Лабораторная работа №3	M3137	2022
ISA	Булавко Тимофей Евгеньевич	

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

Инструментарий: работа выполнена на языке высокого уровня Java 17.

RISC-V

RISC-V — расширяемая открытая и свободная система команд и процессорная архитектура на основе концепции RISC для микропроцессоров и микроконтроллеров.

Базовая спецификация «RV32I» (RV — risc5, 32-разрядная, I означает наличие целочисленной арифметики) включает 39 целочисленных инструкций. Используется 6 типов кодирования инструкций (форматов).

Расширения:

- M — целочисленное умножение/деление
- A — атомарные операции с памятью
- F и D — вычисления с плавающей точкой float, double
- C — сжатый формат команд, для удвоения плотности упаковки в машинном слове наиболее востребованных стандартных инструкций
- E — встраиваемые системы (например, только 16 регистров для маломощных процессоров loT)

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

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).

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

Инструкции базового набора имеют длину 32 бита с выравниванием на границу 32-битного слова, но в общем формате предусмотрены инструкции различной длины

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.

Операции умножения, деления и вычисления остатка не входят в минимальный набор инструкций, а выделены в отдельное расширение (M — Multiply extension). Имеется ряд доводов в пользу разделения и данного набора на два отдельных (умножение и деление).

Однозначная декодировка инструкций производится за счет сравнения кода операции, и особых разделов битов предназначенных для этого funct3 и funct7.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

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

Elf файл

ELF — это сокращение от Executable and Linkable Format (формат исполняемых и связываемых файлов) и определяет структуру бинарных файлов, библиотек, и файлов ядра (core files). Спецификация формата позволяет операционной системе корректно интерпретировать содержащиеся в файле машинные команды. Файл ELF, как правило, является выходным файлом компилятора или линкера и имеет двоичный формат. С помощью подходящих инструментов он может быть проанализирован и изучен.

ELF файл состоит из следующих частей: заголовка, секций, таблицы заголовков программы, сегментов.

- заголовок файла (ELF Header) имеет фиксированное расположение в начале файла и содержит общее описание структуры файла и его основные характеристики, такие как: тип, версия формата, архитектура процессора, виртуальный адрес точки входа, размеры и смещения остальных частей файла. Заголовок имеет размер 52 байта для 32-битных файлов или 64 для 64-битных. Данное различие обуславливается тем, что в заголовке файла содержится три поля, имеющих размер указателя, который составляет 4 и 8 байт для 32- и 64-битных процессоров соответственно.
- Информация, хранящаяся в ELF-файле, организована в секции. Каждая секция имеет свое уникальное имя. Некоторые секции хранят служебную информацию ELF-файла (например, таблицы строк), другие секции хранят отладочную информацию, третьи секции хранят код или данные программы.
  Для декодирования нам требуется 3 секции:
- .text - Хранятся исполняемые инструкции
- .symtab - Таблица символов объектного файла содержит информацию, необходимую для поиска и перемещения символьных определений и ссылок программы.
- .strtab - Разделы таблицы строк содержат последовательности символов, заканчивающиеся нулем, обычно называемые строками. Объектный файл использует эти строки для представления имен символов и разделов. Один ссылается на строку как индекс в разделе таблицы строк.

ElfParser

Программа запускается в классе Main. Входной elf файл подается как args[0], файл с результатом работы программы – args[1]. Основной код программы находится в классе ElfParser. При его инициализации из elf файла достаются его основные характеристики и сам файл переводится в байты. Пасинг файла начинается при вызове метода parse. В этой части кода файл разбивается на .text и .symtab и выводится результат лействия программы, после .text построчно парсится в parseCommand, а для каждой строчки .symtab создается в SymtabLine. В этом классе после его инициализации, по основным характеристикам происходит парсинг строки. В вспомогательных методах toBitString происходит перевод числа в битовую строку, и в get получение байтов из elf файла.

