Лабораторная работа

Барышева Ксения

Лабораторная работа	Группа 01	2023
ISA	Барышева Ксения	

Постановка задачи

Написана программа-транслятор (дизассемблер), с помощью которой можно преобразовывать машинный код (извлеченный из elf-файла) в текст программы на языке ассемблера. Поддерживаться следующий набор команд RISC-V: RV32I, RV32M.

Кодирование: little endian. Вывод регистров: ABI. Регистр x8 выводится как s0. Псевдонимы команд: псевдонимы команд парсить не нужно. Обрабатывать нужно только секции .text, .symtab.

Программа выполнена на python3.11

Сделано всё задание

Ссылка на репозиторий

https://github.com/the-coolest-skeleton/parsing-ELF

Описание программы

Приведённые ниже строчки считывают из аргументов командной строки входной elf-file и, записывают ответ в выходной файл, если он подан, или в консоль.

Для elf-file создан класс. В классе хранится некоторая часть содержимого файла. Функции `elf.symtab()` и `elf.text()` возвращают строки, в которых записаны соответсвующие секции.

```
if __name__ == '__main__':
    sys.argv[1:]

with open(sys.argv[1], 'rb') as our_elf_file:
    bits = our_elf_file.read()

elf = elf_file(bits)
st_sym = elf.symtab()
st_text = elf.text()
answer = st_text + '\n' + st_sym

if len(sys.argv) == 3:
    with open(sys.argv[2], 'w') as outfile:
        outfile.write(answer)
else:
    print(answer)
```

Так же в программе используются словари, для раскодировки команд

```
dict_type: dict[int, str] = {
    0: "NOTYPE",
    1: "OBJECT",
    2: "FUNC",
    3: "SECTION",
    4: "FILE",
    5: "COMMON",
    6: "TLS",
    10: "LOOS",
    12: "HIOS",
    13: "LOPROC",
    15: "HIPROC"
}
dict_vis: dict[int, str] = {
    0: "DEFAULT",
    1: "INTERNAL",
    2: "HIDDEN",
    3: "PROTECTED"}

dict_bind: dict[int, str] = {
    0: "LOCAL",
    1: "GLOBAL",
    2: "WEAK",
    10: "LOOS",
    12: "HIOS",
    13: "LOPROC",
    15: "HIPROC"}

dict_index: dict[int, str] = {
    0: "UNDEF",
    65280: "LOPROC",
    65311: "HIPROC",
    65312: "LOOS",
    65343: "HIOS",
    65521: "ABS",
    65522: "COMMON",
    65535: "HIRESERVE"
}

names_func = {"0": 0}
count_reg = 0
reg = {
    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"}
```

Описание класса title

```
class title:
    def __init__(self, our_bytes, index):
        self.p_type = our_bytes[index:index + 4]
        index += 4
        self.p_offset = our_bytes[index:index + 4]
        index += 4
        self.p_vaddr = our_bytes[index:index + 4]
        index += 4
        self.p_paddr = our_bytes[index:index + 4]
        index += 4
        self.p_filesz = our_bytes[index:index + 4]
        index += 4
        self.memsz = our_bytes[index:index + 4]
        index += 4
        self.p_flags = our_bytes[index:index + 4]
        index += 4
        self.p_align = our_bytes[index:index + 4]
```

Данный класс описывает структуру заголовка, то есть одного элемента из таблицы заголовков elf-file. Переменные our_bytes - строка байтов, считываемая из elf-file, index - адрес начала читаемого заголовка(относительно начала файла). Каждая переменная данного класса названа так же, как в статье об elf-file в Википедии.

Таблица заголовков программы

Таблица заголовков программы содержит заголовки, каждый из которых описывает отдельный сегмент программы и его атрибуты либо другую информацию, необходимую операционной системе для подготовки программы к исполнению. Данная таблица может располагаться в любом месте файла, её местоположение (смещение относительно начала файла) описывается в поле `e_phoff` заголовка ELF.

При анализе структуры заголовка программы можно обнаружить различное местоположение поля `p_flags` для 32- и 64-битных ELF файлов. Данное различие обуславливается выравниванием структуры для увеличения эффективности обработки.

