| Лабораторная работа №3 | M3137 | 2022 |
|---|---|---|
| Название работы: ISA | Кузнецов Сергей Павлович | |

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

**Инструментарий и требования к работе:** Python 3.11

**Описание системы кодирование команд RISC-V**

ISA RISC-V представляет собой архитектуру Reg Reg, с памятью общаются только два типа команд (load/store), а остальные взаимодействуют внутри с регистрами. RISC-V представляет собой семейство связанных ISA, каждый базовый набор целочисленных команд характеризуется шириной целочисленных регистров и соответствует размером адресного пространства и количеством целочисленных регистров. Существует два варианта первичных базовых целых чисел RV32I и RV64I, которые представляют собой 32-разрядные или 64-разрядные адресные пространства соответственно.

У каждого регистра есть номер от 0 до 31 и специальное имя для обозначения обычного назначения регистра.

| Название | Номер | Назначение |
|---|---|---|
| zero | x0 | Константа нуля |
| ra | xl | Адрес возврата (от англ. *return address*) |
| sp | x2 | Указатель стека (от англ. *stack pointer*) |
| gp | x3 | Глобальный указатель (от англ. *global pointer*) |
| Tp | x4 | Указатель потока (от англ. *thread pointer*) |
| t0—t2 | x5—x7 | Временные переменные |
| s0/fp | x8 | Сохраняемая переменная / Указатель фрейма стека |
| s1 | x9 | Сохраняемая переменная |
| a0—a1 | x10—x11 | Аргументы функций / Возвращаемые значения |
| a2—a7 | x12—x17 | Аргументы функций |
| s2—s11 | x18—x27 | Сохраняемые переменные |
| t3—t6 | x28—x31 | Временные переменные |

Рисунок 1 – набор регистров RISC-V

В нулевом регистре всегда хранится константа 0; попытка записать в него другое значение игнорируется. Регистры от s0 до s11 (регистры 8–9 и 18–27) и от t0 до t6 (регистры 5–7 и 28–31) используются для хранения переменных; ra и регистры от a0 до a7 служат для вызовов функций.

Существует 6 типов команд R, I, S, B, U, J-types. Команды R типа использует три регистра в качестве операндов: два регистра-источника и один регистр-значение. 32-битная команда состоит из шести полей funct7, rs2, rs1, funct3, rd и opcode. Rs2 и rs1 регистры источники, rd – регистр значение.

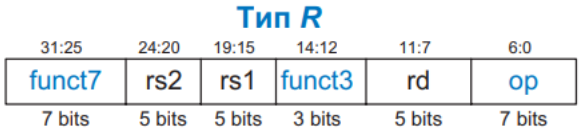


Рисунок 2 – команда R типа

К командам R типа относится математические команды (add, sub, mul, div, rem), команды сдвига (sll, srl, sra) и логические операции (and, or, xor).

Команда типа I используют два регистровых операнда и один непосредственный операнд (константу). К инструкциям I типа относятся addi, andi, ori, xori, операции загрузки (lw, lh, lb, lhu, lbu) и регистрового перехода jalr.

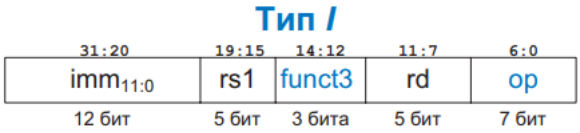


Рисунок 3 – команда I типа

Поле константы представляет собой 12-битное число со знаком (в дополнительном коде), для все инструкций кроме slli, srli srai. Для них поле imm представляет собой 5-битное значение сдвига (без знака).

Аналогично инструкциям типа I, инструкции типа S / B (store / branch, хранение слова в памяти / условный переход) используют два регистровых операнда и один непосредственный операнд. Но оба операнда являются

регистрами-источниками (rs1 и rs2). Константа imm разбита на поля funct7 и rd.

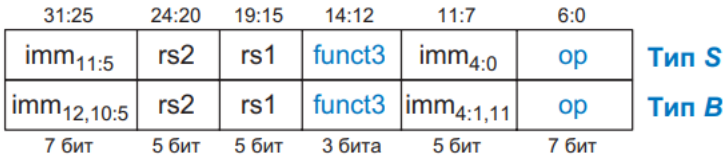


Рисунок 4 – команды типа S и B

Команды сохранения в памяти используют тип S, а команды условного перехода используют тип B. Форма S и B различаются только кодированием константы. Код инструкции типа S содержит 12-битную константу со знаком (в дополнительном коде), со старшими семью битами (imm11:5) в битах 31:25 кода команды и младшими пятью битами (imm4:0) в битах 11:7 кода команды. Команды типа B содержат в коде 13-битовую константу со знаком, представляющую собой смещение перехода (branch offset), но только старшие 12 бит присутствуют в коде команды.

