAL";
                case 2 -> "WEAK";
                case 10 -> "LOOS";
                case 12 -> "HIOS";
                case 13 -> "LOPROC";
                case 15 -> "HIPROC";
```

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


    private static void solution(InputStream in) throws IOException {
        int read = in.read();
        while (read != -1) {
            elf.add(read);
            read = in.read();
        }
        checkIsCorrect();

        sectionHeadPosition = makeBitNumber(SECTION_HEADER_POSITION, 4); //
адрес начала секций
        NUMBER_OF_SECTIONS = makeBitNumber(48, 2);
        SECTION_HEADER_NAMES = makeBitNumber(sectionHeadPosition +
LEN_SECTION * makeBitNumber(50, 2) + 0x10, 4); // индекс начала всех имен
        int addressStr; // адрес на .text || .???

        int currentPos = sectionHeadPosition;
        for (int i = 0; i < NUMBER_OF_SECTIONS; i++) { // пробегаемся по
секциям и ищем .text, .symtab
            checkAddressStr(currentPos);// currentPos - индекс, указывающий
на начало текущ. секции
            currentPos += LEN_SECTION;

        }
        // мы нашли указатели на нужные данные

        for (int i = 0; i < LEN_SYMTABLE_SECTION; i += 16) { // 16 байт одна
строка (32 + 32 + 32 + 8 + 8 + 32/2)
            readSymtable(i);
        }

        for (int i = 0; i < LEN_TEXT_SECTION; i += 4) { // одна команда - 32
бита - 4 байта
            //litle-endian ...? - первые 6 бит?
            readText(i, 0);
        }
        for (int i = 0; i < LEN_TEXT_SECTION; i += 4) { // одна команда - 32
бита - 4 байта
            //litle-endian ...? - первые 6 бит?
            readText(i, 1);
        }

    }



}
```

Источники:

1) ELF - OSDev Wiki
2) https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
3) https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html
4) https://github.com/riscv/riscv-isa-manual/releases/download/riscv-isa-release-056b6ff-2023-10-02/unpriv-isa-asciidoc.pdf