Поля заголовка программы

<table>
<tr><th colspan="2">Размер</th><th rowspan="2">Название</th><th rowspan="2">Назначение</th></tr>
<tr><th>ELF 32</th><th>ELF 64</th></tr>
<tr><td>4</td><td></td><td><code>p_type</code></td><td>Тип сегмента, который описывает данный заголовок, или каким образом интерпретировать значения полей этого заголовка.
<table>
<tr><th>Название</th><th>Значение</th><th>Описание</th></tr>
<tr><td><code>PT_NULL</code></td><td>0</td><td>Заголовок не используется, остальные поля не определены. Данный тип позволяет включать в таблицу заголовков программы файла игнорируемые элементы.</td></tr>
<tr><td><code>PT_LOAD</code></td><td>1</td><td>Загружаемый сегмент, описываемый полями <code>p_filesz</code> и <code>p_memsz</code>. Байты из файла отражаются на сегменте в памяти. Если размер сегмента в памяти (<code>p_memsz</code>) больше размера сегмента в файле (<code>p_filesz</code>), дополнительные байты заполняются нулями (они следуют сразу за определенными в сегменте байтами). Размер сегмента в файле (<code>p_filesz</code>) не может быть больше размера сегмента в памяти (<code>p_memsz</code>). Заголовки программы загружаемых сегментов располагаются в таблице заголовков программ в порядке возрастания значения поля <code>p_vaddr</code> .</td></tr>
<tr><td><code>PT_DYNAMIC</code></td><td>2</td><td>Заголовок программы предоставляет информацию о динамической компоновке.</td></tr>
<tr><td><code>PT_INTERP</code></td><td>3</td><td>Заголовок программы предоставляет размер и местоположение пути (строки в стиле C с завершающим нулём) для запуска в качестве интерпретатора. Этот тип сегмента имеет смысл только для исполняемых файлов (хотя он может быть и в совместно используемом объектном файле); он не может встречаться более одного раза в файле. Если заголовок такого типа присутствует, он должен предшествовать любому заголовку программы загружаемого сегмента.</td></tr>
<tr><td><code>PT_NOTE</code></td><td>4</td><td>Заголовок программы определяет местоположение и размер вспомогательной информации.</td></tr>
<tr><td><code>PT_SHLIB</code></td><td>5</td><td>Этот тип сегмента зарезервирован, но его смысл не определён. Программы, содержащие заголовок программы этого типа, не</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table>
<tr><td><code>PT_NOTE</code></td><td>4</td><td>Заголовок программы определяет местоположение и размер вспомогательной информации.</td></tr>
<tr><td><code>PT_SHLIB</code></td><td>5</td><td>Этот тип сегмента зарезервирован, но его смысл не определён. Программы, содержащие заголовок программы этого типа, не соответствуют ABI.</td></tr>
<tr><td><code>PT_PHDR</code></td><td>6</td><td>Заголовок программы, если он присутствует, определяет местоположение и размер самой таблицы заголовков программы, как в файле, так и в образе памяти программы. Этот тип сегмента не может встречаться более одного раза в файле. Более того, он может встретиться только при наличии в файле таблицы заголовков программы. Если заголовок такого типа присутствует, он должен предшествовать любому заголовку программы загружаемого сегмента.</td></tr>
<tr><td><code>PT_TLS</code></td><td>7</td><td>Заголовок программы определяет шаблон Thread-Local Storage. Загрузчики ELF не должны поддерживать эту запись в таблице заголовков программ.</td></tr>
<tr><td><code>PT_LOOS</code> - <code>PT_HIOS</code></td><td>1610612736 - 1879048191</td><td>Зависимые от операционной системы значения.</td></tr>
<tr><td><code>PT_LOPROC</code> - <code>PT_HIPROC</code></td><td>1879048192 - 2147483647</td><td>Зависимые от процессора значения.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>4</td><td><code>p_flags</code></td><td>Флаги, относящиеся к сегменту (для ELF64).
<table>
<tr><th>Название</th><th>Значение</th><th>Описание</th></tr>
<tr><td><code>PF_X</code></td><td><code>0x1</code></td><td>Разрешение на исполнение</td></tr>
<tr><td><code>PF_W</code></td><td><code>0x2</code></td><td>Разрешение на запись</td></tr>
<tr><td><code>PF_R</code></td><td><code>0x4</code></td><td>Разрешение на чтение</td></tr>
<tr><td><code>PF_MASKOS</code></td><td><code>0x0ff00000</code></td><td>Все биты, включенные в это поле, определяют зависящие от операционной системы значения</td></tr>
<tr><td><code>PF_MASKPROC</code></td><td><code>0xf0000000</code></td><td>Все биты, включенные в это поле, определяют зависящие от процессора значения</td></tr>
</table></td></tr>
<tr><td>4</td><td>8</td><td><code>p_offset</code></td><td>Смещение сегмента от начала файла.</td></tr>
<tr><td>4</td><td>8</td><td><code>p_vaddr</code></td><td>Виртуальный адрес сегмента в памяти, куда должен быть загружен сегмент при отображении в память.</td></tr>
<tr><td>4</td><td>8</td><td><code>p_paddr</code></td><td>Физический адрес сегмента (для систем, в которых он важен).</td></tr>
<tr><td>4</td><td>8</td><td><code>p_filesz</code></td><td>Размер сегмента в файле. Может быть нулевым.</td></tr>
<tr><td>4</td><td>8</td><td><code>p_memsz</code></td><td>Размер сегмента в памяти. Может быть нулевым.</td></tr>
<tr><td>4</td><td></td><td><code>p_flags</code></td><td>Флаги, относящиеся к сегменту (для ELF32) (возможные значения см. выше).</td></tr>
<tr><td>4</td><td>8</td><td><code>p_align</code></td><td>Выравнивание сегмента. <code>0</code> и <code>1</code> определяют отсутствие выравнивания. В противном случае должно быть положительной двойкой в определённой степени.</td></tr>
</table>