Инструкции типа U/J (upper immediate/jump, старшие разряды константы / безусловный переход) содержат в своем машинном коде один операнд регистра-назначения rd и 20-битовое поле константы. Аналогично другим видам инструкций, инструкции типа U/J имеют 7-битный opcode. В инструкциях типа U оставшиеся биты отведены под 20 старших разрядов 32-битной константы. В инструкциях типа J оставшиеся биты отведены под 20 старших бит 21-битной константы смещения безусловного перехода.

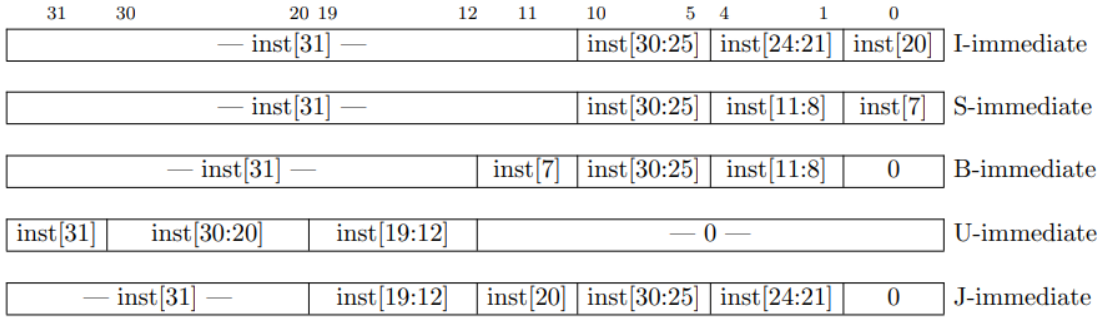


Рисунок 5 – кодирование констант в различных типах

**Описание структуры ELF файла**

Каждый файл ELF состоит из одного заголовка ELF, за которым следуют данные файла. Данные могут включать:

- Таблица заголовка программы, описывающая ноль или более сегментов
- Таблица заголовков разделов, описывающая ноль или более разделов

Данные, на которые ссылаются записи в таблице заголовка программы или таблице заголовка раздела

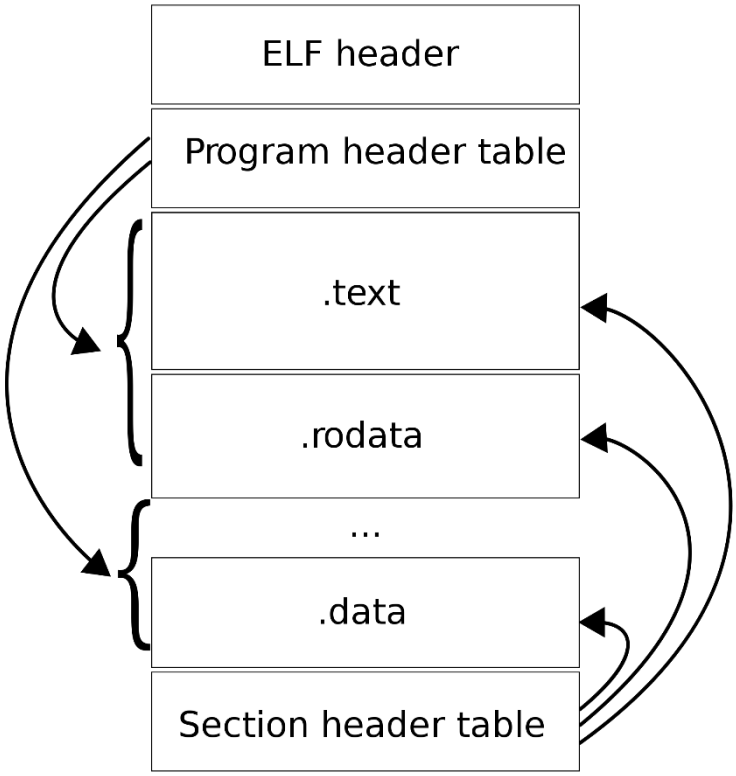


Рисунок 6 – структура elf файла

Первые 4 байта это 0x7f и ELF (0x45, 0x4c, 0x46).

e_shoff – индекс начала таблицы заголовка раздела с 32 по 35 байт.
e_shstrndx – индекс записи таблицы, которая содержит названия разделов с 50 по 51 байты.

У каждого заголовка раздела (таблицы) есть sh_name с 0 по 3 байт (+ индекс начала заголовка), которое содержит смещение к строке в разделе .shstrndx, которая представляет имя этого раздела.

sh_addr – виртуальный адрес в памяти с 12 по 15 байты, sh_offset – смещение раздела в файле с 16 по 19 байты. sh_size – размер раздела файла. Все значения байтов указаны без смещения к индексу начала заголовка. Размер одной таблицы заголовка – 40 байт.