Результат работы программы

```
.text
00010074   <main>:
   10074:   ff010113        addi      sp, sp, -16
   10078:   00112623          sw ra, 12(sp)
   1007c:   030000ef         jal      ra, L8
   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, 1048576
   10098:   fddff0ef         jal      ra, L8
   1009c:   00050593        addi      a1, a0, 0
   100a0:   00a00893        addi      a7, zero, 10
   100a4:   0ff0000f       fence
   100a8:   00000073       ecall
000100ac   <mmul>:
   100ac:   00011f37         lui      t5, 69632
   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
   100c8:   fec50e13        addi      t3, a0, -20
   100cc:   000f0313        addi      t1, t5, 0
   100d0:   000f8893        addi      a7, t6, 0
   100d4:   00000813        addi      a6, zero, 0
   100d8:   00088693        addi      a3, a7, 0
   100dc:   000e0793        addi      a5, t3, 0
   100e0:   00000613        addi      a2, zero, 0
   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, L8
   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, L8
   10114:   050f0f13        addi      t5, t5, 80
   10118:   01478513        addi      a0, a5, 20
   1011c:   fa5f16e3         bne      t5, t0, L8
   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
```

Источники

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

https://marsohod.org/projects/proekty-dlya-platy-marsokhod3/risc-v/414-risc-v-begin https://ru.wikipedia.org/wiki/RISC-V

http://db4.sbras.ru/elbib/data/show_page.phtml?20+740

https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html#visibility

https://ejudge.ru/study/3sem/elf.html

https://wiki.osdev.org/ELF_Tutorial

https://docs.oracle.com/cd/E19455-01/816-0559/6m71o2afs/index.html#chapter6-94076