Описание класса Section

```
class section:
   def __init__(self, our_bytes):
       self.sh_name = our_bytes[0:4]
       self.sh_type = our_bytes[4:8]
       self.sg_flags = our_bytes[8:12]
       self.sh_addr = our_bytes[12:16]
       self.sh_offset = our_bytes[16:20]
       self.sh_size = our_bytes[20:24]
       self.sh_link = our_bytes[24:28]
       self.sh_info = our_bytes[28:32]
       self.sh_addralign = our_bytes[32:36]
       self.sh_entsize = our_bytes[36:40]
```

Данный класс отвечает за хранение одной секции, то есть одного элемента таблицы заголовков секций. Тут переменная our_bytes это уже не всё содержимое elf-file, а только содержимое данной секции. Все переменные названы так же, как в статье из Википедии

Таблица заголовков секций

Таблица заголовков секций содержит атрибуты секций файла. Данная таблица необходима только компоновщику, исполняемые файлы в наличии этой таблицы не нуждаются (ELF загрузчик её игнорирует). Предоставленную в таблице заголовков секций информацию компоновщик использует для оптимального размещения данных секций по сегментам при сборке файла с учётом их атрибутов.

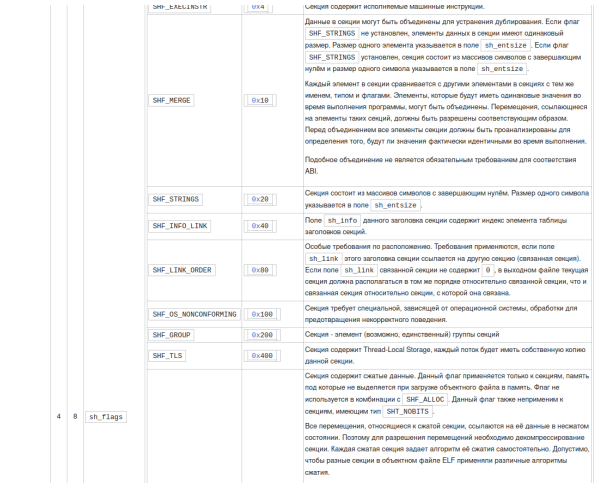
Поля заголовка секции

Размер ELF 32	Размер ELF 64	Название	Назначение
4		sh_name	Смещение строки, содержащей название данной секции, относительно начала таблицы названий секций.
4		sh_type	Тип заголовка.