Symbol Table Section содержит такие элементы как name (4 байта), value (байты), size (байта), info (1 байт), other (1 байт), shndx (2 байта). В сумме 16 байт, т.е. каждые 16 байт в данных такой таблицы 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;
```

Рисунок 7 – элементы symbol table section

Из этих данных можно получить bind = info >> 4, type = info & 0xf, vis = 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 |

Рисунок 8 – соответствующие значение bind

| 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 |

Рисунок 9 – соответствующие значение type

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

Рисунок 10 – соответствующие значение vis

```
0: "UNDEF",
0xff00: "LOPROC",
0xff1f: "HIPROC",
0xfff1: "ABS",
0xfff2: "COMMON",
0xffff: "HIRESERVE"
```

Рисунок 11 – соответствующие значение shndx

**Описание работы написанного кода**

Исполняем файл имеет название main.py. Для выполнения задания служит функция disassembler, которая принимает на вход название входного и выходного файлов.

```
def disassembler(input, output):
    with open(input, 'rb') as file_input:
        _list = []
        lines = file_input.readlines()
        for i in lines:
            _list.extend(list(i))
        ELF = chr(_list[1]) + chr(_list[2]) + chr(_list[3])
        if ELF != "ELF":
            raise TypeError("Type Error")
        e_shoff = convert_to_10_in_list_byte(_list, 32, 4) # index start header
        e_shstrndx = 40 * convert_to_10_in_list_byte(_list, 50, 2) + e_shoff  #
index header names
        size_section_header = 40
        index_text_header, index_symtab_header, index_strtab_header =
search_index_text_symtab_strtab(_list, e_shoff, e_shstrndx, size_section_header)

        size_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header +
20, 4)
        index_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header +
16, 4)
        index_data_strtab = convert_to_10_in_list_byte(_list, index_strtab_header +
16, 4)
```

Таблица 1 – функция disassembler в файле main.py

Происходит чтение входного файла в бинарном формате, и каждый байт обрабатывается как 8 битное число и приводится в 10 формат. Далее проверяется, что входной файл является elf файлом, иначе выкидывается исключение об ошибке. Далее из данных выбираются e_shoff и e_shstrndx, при помощи функции convert_to_10_list_byte, которая конвертирует массив байт в 10 число, на вход ей подается список, в котором находятся нужные байты, индекс начала нужного байта, количество нужных байт.

```
def convert_to_10_in_list_byte(_list, index, size):
    num = ""
    vector = _list[index: index + size][::-1]  # little endian
    for i in vector:
        now = hex(i)[2:]
        now = (2 - len(now)) * "0" + now
        num += now
    return int(num, 16)
```

Таблица 2 – функция convert_to_10_list_byte в файле convert.py

Далее после выбора e_shoff и e_shstrndx, происходит поиск индексов таблиц symtab, strtab, text, при помощи функции search_index_text_symtab_strtab, которая принимает аргументы массив байт, индекс начала таблицы с заголовками (e_shoff), индекс начала таблицы с именами (e_shstrndx), размер одной таблицы с заголовками.

```
def search_index_text_symtab_strtab(_list, index_start_header, index_header_names,
size_section_header=40):
    index_shstrndx_data = convert_to_10_in_list_byte(_list, index_header_names + 16,
4)
    symtab = 0
    strtab = 0
    text = 0
    i = 0
    while not symtab or not strtab or not text:
        name = ""
        index_now = index_shstrndx_data + convert_to_10_in_list_byte(_list,
index_start_header + i * size_section_header, 4)
        while _list[index_now] != 0:
            name += chr(_list[index_now])
            index_now += 1
        if name == ".text":
            text = index_start_header + i * size_section_header
        elif name == ".symtab":
            symtab = index_start_header + i * size_section_header
        elif name == ".strtab":
            strtab = index_start_header + i * size_section_header
        i += 1
    return text, symtab, strtab
```

Таблица 3 – функция search_index_text_symtab_strtab в main.py

В этой функции ищется индекс начала данных таблицы с заголовками. Пока не найдутся индексы нужных таблиц заголовков происходит поиск в данных таблицы shstrndx, из которых собирается имя текущей таблицы. И так продолжается пока не найдутся нужные индексы начала таблиц.

После того как найдены индекс таблиц text, symtab, strtab происходит парсинг symtab

```
size_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header + 20, 4)
index_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header + 16, 4)
index_data_strtab = convert_to_10_in_list_byte(_list, index_strtab_header + 16, 4)
```

