Лабораторная работа: № 4	ФИО: Кинзябулатов Эдуард Шамилевич
Название работы: ISA	Группа: M3137

Ссылка на репозиторий: https://github.com/skkv-itmo/itmo-comp-arch-2023-riscv-Tortik3000

Инструментарий: Python (3.11.4)

Описание ISA RISC-V:

RISV-V это специальная ISA, реализованная на принципах RISC.

RISC-V это архитектура Reg Reg, в которой с памятью общаются только два типа инструкций(load, store).

В лабораторной используются наборы команд: RV32I который включает 40 инструкций(базовая спецификация для целочисленной арифметики), RV32M который включает 8 инструкций(базовое расширение целочисленное умножение/деление)

Так как у нас 32 разрядная спецификация то она содержит 32 регистра, инструкции состоят из 4 байт.

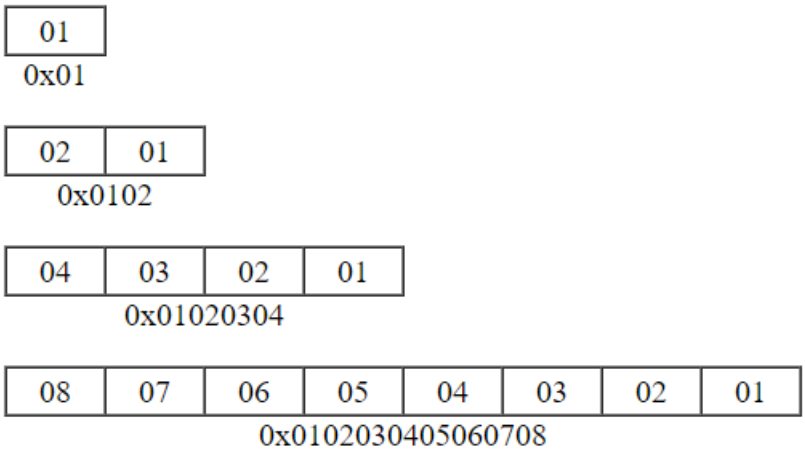
Результат работы программы:

```
.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
```

Кодировка little endian:

Кодировка определяет значения дополнения до 2, при этом младший байт занимает младший адрес



(приложение №2 Figure 4-5)

Реализация функции декодировки данных:

```
def decoderElfData(data):
    num = ""
    for i in range(len(data) - 1, -1, -1):
        nowBit = hex(data[i])[2:]
        if (len(nowBit) == 1):
            nowBit = "0" + nowBit
        num += nowBit
    return int(num, 16)
```

Структура ELF файла

В начале файла идет заголовок ELF файла, после которого идет таблица заголовка, таблица заголовков программы.

ELF Header:

```
typedef struct {
          unsigned char  e_ident[EI_NIDENT];
          Elf32_Half     e_type;
          Elf32_Half     e_machine;
          Elf32_Word     e_version;
          Elf32_Addr     e_entry;
          Elf32_Off      e_phoff;
          Elf32_Off      e_shoff;
          Elf32_Word     e_flags;
          Elf32_Half     e_ehsize;
          Elf32_Half     e_phentsize;
          Elf32_Half     e_phnum;
          Elf32_Half     e_shentsize;
          Elf32_Half     e_shnum;
          Elf32_Half     e_shstrndx;
} Elf32_Ehdr;
```

(Приложение №1 стр 18)

В лабораторной использовались следующие поля:

e_shoff (с 32 по 35 байт)- индекс начала section header от начала файла.

e_shentize(с 46 по 47 байт) – размер заголовка section header в байтах. Обычно 40 байт

e_shstrndx(с 50 по 52 байт) – индекс в section header, который указывает на таблицу имени раздела.

e_shnum(с 48 по 50 байт) – количество заголовков в section header. С 48 по 50 байт

Section Header:

```
typedef struct
{
    uint32_t   sh_name;
    uint32_t   sh_type;
    uint32_t   sh_flags;
    uint32_t   sh_addr;
    uint32_t   sh_offset;
    uint32_t   sh_size;
    uint32_t   sh_link;
    uint32_t   sh_info;
    uint32_t   sh_addralign;
    uint32_t   sh_entsize;
} Elf32_Shdr;
```

(Приложение №1 стр 24)

В лабораторной использовались следующие поля:

sh_name(с 0 по 3 байт) – смещение в данных секции, индекс которой задается в поле e_shstrndx.