Название	Значение	Описание
SHT_NULL	0	Заголовок не используется, остальные поля не определены.
SHT_PROGBITS	1	Секция содержит информацию, определённую программой, её формат и значение определяется программой единолично.
SHT_SYMTAB	2	Секция содержит таблицу символов. В настоящий момент в файле может быть только одна такая секция.
SHT_STRTAB	3	Секция содержит таблицу строк. Файл может иметь множество секций такого типа.
SHT_RELA	4	Секция содержит расширенную информацию о перемещениях. Файл может иметь множество секций такого типа.
SHT_HASH	5	Секция содержит таблицу хэшей символов. В настоящий момент в файле может быть только одна такая секция.
SHT_DYNAMIC	6	Секция содержит информацию о динамической компоновке. В настоящий момент в файле может быть только одна такая секция.
SHT_NOTE	7	Секция содержит информацию, которая каким-то образом отмечает файл.
SHT_NOBITS	8	Секция не занимает места в файле, в противном случае схожа с SHT_PROGBITS.
SHT_REL	9	Секция содержит информацию о перемещениях. Файл может иметь множество секций такого типа.
SHT_SHLIB	10	Данный тип секции определён, но не имеет определённого значения.
SHT_DYNSYM	11	Секция содержит таблицу символов. В настоящий момент в файле может быть только одна такая секция.
SHT_INIT_ARRAY	14	Секция содержит массив указателей на функции инициализации программы. Функции не

		SHT_SHLIB	10
		SHT_DYNSYM	11
		SHT_INIT_ARRAY	14
4		sh_type	
		SHT_FINI_ARRAY	15
		SHT_PREINIT_ARRAY	16
		SHT_GROUP	17
		SHT_SYMTAB_SHNDX	18
		SHT_LOOS - SHT_HIOS	1610612736 - 1879048191
		SHT_LOPROC - SHT_HIPROC	1879048192 - 2147483647
		SHT_LOUSER - SHT_HIUSER	2147483648 - 4294967295

Название	Значение	Описание
SHF_WRITE	0x1	Разрешение на запись.
SHF_ALLOC	0x2	Секция занимает память во время выполнения процесса. Некоторые служебные секции не загружаются в память при загрузке объектного файла, для таких секций