```
dictionary_index_mark = {}
list_item_symtab = []
for i in range(size_data_symtab // 16):
    list_item_symtab.append(SymbolTableSection(_list[index_data_symtab + i *
16:index_data_symtab + (i + 1) * 16]))
symtab_out = ["Symbol Value              Size Type     Bind    Vis          Index
Name\n"]
for i in range(size_data_symtab // 16):
    now = list_item_symtab[i]
    index_name = index_data_strtab + now.get_name()
    name = ""
    while _list[index_name] != 0:
        name += chr(_list[indДex_name])
        index_name += 1
    symtab_out.append("[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\n" % (
           i, now.get_value(), now.get_size(), now.get_type(), now.get_bind(),
now.get_vis(), now.get_index(), name))
    if now.get_type() == "FUNC":
        dictionary_index_mark[now.get_value()] = name
```

Таблица 4 – парсинг symtab в функции dissembler

Для парсинга требуется размер и индекс начала данных symtab, которые находятся из таблицы. Также, чтобы получить имя текущего слота, нужно найти strtab, в котором последовательно хранятся имена для команд из symtab. Далее каждый 16 байт из данных symtab преобразуются в класс SymbolTableSection.

```
class SymbolTableSection:
    def __init__(self, data):
        self.name = convert_to_10_in_list_byte(data, 0, 4)
        self.value = convert_to_10_in_list_byte(data, 4, 4)
        self.size = convert_to_10_in_list_byte(data, 8, 4)
        self.info = convert_to_10_in_list_byte(data, 12, 1)
        self.other = convert_to_10_in_list_byte(data, 13, 1)
        self.shndx = convert_to_10_in_list_byte(data, 14, 2)
        self.type = self.info & 15
        self.bind = self.info >> 4
        self.vis = self.other & 3

    def get_value(self):
        return self.value

    def get_size(self):
        return self.size

    def get_type(self):
        return dictionary_type_symtab[self.type]

    def get_bind(self):
        return dictionary_bind_symtab[self.bind]

    def get_index(self):
```

```
        return dictionary_index_symtab[self.shndx] if self.shndx in
dictionary_index_symtab else self.shndx

    def get_vis(self):
        return dictionary_vis_symtab[self.vis]

    def get_name(self):
        return self.name
```

Таблица 5 – класс SymbolTableSection из SymbolTableSection.py

Здесь происходит разделение и получение нужных данных.

Далее из strtab собирается имя (собираются символы пока не встретится 0). И получившийся результат добавляется в массив, который в дальнейшим используется для вывода в исходный файл. Также если встретилась строка с типом FUNC, то это метка, она добавляется в словарь, ключом которой является value (двоичное представление текущей строки (16 байт)).

```
# text
virtual_addr_text = convert_to_10_in_list_byte(_list, index_text_header + 12, 4)
index_data_text = convert_to_10_in_list_byte(_list, index_text_header + 16, 4)
size_data_text = convert_to_10_in_list_byte(_list, index_text_header + 20, 4)
count_url = 0
list_commands = []
for i in range(size_data_text // 4):
    command_bin = bin(convert_to_10_in_list_byte(_list, index_data_text + i * 4,
4))[2:]
    command_bin = (32 - len(command_bin)) * "0" + command_bin
    opcode = command_bin[-7:]
    type_command = dictionary_command_type[opcode]
    try:
        if type_command == "R":
            command = CommandTypeR(virtual_addr_text + i * 4, command_bin)
        elif type_command == "I":
            command = CommandTypeI(virtual_addr_text + i * 4, command_bin)
        elif type_command == "S":
            command = CommandTypeS(virtual_addr_text + i * 4, command_bin)
        elif type_command == "B":
            command = CommandTypeB(virtual_addr_text + i * 4, command_bin)
            if command.addr_url not in dictionary_index_mark:
                dictionary_index_mark[command.addr_url] = "L" + str(count_url)
                count_url += 1
            command.set_name_url(dictionary_index_mark[command.addr_url])
        elif type_command == "U":
            command = CommandTypeU(virtual_addr_text + i * 4, command_bin)
        elif type_command == "J":
            command = CommandTypeJ(virtual_addr_text + i * 4, command_bin)
            if command.addr_url not in dictionary_index_mark:
                dictionary_index_mark[command.addr_url] = "L" + str(count_url)
                count_url += 1
            command.set_name_url(dictionary_index_mark[command.addr_url])
        else:
```

```
            command = "unknown_instruction"
    except:
        command = "unknown_instruction"
    list_commands.append(command)
```

Таблица 6 – парсинг text в функции dissembler

Также как и при прасинге symtab получаются значения индекса данных и их размер, еще виртуальный адрес, который будет нужен в дальнейшем для меток. Далее происходит получение двоичного формата текущей команды (4 байта – одна команда), по opcode вычисляется (понимается по словарю) тип команды. Далее для каждой конкретной команды существует класс, который располагается в Command.py. В классе происходит трансформация полученной команды, которая соответствует её тип. Для примера рассмотрим R тип.