По этому смещению размещается строка, завершающаяся нулевым байтом, являющаяся именем секции.

sh_addr(с 12 по 15 байт) – хранит виртуальный адрес начала секции

sh_offset(с 16 по 19 байт) – хранит смещение от начала файла, по которому размещаются данные секции

sh_size(с 20 по 23 байт) – хранит размер секции в байтах

Symbol Table:

Каждые 16 байт в .symtab преобразуются в такие данные:

```
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;
```

(Приложение №1 стр 32)

st_name(с 0 по 3 байт) - индекс таблицы .strtab, в которой хранятся символьные представления имен символов

st_value(с 4 по 7 байт) – значение связанного символа

st_size(с 8 по 11 байт) – размер объекта данных на которые ссылается данная запись

st_info(12 байт) – тип символа

st_other(13 байт) – отвечает за visibility

st_shndx(с 14 по 15 байт) – соответствующий индекс таблицы заголовков разделов

bind = info >> 4

type = info & 0xf

vis = other & 0x3

Соответствующие значения bind:

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOPROC	13
STB_HIPROC	15

(Приложение №1 стр 33)

Соответствующие значения type:

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_LOPROC	13
STT_HIPROC	15

(Приложение №1 стр 34)

Соответствующие значения vis:

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

Соответсвующие значения index:

Name	Value
SHN_UNDEF	0
SHN_LORESERVE	0xff00
SHN_LOPROC	0xff00
SHN_HIPROC	0xff1f
SHN_ABS	0xfff1
SHN_COMMON	0xfff2
SHN_HIRESERVE	0xffff

(Приложение №1 стр 23)

Регистры:

Приложение №3

Registers

Register	ABI Name	Description	Saver
x0	zero	Zero constant	—
x1	ra	Return address	Callee
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Fn args/return values	Caller
x12-x17	a2-a7	Fn args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Таб. 1

Так как используются только целочисленные инструкции,

в лабораторной работе используются только целочисленные регистры

Text

Text состоит из подряд идущих инструкций по 4 байта

Существует 6 типов команд R, S, I, B, U, J.

(Приложение №4)

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
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Таб. 2

Команды R типа:

В командах R типа используется три регистра в качестве операндов: два регистра источника (rs1, rs2) и регистр значение. Команда состоит из полей funct7, funct3, opcode, rs1, rs2, rd

Funct7, funct3 и opcode нужны для того чтобы узнать какая команда.

Команды I типа:

Команды I типа используют в качестве операндов: регистр источник(rs1), регистр значение(rd),

и константу (imm). Команда состоит из полей imm, rs1, rd, funct3, opcode

Константа imm это 12 битное число со знаком в дополнении до двух.

Для команд slli, srli, srai – imm 5 битное значение сдвига

Команды S типа:

Команды S типа используют в качестве операндов: два регистра источника(rs1, rs2) и константу imm

Константа imm это 12 битное число со знаком в дополнении до двух, она состоит из двух полей

imm[11:5] = command[31:25] и imm[4:0] = command[11:7]

Команды B типа:

Аналогично командам типа S, только константа является 13 битным числом у которого imm[0] = 0

Imm[1:4] = command[8:11], imm[5:10] = command[25, 30], imm[11] = command[7], imm[12] = command[31]

Команды U типа:

В командах U типа используется один операнд регистра значения(rd) и один операнд константа

Imm[0:20] = command[12:31]

Команды J типа:

Аналогично командам U типа, но константа imm[20:0] = command[20|10:1|11|19:12]

Команда fence:

(Приложение №4)

31-28	27-24	23-20	19-15	14-12	11-7	6-2	1-0
0000	pred	succ	00000	000	00000	00011	11

В инструкции fence каждый бит в pred и succ отвечает за одну из 4-х опций:

Устройства ввода(I), вывода(O), чтения(R) и записи(W) памяти

Пример разбора инструкции:

```
def commandB(addr, data, indexMark, countUrl):
    value = int(data, 2)
    opcode = data[-7:]
    funct3 = data[-15: -12]
    rs1 = registerRiscFive[int(data[-20:-15], 2)]
    rs2 = registerRiscFive[int(data[-25:-20], 2)]
    imm = dopForTwo(data[-32] + data[-8] + data[-31:-25] + data[-12:-8] + "0")
    name = CommandB[opcode + funct3]
    addrUrl = imm + addr

    if (addrUrl not in indexMark):
        indexMark[addrUrl] = "L" + str(countUrl)
        countUrl += 1
    nameUrl = indexMark[addrUrl]

    return ("   %05x:\t%08x\t%7s\t%s, %s, 0x%x, <%s>\n" %
            (addr, value, name, rs1, rs2, addrUrl, nameUrl)), countUrl
```

В соответствии с таб. 2

Data разбивается на поля: funct7, funct3, opcode, rs1, rs2, rd, imm.

Opcode нужен для определения типа команды.

Opcode, funct7, funct3 по этим данным из словаря названий для каждого типа команд достается название команды

Названия регистров rs1, rs2, rd достаются из словаря по номеру регистра(в соответствии с таб. 1)

Imm переводится в число типа int в формате дополнения до двух

Если у нас команда типа B или J, то у таких команд есть метки

У метки есть адрес и название, адрес собирается из адреса команды и значения константы

Название метки достается из словаря меток indexMark(который пополнялся во время парсинга symtab) если оно там есть, иначе создается новая метка(название которой завист от количества уже созданных меток) и добавляется в indexMark.

если его там нет то создается новая метка

Описание кода

```
if(len(sys.argv) == 3):
    try:
        with open(sys.argv[1], 'rb') as inputFile:
            inputList = []
            lines = inputFile.readlines()
            for line in lines:
                inputList += list(line)

    except FileNotFoundError:
        print(f"FileNotFound: {sys.argv[1]}", file=sys.stderr)


    if(chr(inputList[0]) + chr(inputList[1]) + chr(inputList[2]) + chr(inputList[3])  == '\x7fELF'):
        Out = parser(inputList)

        with open(sys.argv[2], 'w') as outputFile:
            for line in Out:
                outputFile.write(line)
    else:
        print("Expect ELF file", file=sys.stderr)

else:
    print(f"Expect 2 argument, actual: {len(sys.argv) - 1}", file=sys.stderr)
```

Здесь происходит чтение входного файла, каждый символ хранится в массиве inputList

в численном представлении.

Далее для декодирования файла вызывается функция parser

И потом декодированный файл выводится в выходной файл

Функция parser:

```
def parser(inputList):
    e_shentsize = 40
    e_shoff = decoderElfData(inputList[32:36])
    e_shnum = decoderElfData(inputList[48:50])
    e_shstrndx = decoderElfData(inputList[50: 52])

    indexShstrndx = e_shstrndx * e_shentsize + e_shoff
    indexShstrndxOffset = indexShstrndx + 16
    indexShstrndxData = decoderElfData(inputList[indexShstrndxOffset: indexShstrndxOffset + 4])

    symtabOut = []
    indexMark = parserSymtab(inputList, e_shoff, indexShstrndxData, symtabOut)

    textOut = []
    parserText(inputList, e_shoff, indexShstrndxData, indexMark, textOut)
    return textOut + symtabOut
```

На вход функции parser подается массив входных символов,

Потом из входных данных достается e_shoff(индекс начала таблицы с заголовками раздела), e_shstrndx(индекс начала таблицы с именами).

Далее вычисляется индекс начала данных таблицы shstrnndx.