Листинг кода

ElfParser.java

```
import java.io.*;
import java.nio.charset.*;
import java.util.*;
import static java.lang.Integer.*;

public class ElfParser {
    BufferedWriter writer;
    HashMap<Integer, SymtabLine> dictionary = new HashMap<>();
    List<SymtabLine> symtab = new ArrayList<>();
    private int[] elf;
    private int osS, osT, arT, offsT, szT, offsS, szS;
    public ElfParser(String[] args) throws Exception {
        String exc = "Incorrect arguments";
        if (args.length != 2) {
            throw new Exception(exc);
        }
        if (args[0] == null) {
            throw new Exception(exc);
        }
        if (args[1] == null) {
            throw new Exception(exc);
        }
        try {
            byte[] temp;
            FileInputStream input = new FileInputStream(args[0]);
            temp = input.readAllBytes();
            elf = new int[temp.length];
            for (int i = 0; i < elf.length; i++) {
                elf[i] = temp[i];
                if (elf[i] < 0) {
                    elf[i] += 256;
                }
            }
            int e_shoff = get(32, 4);
            int e_shentsize = get(46, 2);
            int e_shnum = get(48, 2);
            int e_shstrndx = get(50, 2);
            osS = get(e_shoff + e_shentsize * e_shstrndx + 16, 4);
            for (int i = 0; i < e_shnum; i++) {
```

```
                final int sh_name = get(e_shoff + e_shentsize * i, 4);
                final int sh_addr = get(e_shoff + e_shentsize * i + 12, 4);
                final int sh_offset = get(e_shoff + e_shentsize * i + 16, 4);
                final int sh_size = get(e_shoff + e_shentsize * i + 20, 4);
                String name;
                if (osS + sh_name == osS) {
                    name = "";
                } else {
                    StringBuilder s = new StringBuilder();
                    int x = 0;
                    while (elf[osS + sh_name + x] != 0) {
                        s.append((char) elf[osS + sh_name + x]);
                        x++;
                    }
                    name = s.toString();
                }
                if (".text".equals(name)) {
                    arT = sh_addr;
                    offsT = sh_offset;
                    szT = sh_size;
                }
                if (".symtab".equals(name)) {
                    offsS = sh_offset;
                    szS = sh_size;
                }
                if (".strtab".equals(name)) {
                    osT = sh_offset;
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("Input file not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Input file exception: " + e.getMessage());
        }
        try {
            writer = new BufferedWriter(new FileWriter(args[1],
StandardCharsets.UTF_8));
        } catch (FileNotFoundException e) {
            System.out.println("Output file not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Output file exception: " + e.getMessage());
        }
    }
```

```
    public void parse () throws Exception {
        int offset = offsS;
        for (int i = 0; i < szS / 16; i++) {
            int sym_offset = offset + 16 * i;
            String name;
            if (osT + get(sym_offset, 4) == osS) {
                name = "";
            } else {
                StringBuilder s = new StringBuilder();
                int x = 0;
                while (elf[osT + get(sym_offset, 4) + x] != 0) {
                    s.append((char) elf[osT + get(sym_offset, 4) + x]);
                    x++;
                }
                name = s.toString();
            }
            SymtabLine sym = new SymtabLine(i, name, get(sym_offset + 4, 4),
                    get(sym_offset + 8, 4), get(sym_offset + 12, 1),
                    get(sym_offset + 13, 1), get(sym_offset + 14, 2));
            dictionary.put(sym.x, sym);
            symtab.add(sym);
        }
        try{
            writer.write(".text" + System.lineSeparator());
            for (int i = 0, ind = offsT; i < szT; i += 4, ind += 4) {
                var sym = dictionary.get(arT + i);
                if (sym != null && "FUNC".equals(sym.type)) {
                    writer.write(String.format("%08x   <%s>:", arT + i, sym.nm) +
System.lineSeparator());
                }
                String com = toBitString(get(ind + 3, 1)) + toBitString(get(ind + 2,
1)) +
                        toBitString(get(ind + 1, 1)) + toBitString(get(ind, 1));
                writer.write(String.format("   %05x: \t%08x\t", arT + i,
parseUnsignedInt(com, 2)));
                try {
                    writer.write(parseCommand(com) + System.lineSeparator());
                } catch (RuntimeException e) {
                    writer.write("unknown" + System.lineSeparator());
                }
            }
            writer.newLine();
            writer.write(".symtab" + System.lineSeparator() + "Symbol Value
\t" +
```

```
                    "Size Type \tBind  \tVis    \tIndex   Name" +
System.lineSeparator());
            for (var sym: symtab) {
                writer.write(sym.line + System.lineSeparator());
            }
        } catch(final IOException e) {
            throw new Exception("Writing exception: " + e.getMessage());
        }
        try{
            writer.close();
        } catch(final IOException e) {
            throw new Exception("Output file closing exception: " + e.getMessage());
        }
    }
    String parseCommand(final String s) throws Exception {
        if ("00000000000000000000000001110011".equals(s)) {
            return String.format("%7s", "ecall");
        } else if ("00000000000100000000000001110011".equals(s)){
            return String.format("%7s", "ebreak");
        }
        String repeat = String.valueOf(s.charAt(0)).repeat(20);
        final String s1 = repeat + s.substring(0, 12);
        return switch (s.substring(25, 32)) {
            case "0110111", "0010111" -> String.format("%7s\t%s, %s", switch
(s.substring(25, 32)) {
                case "0010111" -> "auipc";
                case "0110111" -> "lui";
                default -> throw new Exception("Unknown instruction");
            },  getRegister(s.substring(20, 25)), (parseUnsignedInt(s.substring(0,
20), 2) << 12) + "");
            case "1101111" -> String.format("%7s\t%s, %s", "jal",
getRegister(s.substring(20, 25)),
                    switch (dictionary.containsKey(arT +
                            (parseUnsignedInt(String.valueOf(s.charAt(0)).repeat(12) +
                                    s.substring(12, 20) + s.charAt(11) +
s.substring(1, 11), 2) << 1)) ? 1 : 0) {
                case 0 -> {
                    String label = String.format("L%d", dictionary.size());