```
class CommandTypeR:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.funct3 = data[-15:-12]
        self.funct7 = data[-32:-25]
        self.name = dictionary_command_R[(self.opcode, self.funct3,
self.funct7)]
        self.rd = dictionary_register_risc5[int(data[-12:-7], 2)]
        self.rs1 = dictionary_register_risc5[int(data[-20:-15], 2)]
        self.rs2 = dictionary_register_risc5[int(data[-25:-20], 2)]

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rd, self.rs1,
self.rs2)
        text = "   %05x:\t%08x\t%7s\t%s, %s, %s" % args
        return text
```

Таблица 7 – класс CommandTypeR из файла Command.py

От всех классов отличаются CommandTypeB и CommandTypeJ, у них есть метод set_name_url. Она нужна для того, чтобы можно было найти название метки в словаре, так как по входным данным можно определить только адрес. Поэтому после создания одного из этих классов, дополнительно из словаря меток по их вычисленному адресу, задается название метки.

Все команды собираются в массив для вывода.

Запись в выходной файл происходит так:

```
with open(output, "w") as file_output:
    file_output.write(".text\n")
    for i in range(size_data_text // 4):
        if virtual_addr_text + i * 4 in dictionary_index_mark:
            file_output.write("\n")
            file_output.write("%08x   <%s>:\n" % (virtual_addr_text + i * 4,
dictionary_index_mark[virtual_addr_text + i * 4]))
        file_output.write(list_commands[i].__str__() + "\n")
    file_output.write("\n")
    file_output.write(".symtab\n")
    for i in symtab_out:
        file_output.write(i)
```

Таблица 8 – запись в выходной файл

Так как у каждого класса команды есть текстовое представление, то оно идет в итоговый файл. Также проверятся адрес для простановки метки, если она существует.

## Результат работы программы

```
.text

00010074   <main>:
   10074:  ff010113      addi    sp, sp, -16
   10078:  00112623        sw    ra, 12(sp)
   1007c:  030000ef       jal    ra, 48 <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, -36 <main>
   1009c:  00050593      addi    a1, a0, 0
   100a0:  00a00893      addi    a7, zero, 10
unknown_instruction
   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, 100e4 <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, 100d8 <L1>
   10114:  050f0f13      addi    t5, t5, 80
   10118:  01478513      addi    a0, a5, 20
   1011c:  fa5f16e3       bne    t5, t0, 100c8 <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
```

**Список источников**

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

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

https://wdfiles.ru/3a32bf - книга о архитектуре RISC-V

https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

## Листинг кода

main.py

```
import sys
from SymbolTableSection import SymbolTableSection
from convert import convert_to_10_in_list_byte
from Command import dictionary_command_type, CommandTypeI, CommandTypeR, CommandTypeS,
CommandTypeB, CommandTypeU, CommandTypeJ


def search_index_text_symtab_strtab(_list, index_start_header, index_header_names,
size_section_header=40):
    index_shstrndx_data = convert_to_10_in_list_byte(_list, index_header_names + 16,
4)
    symtab = 0
    strtab = 0
    text = 0
    i = 0
    while not symtab or not strtab or not text:
        name = ""
        index_now = index_shstrndx_data + convert_to_10_in_list_byte(_list,
index_start_header + i * size_section_header, 4)
        while _list[index_now] != 0:
            name += chr(_list[index_now])
            index_now += 1
        if name == ".text":
            text = index_start_header + i * size_section_header
        elif name == ".symtab":
            symtab = index_start_header + i * size_section_header
        elif name == ".strtab":
            strtab = index_start_header + i * size_section_header
        i += 1
    return text, symtab, strtab


def disassembler(input, output):
    with open(input, 'rb') as file_input:
        _list = []
        lines = file_input.readlines()
        for i in lines:
            _list.extend(list(i))
        ELF = chr(_list[1]) + chr(_list[2]) + chr(_list[3])
        if ELF != "ELF":
            raise TypeError("Type Error")
        e_shoff = convert_to_10_in_list_byte(_list, 32, 4) # index start header
        e_shstrndx = 40 * convert_to_10_in_list_byte(_list, 50, 2) + e_shoff  # index
header names
        size_section_header = 40
        index_text_header, index_symtab_header, index_strtab_header =
search_index_text_symtab_strtab(_list, e_shoff, e_shstrndx, size_section_header)

        size_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header + 20,
4)
        index_data_symtab = convert_to_10_in_list_byte(_list, index_symtab_header +
16, 4)
        index_data_strtab = convert_to_10_in_list_byte(_list, index_strtab_header +
16, 4)
```