			SHF_EXECINSTR	0x4	Секция содержит исполняемые машинные инструкции.
			SHF_MERGE	0x10	Данные в секции могут быть объединены для устранения дублирования. Если флаг SHF_STRINGS не установлен, элементы данных в секции имеют одинаковый размер. Размер одного элемента указывается в поле sh_entsize. Если флаг SHF_STRINGS установлен, секция состоит из массивов символов с завершающим нулём и размер одного символа указывается в поле sh_entsize. Каждый элемент в секции сравнивается с другими элементами в секциях с тем же именем, типом и флагами. Элементы, которые будут иметь одинаковые значения во время выполнения программы, могут быть объединены. Перемещения, ссылающиеся на элементы таких секций, должны быть разрешены соответствующим образом. Перед объединением все элементы секции должны быть проанализированы для определения того, будут ли значения фактически идентичными во время выполнения. Подобное объединение не является обязательным требованием для соответствия ABI.
			SHF_STRINGS	0x20	Секция состоит из массивов символов с завершающим нулём. Размер одного символа указывается в поле sh_entsize.
			SHF_INFO_LINK	0x40	Поле sh_info данного заголовка секции содержит индекс элемента таблицы заголовков секций.
			SHF_LINK_ORDER	0x80	Особые требования по расположению. Требования применяются, если поле sh_link этого заголовка секции ссылается на другую секцию (связанная секция). Если поле sh_link связанной секции не содержит 0, в выходном файле текущая секция должна располагаться в том же порядке относительно связанной секции, что и связанная секция относительно секции, с которой она связана.
			SHF_OS_NONCONFORMING	0x100	Секция требует специальной, зависящей от операционной системы, обработки для предотвращения некорректного поведения.
			SHF_GROUP	0x200	Секция - элемент (возможно, единственный) группы секций
			SHF_TLS	0x400	Секция содержит Thread-Local Storage, каждый поток будет иметь собственную копию данной секции.
4	8	sh_flags			Секция содержит сжатые данные. Данный флаг применяется только к секциям, память под которые не выделяется при загрузке объектного файла в память. Флаг не используется в комбинации с SHF_ALLOC. Данный флаг также неприменим к секциям, имеющим тип SHT_NOBITS. Все перемещения, относящиеся к сжатой секции, ссылаются на её данные в несжатом состоянии. Поэтому для разрешения перемещений необходимо декомпрессирование секции. Каждая сжатая секция задает алгоритм её сжатия самостоятельно. Допустимо, чтобы разные секции в объектном файле ELF применяли различные алгоритмы сжатия.
					ELFCOMPRESS_HIPROC 2147483647 значения.
				4	ch_reserved	Зарезервировано для будущего использования.
				4	8	ch_size	Размер в байтах декомпрессированной секции.
				4	8	ch_addralign	Необходимое для декомпрессированной секции выравнивание.
			SHF_MASKOS	0x0ff00000	Все биты, включенные в это поле, определяют зависящие от операционной системы значения
			SHF_MASKPROC	0xf0000000	Все биты, включенные в это поле, определяют зависящие от процессора значения
4	8	sh_addr	Если секция должна быть загружена в память при загрузке объектного файла, это поле указывает адрес, начиная с которого секция будет загружена, в противном случае поле содержит 0.			
4	8	sh_offset	Смещение секции от начала файла в байтах. Секции типа SHT_NOBITS не занимают места в файле, для них данное поле содержит концептуальное местоположение в файле.			
4	8	sh_size	Размер секции в файле. Может быть нулевым.			
4		sh_link	Индекс ассоциированной секции. Данное поле может иметь различное предназначение в зависимости от типа заголовка. **Интерпретация полей sh_link и sh_info в зависимости от значения поля sh_type** sh_type / sh_link / sh_info SHT_DYNAMIC — Индекс заголовка секции таблицы строк, которая используется элементами данной секции. — 0 SHT_HASH — Индекс заголовка секции таблицы символов, к которой относится данная таблица хэшей. — 0 SHT_REL, SHT_RELA — Индекс заголовка секции ассоциированной таблицы символов. — Индекс заголовка секции, к которой должны быть применены данные перемещения. SHT_SYMTAB, SHT_DYNSYM — Индекс заголовка секции ассоциированной таблицы строк. — На один больше, чем индекс последнего локального символа (STB_LOCAL) в таблице символов. SHT_GROUP — Индекс заголовка секции ассоциированной таблицы символов. — Индекс элемента в ассоциированной таблице символов. Имя указанного элемента предоставляет сигнатуру группы секций. SHT_SYMTAB_SHNDX — Индекс заголовка секции ассоциированной секции таблицы символов. — 0			
4		sh_info	Дополнительная информация о секции (возможные значения см. выше).			
4	8	sh_addralign	Необходимое выравнивание секции.			
4	8	sh_entsize	Размер в байтах каждой записи (в том случае, если секция содержит массив записей фиксированного размера, в противном случае поле содержит 0).			

Описание класса elf_file

Инициализация

```
class elf_file:
    def __init__(self, our_bytes):
        self.our_bytes = our_bytes
        # Заголовок elf-file
        self.e_ident = our_bytes[:16]
        self.e_type = our_bytes[16:18]
        self.e_machine = our_bytes[18:20]
        self.e_version = our_bytes[20:24]
        self.e_entry = our_bytes[24:28]
        self.e_phoff = our_bytes[28:32]
        self.e_shoff = our_bytes[32:36]
        self.flags = our_bytes[36:40]
        self.e_ehsize = our_bytes[40:42]
        self.e_phentsize = our_bytes[42:44]
        self.e_phnum = our_bytes[44:46]
        self.e_shentsize = our_bytes[46:48]
        self.e_shnum = our_bytes[48:50]
```