Далее вызывается функция parserSymtab, которая собирает .symtab,

также эта функция возвращает словарь indexMark, который содержит название меток

для вывода .text

Функция searchIndexHeader:

```
def searchIndexHeader(inputList, nameHeader, indexStartHeader, indexShstrndxData):
    ans = 0
    i = 0
    while ans == 0:
        name = ""
        indexNow = indexShstrndxData + decoderElfData(
            inputList[indexStartHeader + i * 40:indexStartHeader + i * 40 + 4])
        while inputList[indexNow] != 0:
            name += chr(inputList[indexNow])
            indexNow += 1
        if name == nameHeader:
            ans = indexStartHeader + i * 40
        i += 1
    return ans
```

Функция searchIndexHeader,принимает на вход название таблицы индекс начала которой нужно найти, индекс начала таблицы с заголовками(e_shoff), индекс начала данных таблицы shstrnndx(indexShstrndxData), и возвращает индекс начала нужной таблицы.

Здесь пока не найдется нужный индекс в таблице заголовков, происходит поиск имени(имена разделены нулевым байтом) нужной таблицы.

Функция parserSymtab:

```
def parserSymtab(inputList, e_shoff, indexShstrndxData, symtabOut):
    indexSymtabHeader = searchIndexHeader(inputList,  nameHeader: ".symtab", e_shoff, indexShstrndxData)
    indexStrtabHeader = searchIndexHeader(inputList,  nameHeader: ".strtab", e_shoff, indexShstrndxData)

    indexDataSymtab = decoderElfData(inputList[indexSymtabHeader + 16: indexSymtabHeader + 20])
    sizeDataSymtab = decoderElfData(inputList[indexSymtabHeader + 20: indexSymtabHeader + 24])
    indexDataStrtab = decoderElfData(inputList[indexStrtabHeader + 16: indexStrtabHeader + 20])

    indexMark = {}
    symtabOut.append("\n.symtab\n")
    symtabOut.append("\nSymbol Value              Size Type     Bind     Vis       Index Name\n")
```

С помощью функции searchIndexHeader, находится индекс начала таблиц symtab и strtab

Далее в этих таблицах находятся индексы начала данных symtab и strtab, и размер symtab

```
for i in range(sizeDataSymtab // 16):
    nowMark = inputList[indexDataSymtab + i * 16: indexDataSymtab + (i + 1) * 16]
    indexName = indexDataStrtab + decoderElfData(nowMark[0:4])
    name = ""
    while inputList[indexName] != 0:
        name += chr(inputList[indexName])
        indexName += 1

    if (decoderElfData(nowMark[14:16]) in indexSymtab):
        index = indexSymtab[decoderElfData(nowMark[14:16])]
    else:
        index = decoderElfData(nowMark[14:16])
    symtabOut.append("[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\n" % (
        i, decoderElfData(nowMark[4:8]), decoderElfData(nowMark[8:12]),
        typeSymtab[decoderElfData(nowMark[12: 13]) & 15],
        bindSymtab[decoderElfData(nowMark[12: 13]) >> 4], visSymtab[decoderElfData(nowMark[13:14]) & 3], index,
        name))
    if (typeSymtab[decoderElfData(nowMark[12: 13]) & 15] == "FUNC"):
        indexMark[decoderElfData(nowMark[4:8])] = name
return indexMark
```

Далее каждые 16 байт в данных symtab, разделяются на: st_name(индекс имени в strtab), value, size, type, bind, vis, index, которые нужны для вывода

Если встретилась строка с типом FUNC, то эта метка добавляется в словарь indexMark