```
        dictionary_index_mark = {}
        list_item_symtab = []
        for i in range(size_data_symtab // 16):
            list_item_symtab.append(SymbolTableSection(_list[index_data_symtab + i *
16:index_data_symtab + (i + 1) * 16]))
        symtab_out = ["Symbol Value              Size Type     Bind    Vis
Index Name\n"]
        for i in range(size_data_symtab // 16):
            now = list_item_symtab[i]
            index_name = index_data_strtab + now.get_name()
            name = ""
            while _list[index_name] != 0:
                name += chr(_list[index_name])
                index_name += 1
            symtab_out.append("[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\n" % (
                   i, now.get_value(), now.get_size(), now.get_type(), now.get_bind(),
now.get_vis(), now.get_index(), name))
            if now.get_type() == "FUNC":
                dictionary_index_mark[now.get_value()] = name

        # text
        virtual_addr_text = convert_to_10_in_list_byte(_list, index_text_header + 12,
4)
        index_data_text = convert_to_10_in_list_byte(_list, index_text_header + 16, 4)
        size_data_text = convert_to_10_in_list_byte(_list, index_text_header + 20, 4)
        count_url = 0
        list_commands = []
        for i in range(size_data_text // 4):
            command_bin = bin(convert_to_10_in_list_byte(_list, index_data_text + i *
4, 4))[2:]
            command_bin = (32 - len(command_bin)) * "0" + command_bin
            opcode = command_bin[-7:]
            type_command = dictionary_command_type[opcode]
            try:
                if type_command == "R":
                    command = CommandTypeR(virtual_addr_text + i * 4, command_bin)
                elif type_command == "I":
                    command = CommandTypeI(virtual_addr_text + i * 4, command_bin)
                elif type_command == "S":
                    command = CommandTypeS(virtual_addr_text + i * 4, command_bin)
                elif type_command == "B":
                    command = CommandTypeB(virtual_addr_text + i * 4, command_bin)
                    if command.addr_url not in dictionary_index_mark:
                        dictionary_index_mark[command.addr_url] = "L" + str(count_url)
                        count_url += 1
                    command.set_name_url(dictionary_index_mark[command.addr_url])
                elif type_command == "U":
                    command = CommandTypeU(virtual_addr_text + i * 4, command_bin)
                elif type_command == "J":
                    command = CommandTypeJ(virtual_addr_text + i * 4, command_bin)
                    if command.addr_url not in dictionary_index_mark:
                        dictionary_index_mark[command.addr_url] = "L" + str(count_url)
                        count_url += 1
                    command.set_name_url(dictionary_index_mark[command.addr_url])
                else:
                    command = "unknown_instruction"
            except:
                command = "unknown_instruction"
            list_commands.append(command)
```

```
        with open(output, "w") as file_output:
            file_output.write(".text\n")
            for i in range(size_data_text // 4):
                if virtual_addr_text + i * 4 in dictionary_index_mark:
                    file_output.write("\n")
                    file_output.write("%08x   <%s>:\n" % (virtual_addr_text + i * 4,
dictionary_index_mark[virtual_addr_text + i * 4]))
                file_output.write(list_commands[i].__str__() + "\n")
            file_output.write("\n")
            file_output.write(".symtab\n")
            for i in symtab_out:
                file_output.write(i)


if __name__ == "__main__":
    if len(sys.argv) == 3:
        try:
            disassembler(sys.argv[1], sys.argv[2])
            print("Complete!")
        except TypeError as error:
            print("Error, does not match the ELF file ", error)
        except Exception as error:
            print("Error", error)
    else:
        print("Error! Expect 2 argument, actual: ", len(sys.argv) - 1)
```

Command.py

```
from convert import convert_to_10_in_list_byte
from math import pow

dictionary_register_risc5 = {
    0: "zero",
    1: "ra",
    2: "sp",
    3: "gp",
    4: "Tp",
    5: "t0",
    6: "t1",
    7: "t2",
    8: "s0",
    9: "s1",
    10: "a0",
    11: "a1",
    12: "a2",
    13: "a3",
    14: "a4",
    15: "a5",
    16: "a6",
    17: "a7",
    18: "s2",
    19: "s3",
    20: "s4",
    21: "s5",
    22: "s6",
```