                    dictionary.put(arT, new SymtabLine(0, label, 0, 0, 0, 0, 0));
                    yield label;
                }
                case 1 -> dictionary.get(arT +
(parseUnsignedInt(String.valueOf(s.charAt(0)).repeat(12) +
                        s.substring(12, 20) + s.charAt(11) + s.substring(1, 11), 2) <<
1)).nm;
```

```
                default -> throw new Exception("Incorrect label");
            });
            case "1100111" -> String.format("%7s %s, %s(%s)", "jalr",getRegister(
                    s.substring(20, 25)), parseUnsignedInt(s1, 2) + "",
getRegister(s.substring(12, 17)));
            case "1100011" -> String.format("%7s\t%s, %s, %s", switch (s.substring(17,
20)) {
                case "000" -> "beq";
                case "001" -> "bne";
                case "100" -> "blt";
                case "101" -> "bge";
                case "110" -> "bltu";
                case "111" -> "bgeu";
                default -> throw new Exception("Unknown instruction");
            }, getRegister(s.substring(12, 17)),  getRegister(s.substring(7, 12)),
switch (dictionary.containsKey(arT + (parseUnsignedInt(
                    repeat + s.charAt(24) + s.substring(1, 7) + s.substring(20, 24),
2) << 1) + 4) ? 1 : 0) {
                case 0 -> {
                    String label = String.format("L%d", dictionary.size());
                    dictionary.put(arT, new SymtabLine(0, label, 0, 0, 0, 0, 0));
                    yield label;
                }
                case 1 -> dictionary.get(arT + (parseUnsignedInt(
                        repeat + s.charAt(24) + s.substring(1, 7) + s.substring(20,
24), 2) << 1) + 4).nm;
                default -> throw new Exception("Incorrect label");
            });
            case "0000011" -> String.format("%7s %s, %s(%s)", switch (s.substring(17,
20)) {
                case "000" -> "lb";
                case "001" -> "lh";
                case "010" -> "lw";
                case "100" -> "lbu";
                case "101" -> "lhu";
                default -> throw new Exception("Unknown instruction");
            }, getRegister(s.substring(20, 25)), parseUnsignedInt(s1, 2) + "",
getRegister(s.substring(12, 17)));
            case "0100011" -> String.format("%7s %s, %s(%s)", switch (s.substring(17,
20)) {
                case "000" -> "sb";
                case "001" -> "sh";
                case "010" -> "sw";
                default -> throw new Exception("Unknown instruction");
            }, getRegister(s.substring(7, 12)), (parseUnsignedInt(repeat
```

```
                    + s.substring(0, 7) + s.substring(20, 25), 2)) + "",
getRegister(s.substring(12, 17)));
            case "0010011" -> String.format("%7s\t%s, %s, %s", switch (s.substring(17,
20)) {
                case "000" -> "addi";
                case "001" -> {
                    if (("0".repeat(7)).equals(s.substring(0, 7))) {
                        yield "slli";
                    } else {
                        throw new Exception("Unknown instruction");
                    }
                }
                case "010" -> "slti";
                case "011" -> "sltiu";
                case "100" -> "xori";
                case "101" -> switch (s.substring(0, 7)) {
                    case "0000000" -> "srli";
                    case "0100000" -> "srai";
                    default -> throw new Exception("Unknown instruction");
                };
                case "110" -> "ori";
                case "111" -> "andi";
                default -> throw new Exception("Unknown instruction");
            },  getRegister(s.substring(20, 25)), getRegister(s.substring(12, 17)),
                    "101".equals(s.substring(17, 20)) || "001".equals(s.substring(17,
20)) ?
                            String.valueOf(parseUnsignedInt(s.substring(7, 12), 2))
:parseUnsignedInt(s1, 2) + "");
            case "0110011" -> String.format("%7s\t%s, %s, %s", switch (s.substring(0,
7)) {
                case "0000000" -> switch (s.substring(17, 20)) {
                    case "000" -> "add";
                    case "001" -> "sll";
                    case "010" -> "slt";
                    case "011" -> "sltu";
                    case "100" -> "xor";
                    case "101" -> "srl";
                    case "110" -> "or";
                    case "111" -> "and";
                    default -> throw new Exception("Unknown instruction");
                };
                case "0100000" -> switch (s.substring(17, 20)) {
                    case "000" -> "sub";
                    case "101" -> "sra";
```

```
                    default -> throw new Exception("Unknown instruction");
                };
                case "0000001" -> switch (s.substring(17, 20)) {
                    case "000" -> "mul";
                    case "001" -> "mulh";
                    case "010" -> "mulhsu";
                    case "011" -> "mulhu";
                    case "100" -> "div";
                    case "101" -> "divu";
                    case "110" -> "rem";
                    case "111" -> "remu";
                    default -> throw new Exception("Unknown instruction");
                };
                default -> throw new Exception("Unknown instruction");
            },  getRegister(s.substring(20, 25)),  getRegister(s.substring(12, 17)),
getRegister(s.substring(7, 12)));
            case "0001111" -> String.format("%7s\t", "fence",
("001".equals(s.substring(17, 20))? ".i": ""));
            case "1110011" -> String.format("%7s\t%s, %s, %s", switch (s.substring(17,
20)) {
                case "001" -> "csrrw";
                case "010" -> "csrrs";
                case "011" -> "csrrc";
                case "101" -> "csrrwi";
                case "110" -> "csrrsi";
                case "111" -> "csrrci";
                default -> throw new Exception("Unknown instruction");
            },  (s.substring(20, 25)), getRegister(s.substring(0, 7) + s.substring(7,
12)), getRegister(s.substring(12, 17)));
            default -> throw new Exception("Unknown instruction");
        };
    }
    private String getRegister(String x) {
        return switch (parseInt(x, 2)) {
            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 -> null;
        };
    }
    private String toBitString (int b) {
        StringBuilder s = new StringBuilder();
        int sz = 8;
        for (int i = 0; i < sz; i++) {
            s.insert(0, (char) (b % 2 + '0'));
            b /= 2;
        }
        return String.valueOf(s);
    }
    public int get(int start, int cnt) {
        int answer = 0;
        for (int i = cnt - 1; i >= 0; i--) {
            answer = (answer << 8) + elf[start + i];
        }
        return answer;
    }
}
```