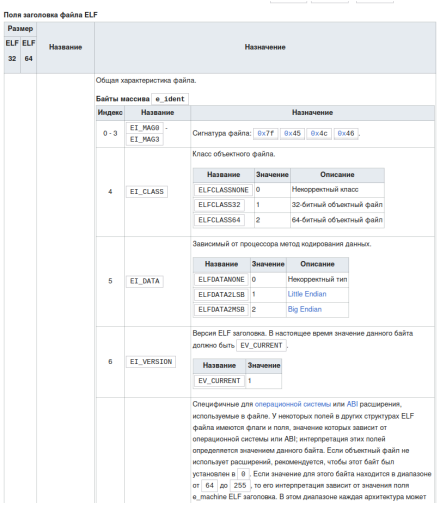
```
self.e_shstrndx = our_bytes[50:52]

# Поля заголовка программы
self.titeles = list()
index = int.from_bytes(self.e_phoff, 'little')
for i in range(int.from_bytes(self.e_phnum, 'little')):
    value = title(our_bytes, index)
    index += int.from_bytes(self.e_phentsize, 'little')
    self.titeles.append(value)

# Поля заголовка секции
self.page_title = list()
index = int.from_bytes(self.e_shoff, 'little')
len_title = int.from_bytes(self.e_shentsize, 'little')
for i in range(int.from_bytes(self.e_shnum, 'little')):
    value = section(our_bytes[index: index + len_title])
    index += len_title
    self.page_title.append(value)

# Специальный адрес
self.min_address = 0
```

Выше приведена инициализация элемента данного класса. Заголовок elf-file раскодируется в первой части инициализации(переменные названы так же, как в Википедии). Некоторые переменные из первого блока вызываются в дальнейшем. В переменной self.titeles хранится таблица заголовков программы. Она заполняется во второй части инициализации(переменные тоже названы так же, как в Википедии). Переменная self.page_title - хранит в себе таблицу заголовок секций. Переменная self.min_address хранит в себе минимальный адрес функции, вызываемой программой. она заполняется позже.



Поля заголовка файла ELF

Размер ELF 32	Размер ELF 64	Название	Назначение
			Общая характеристика файла. Байты массива e_ident Индекс / Название / Назначение 0 - 3 — EI_MAG0 - EI_MAG3 — Сигнатура файла: 0x7f 0x45 0x4c 0x46 4 — EI_CLASS — Класс объектного файла. Название / Значение / Описание ELFCLASSNONE / 0 / Некорректный класс ELFCLASS32 / 1 / 32-битный объектный файл ELFCLASS64 / 2 / 64-битный объектный файл 5 — EI_DATA — Зависимый от процессора метод кодирования данных. Название / Значение / Описание ELFDATANONE / 0 / Некорректный тип ELFDATA2LSB / 1 / Little Endian ELFDATA2MSB / 2 / Big Endian 6 — EI_VERSION — Версия ELF заголовка. В настоящее время значение данного байта должно быть EV_CURRENT. Название / Значение EV_CURRENT / 1 Специфичные для операционной системы или ABI расширения, используемые в файле. У некоторых полей в других структурах ELF файла имеются флаги и поля, значение которых зависит от операционной системы или ABI; интерпретация этих полей определяется значением данного байта. Если объектный файл не использует расширений, рекомендуется, чтобы этот байт был установлен в 0. Если значение для этого байта находится в диапазоне от 64 до 255, то его интерпретация зависит от значения поля e_machine ELF заголовка. В этом диапазоне каждая архитектура может

16	`e_ident[16]`	7	`EI_OSABI`	установлен в `0`. Если значение для этого байта находится в диапазоне от `64` до `255`, то его интерпретация зависит от значения поля e_machine ELF заголовка. В этом диапазоне каждая архитектура может определить свой набор значений. **Название** / **Значение** / **Описание** `ELFOSABI_NONE` / 0 / UNIX System V ABI `ELFOSABI_HPUX` / 1 / HP-UX `ELFOSABI_NETBSD` / 2 / NetBSD `ELFOSABI_GNU` / 3 / Файл использует расширения GNU ELF (GNU/Linux) `ELFOSABI_SOLARIS` / 6 / Solaris `ELFOSABI_AIX` / 7 / AIX `ELFOSABI_IRIX` / 8 / IRIX `ELFOSABI_FREEBSD` / 9 / FreeBSD `ELFOSABI_TRU64` / 10 / Tru64 UNIX `ELFOSABI_MODESTO` / 11 / Modesto `ELFOSABI_OPENBSD` / 12 / OpenBSD `ELFOSABI_OPENVMS` / 13 / OpenVMS `ELFOSABI_NSK` / 14 / Non-Stop Kernel `ELFOSABI_AROS` / 15 / Amiga Research OS `ELFOSABI_FENIXOS` / 16 / FenixOS `ELFOSABI_CLOUDABI` / 17 / CloudABI `ELFOSABI_OPENVOS` / 18 / OpenVOS 64 - 255 / Зависимые от процессора значения
		8	`EI_ABIVERSION`	Версия ABI.
		9	`EI_PAD`	Т. н. padding bytes (байты заполнения). Зарезервированные для будущего использования элементы массива `e_ident`. Обычно устанавливаются в `0`. Программы для чтения объектных файлов должны игнорировать их.
		10	`EI_PAD + 1`	
		11	`EI_PAD + 2`	
		12	`EI_PAD + 3`	
		13	`EI_PAD + 4`	
		14	`EI_PAD + 5`	
		15	`EI_PAD + 6`	

Тип файла.