```
    23: "s7",
    24: "s8",
    25: "s9",
    26: "s10",
    27: "s11",
    28: "t3",
    29: "t4",
    30: "t5",
    31: "t6"
}
dictionary_command_type = {
    "0110111": "U",
    "0010111": "U",
    "1101111": "J",
    "1100111": "I",
    "1100011": "B",
    "0000011": "I",
    "0100011": "S",
    "0010011": "I",
    "0110011": "R",
    "0001111": "F", # Fence?
    "1110011": "I",
}
dictionary_command_I = {
    ("1100111", "000"): "jalr",
    ("0000011", "000"): "lb",
    ("0000011", "001"): "lh",
    ("0000011", "010"): "lw",
    ("0000011", "100"): "lbu",
    ("0000011", "101"): "lhu",
    ("0010011", "000"): "addi",
    ("0010011", "010"): "slti",
    ("0010011", "011"): "sltiu",
    ("0010011", "100"): "xori",
    ("0010011", "110"): "ori",
    ("0010011", "111"): "andi",
    ("1110011", "000"): "ebreak",
    ("0010011", "001"): "slli",
    ("0010011", "101"): "srli",
}
dictionary_command_R = {
    ("0110011", "000", "0000000"): "add",
    ("0110011", "000", "0100000"): "sub",
    ("0110011", "001", "0000000"): "sll",
    ("0110011", "010", "0000000"): "slt",
    ("0110011", "011", "0000000"): "sltu",
    ("0110011", "100", "0000000"): "xor",
    ("0110011", "101", "0000000"): "srl",
    ("0110011", "101", "0100000"): "sra",
    ("0110011", "110", "0000000"): "or",
    ("0110011", "111", "0000000"): "and",
    ("0110011", "000", "0000001"): "mul",
    ("0110011", "001", "0000001"): "mulh",
    ("0110011", "010", "0000001"): "mulhsu",
    ("0110011", "011", "0000001"): "mulhu",
    ("0110011", "100", "0000001"): "div",
    ("0110011", "101", "0000001"): "divu",
    ("0110011", "110", "0000001"): "rem",
    ("0110011", "111", "0000001"): "remu"
```

```
}
dictionary_command_S = {
    ("0100011", "000"): "sb",
    ("0100011", "001"): "sh",
    ("0100011", "010"): "sw"
}
dictionary_command_B = {
    ("1100011", "000"): "beq",
    ("1100011", "001"): "bne",
    ("1100011", "100"): "blt",
    ("1100011", "101"): "bge",
    ("1100011", "110"): "bltu",
    ("1100011", "111"): "bgeu",
}
dictionary_command_U = {
    "0110111": "lui",
    "0010111": "auipc"
}
dictionary_command_J = {
    "1101111": "jal"
}


def number_dop(data):
    number = -int(data[0]) * int(pow(2, (len(data) - 1)))
    data = data[1::][::-1]
    for i in range(len(data)):
        number += int(data[i]) * int(pow(2, i))
    return number


class CommandTypeI:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.funct3 = data[-15:-12]
        self.name = dictionary_command_I[(self.opcode, self.funct3)]
        if self.name == "ebreak":
            if int(data[-32:-20], 2) == 0:
                self.name = "ecall"
        self.rd = dictionary_register_risc5[int(data[-12:-7], 2)]
        self.rs1 = dictionary_register_risc5[int(data[-20:-15], 2)]
        if self.name in ["srli", "slli"]:
            self.imm = int(data[-25:-20])
            if int(data[-32:-25]) != 0 and self.name == "slli":
                self.name = "srai"
        else:
            self.imm = number_dop(data[-32] * 21 + data[-31:-20])

    def __str__(self):
        text = ""
        if self.name in ["lh", "lw", "lbu", "lhu", "lb", "jalr"]:
            args = (self.addr, self.value, self.name, self.rd, self.imm, self.rs1)
            text = "   %05x:\t%08x\t%7s\t%s, %s(%s)" % args
        elif self.name in ["ebreak", "ecall"]:
            args = (self.addr, self.value, self.name)
            text = "   %05x:\t%08x\t%7s" % args
        else:
```

```
            args = (self.addr, self.value, self.name, self.rd, self.rs1, self.imm)
            text = "   %05x:\t%08x\t%7s\t%s, %s, %s" % args
        return text


class CommandTypeR:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.funct3 = data[-15:-12]
        self.funct7 = data[-32:-25]
        self.name = dictionary_command_R[(self.opcode, self.funct3, self.funct7)]
        self.rd = dictionary_register_risc5[int(data[-12:-7], 2)]
        self.rs1 = dictionary_register_risc5[int(data[-20:-15], 2)]
        self.rs2 = dictionary_register_risc5[int(data[-25:-20], 2)]

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rd, self.rs1, self.rs2)
        text = "   %05x:\t%08x\t%7s\t%s, %s, %s" % args
        return text


class CommandTypeS:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.funct3 = data[-15:-12]
        self.name = dictionary_command_S[(self.opcode, self.funct3)]
        self.rs1 = dictionary_register_risc5[int(data[-20:-15], 2)]
        self.rs2 = dictionary_register_risc5[int(data[-25:-20], 2)]
        self.imm = number_dop(data[-32] * 21 + data[-31:-25] + data[-12:-7])