SymtabLine.java

```
public class SymtabLine {
    public final String nm;
    public final int x;
    public final String line;
    public final String type;
    public SymtabLine(int symbol, String nm, int x, int sz, int inf, int access, int
i) {
        this.nm = nm;
        this.x = x;
        line = String.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s", symbol, x, sz,
                switch (inf & 15) {
                    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 -> "UNKNOWN";
                }, switch (inf >> 4) {
                    case 0 -> "LOCAL";
                    case 1 -> "GLOBAL";
                    case 2 -> "WEAK";
                    case 10 -> "LOOS";
                    case 12 -> "HIOS";
                    case 13 -> "LOPROC";
                    case 15 -> "HIPROC";
                    default -> "UNKNOWN";
                }, switch(access & 3) {
                    case 0 -> "DEFAULT";
                    case 1 -> "INTERNAL";
                    case 2 -> "HIDDEN";
                    case 3 -> "PROTECTED";
                    default -> "UNKNOWN";
                }, switch (i) {
                    case 0 -> "UNDEF";
                    case 65280 -> "LOPROC";
                    case 65281 -> "AFTER";
```

```
                    case 65282 -> "AMD64_LCOMMON";
                    case 65311 -> "HIPROC";
                    case 65312 -> "LOOS";
                    case 65343 -> "HIOS";
                    case 65521 -> "ABS";
                    case 65522 -> "COMMON";
                    case 65535 -> "XINDEX";
                    default -> String.valueOf(i);
                }, nm);
        type = switch (inf & 15) {
            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 -> "UNKNOWN";
        };
    }
}
```