для вывода меток в .text

Функция parserText:

```
def parserText(inputList, e_shoff, indexShstrndxData, indexMark, textOut):
    textOut.append(".text\n")
    indexTextHeader = searchIndexHeader(inputList,  nameHeader: ".text", e_shoff, indexShstrndxData)
    virtualAddr = decoderElfData(inputList[indexTextHeader + 12: indexTextHeader + 16])
    indexData = decoderElfData(inputList[indexTextHeader + 16: indexTextHeader + 20])
    sizeData = decoderElfData(inputList[indexTextHeader + 20: indexTextHeader + 24])
    countUrl = 0
    commands = []
    for i in range(sizeData // 4):
        command = decoderElfData(inputList[indexData + i * 4: indexData + (i + 1) * 4])
        command = (32 - len(bin(command)[2:])) * "0" + bin(command)[2:]
        opcode = command[-7:]
        typeCom = commandType[opcode]

        addr = virtualAddr + i * 4

        if typeCom == "I":
            commands.append(commandI(addr, command))
        elif typeCom == "R":
            commands.append(commandR(addr, command))
        elif typeCom == "S":
            commands.append(commandS(addr, command))
        elif typeCom == "U":
            commands.append(commandU(addr, command))
        elif typeCom == "B":
            text, countUrl = commandB(addr, command, indexMark, countUrl)
            commands.append(text)
```

Здесь с помощью функции searchIndexHeader, находится индекс начала таблицы text.

Далее в этой таблице находится адрес первой команды, индекс начала данных и размер данных.

Затем происходит получение команд, каждые 4 байта данных переводятся в 32 битное число.

Потом с помощью opcode = command[-7:] определяется тип команды, и вызывается функция соответствующая данной команде, которая собирает выходную строку

```
        if typeCom == "I":
            commands.append(commandI(addr, command))
        elif typeCom == "R":
            commands.append(commandR(addr, command))
        elif typeCom == "S":
            commands.append(commandS(addr, command))
        elif typeCom == "U":
            commands.append(commandU(addr, command))
        elif typeCom == "B":
            text, countUrl = commandB(addr, command, indexMark, countUrl)
            commands.append(text)
        elif typeCom == "J":
            text, countUrl = commandJ(addr, command, indexMark, countUrl)
            commands.append(text)
        elif typeCom == "fence":
            commands.append(commandFence(addr, command))
        else:
            commands.append(commandUnknown(addr, command))

    for i in range(sizeData // 4):
        if virtualAddr + i * 4 in indexMark:
            textOut.append("\n%08x \t<%s>:\n" % (
                virtualAddr + i * 4, indexMark[virtualAddr + i * 4]))
        textOut.append(commands[i])
    textOut.append("\n")
```

После обработки всех команд, идет проверка на существование метки и добавление команды в выходной массив

Инструкции:

```
def commandB(addr, data, indexMark, countUrl):
    value = int(data, 2)
    opcode = data[-7:]
    funct3 = data[-15: -12]
    rs1 = registerRiscFive[int(data[-20:-15], 2)]
    rs2 = registerRiscFive[int(data[-25:-20], 2)]
    imm = dopForTwo(data[-32] + data[-8] + data[-31:-25] + data[-12:-8] + "0")
    name = CommandB[opcode + funct3]
    addrUrl = imm + addr

    if (addrUrl not in indexMark):
        indexMark[addrUrl] = "L" + str(countUrl)
        countUrl += 1
    nameUrl = indexMark[addrUrl]

    return ("   %05x:\t%08x\t%7s\t%s, %s, 0x%x, <%s>\n" %
            (addr, value, name, rs1, rs2, addrUrl, nameUrl)), countUrl
```

Из данных(data) инструкции достаются соответствующие поля и собираются в выходную строку, если у команды тип B или J, то проверяется есть ли метка с таким же адресом в словаре и если нет то добавляет в словарь новую метку

Используемые источники:

1. https://refspecs.linuxfoundation.org/elf/elf.pdf

(для структуры ELF header, symtab, text, section header)

2. https://refspecs.linuxbase.org/elf/gabi4+/contents.html

(для структуры ELF header, symtab, text, section header, кодировки little endian)

3. https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/notebooks/RISCV/RISCV_CARD.pdf

(для разбора инструкций и регистров)

4. https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#sw

(для разбора инструкций в частности fence)