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rs2, self.imm, self.rs1)
        text = "   %05x:\t%08x\t%7s\t%s, %s(%s)" % args
        return text


class CommandTypeB:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.funct3 = data[-15:-12]
        self.name = dictionary_command_B[(self.opcode, self.funct3)]
        self.rs1 = dictionary_register_risc5[int(data[-20:-15], 2)]
        self.rs2 = dictionary_register_risc5[int(data[-25:-20], 2)]
        self.imm = number_dop(data[-32] * 20 + data[-8] + data[-31:-25] + data[-12:-8]
+ "0")
        self.addr_url = self.imm + addr
        self.name_url = ""

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rs1, self.rs2, self.addr_url,
self.name_url)
        text = "   %05x:\t%08x\t%7s\t%s, %s, %x <%s>" % args
        return text
```

```
    def set_name_url(self, name):
        self.name_url = name


class CommandTypeU:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.rd = dictionary_register_risc5[int(data[-12:-7], 2)]
        self.name = dictionary_command_U[self.opcode]
        self.imm = number_dop(data[-32:-12])

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rd, self.imm)
        text = "   %05x:\t%08x\t%7s\t%s, 0x%x" % args
        return text


class CommandTypeJ:
    def __init__(self, addr, data):
        self.addr = addr
        self.value = int(data, 2)
        self.opcode = data[-7:]
        self.rd = dictionary_register_risc5[int(data[-12:-7], 2)]
        self.name = dictionary_command_J[self.opcode]
        self.imm = number_dop(data[-32] * 12 + data[-20:-12] + data[-20] + data[-31:-
25] + data[-25:-21] + "0")
        self.addr_url = addr + self.imm
        self.name_url = ""

    def __str__(self):
        args = (self.addr, self.value, self.name, self.rd, self.imm, self.name_url)
        text = "   %05x:\t%08x\t%7s\t%s, %s <%s>" % args
        return text

    def set_name_url(self, name_url):
        self.name_url = name_url
```

SymbolTableSection.py

```
from convert import convert_to_10_in_list_byte

dictionary_type_symtab = {
    0: "NOTYPE",
    1: "OBJECT",
    2: "FUNC",
    3: "SECTION",
    4: "FILE",
    5: "COMMON",
    6: "TLS",
    10: "LOOS",
    12: "HIOS",
    13: "LOPROC",
```

```
    14: "SPARC_REGISTER",
    15: "HIPROC"
}
dictionary_bind_symtab = {
    0: "LOCAL",
    1: "GLOBAL",
    2: "WEAK",
    10: "LOOS",
    12: "HIOS",
    13: "LOPROC",
    15: "HIPROC"
}
dictionary_vis_symtab = {
    0: "DEFAULT",
    1: "INTERNAL",
    2: "HIDDEN",
    3: "PROTECTED",
    4: "EXPORTED",
    5: "SINGLETON",
    6: "ELIMINATE"
}
dictionary_index_symtab = {
    0: "UNDEF",
    0xff00: "LOPROC",
    0xff1f: "HIPROC",
    0xfff1: "ABS",
    0xfff2: "COMMON",
    0xffff: "HIRESERVE"
}


class SymbolTableSection:
    def __init__(self, data):
        self.name = convert_to_10_in_list_byte(data, 0, 4)
        self.value = convert_to_10_in_list_byte(data, 4, 4)
        self.size = convert_to_10_in_list_byte(data, 8, 4)
        self.info = convert_to_10_in_list_byte(data, 12, 1)
        self.other = convert_to_10_in_list_byte(data, 13, 1)
        self.shndx = convert_to_10_in_list_byte(data, 14, 2)
        self.type = self.info & 15
        self.bind = self.info >> 4
        self.vis = self.other & 3

    def get_value(self):
        return self.value

    def get_size(self):
        return self.size

    def get_type(self):
        return dictionary_type_symtab[self.type]

    def get_bind(self):
        return dictionary_bind_symtab[self.bind]

    def get_index(self):
        return dictionary_index_symtab[self.shndx] if self.shndx in
dictionary_index_symtab else self.shndx
```

```
    def get_vis(self):
        return dictionary_vis_symtab[self.vis]

    def get_name(self):
        return self.name
```

convert.py

```
def convert_to_10_in_list_byte(_list, index, size):
    num = ""
    vector = _list[index: index + size][::-1]  # little endian
    for i in vector:
        now = hex(i)[2:]
        now = (2 - len(now)) * "0" + now
        num += now
    return int(num, 16)
```