2	`e_type`	`ET_NONE`	0	Неопределённый
		`ET_REL`	1	Перемещаемый файл
		`ET_EXEC`	2	Исполняемый файл
		`ET_DYN`	3	Совместно используемый объектный файл
		`ET_CORE`	4	Core file
		`ET_LOOS` - `ET_HIOS`	65024 - 65279	Зависимые от операционной системы значения
		`ET_LOPROC` - `ET_HIPROC`	65280 - 65535	Зависимые от процессора значения

Архитектура аппаратной платформы, для которой файл создан:

Название	Значение	Описание
`EM_NONE`	`0x0`	Неопределено
`EM_M32`	`0x01`	AT&T WE 32100
`EM_SPARC`	`0x02`	SPARC
`EM_386`	`0x03`	Intel 80386
`EM_68K`	`0x04`	Motorola 68000 (M68k)
`EM_88K`	`0x05`	Motorola 88000 (M88k)
`EM_IAMCU`	`0x06`	Intel MCU
`EM_860`	`0x07`	Intel 80860
`EM_MIPS`	`0x08`	MIPS
`EM_S370`	`0x09`	IBM_System/370
`EM_MIPS_RS3_LE`	`0x0A`	MIPS R3000 Little-endian
	`0x0B` - `0x0E`	Reserved for future use
`EM_PARISC`	`0x0F`	Hewlett-Packard PA-RISC
	`0x10`	Reserved for future use
`EM_960`	`0x13`	Intel 80960
`EM_PPC`	`0x14`	PowerPC
`EM_PPC64`	`0x15`	PowerPC (64-bit)
`EM_S390`	`0x16`	S390, including S390x
`EM_SPU`	`0x17`	IBM SPU/SPC
	`0x18` - `0x23`	Reserved for future use
`EM_V800`	`0x24`	NEC V800
`EM_FR20`	`0x25`	Fujitsu FR20
`EM_RH32`	`0x26`	TRW RH-32

2	e_machine	EM_OLD_ALPHA	0x29	Digital Alpha
		EM_SH	0x2A	SuperH
		EM_SPARCV9	0x2B	SPARC Version 9
		EM_TRICORE	0x2C	Siemens TriCore embedded processor
		EM_ARC	0x2D	Argonaut RISC Core
		EM_H8_300	0x2E	Hitachi H8/300
		EM_H8_300H	0x2F	Hitachi H8/300H
		EM_H8S	0x30	Hitachi H8S
		EM_H8_500	0x31	Hitachi H8/500
		EM_IA_64	0x32	IA-64
		EM_MIPS_X	0x33	Stanford MIPS-X
		EM_COLDFIRE	0x34	Motorola ColdFire
		EM_68HC12	0x35	Motorola M68HC12
		EM_MMA	0x36	Fujitsu MMA Multimedia Accelerator
		EM_PCP	0x37	Siemens PCP
		EM_NCPU	0x38	Sony nCPU embedded RISC processor
		EM_NDR1	0x39	Denso NDR1 microprocessor
		EM_STARCORE	0x3A	Motorola Star*Core processor
		EM_ME16	0x3B	Toyota ME16 processor
		EM_ST100	0x3C	STMicroelectronics ST100 processor
		EM_TINYJ	0x3D	Advanced Logic Corp. TinyJ embedded processor family
		EM_X86_64	0x3E	AMD x86-64
		EM_MCST_ELBRUS	0xAF	Эльбрус (процессорная архитектура)
		EM_TI_C6000	0x8C	TMS320C6000 Family
		EM_AARCH64	0xB7	ARM 64-bits (ARMv8/Aarch64)
		EM_RISCV	0xF3	RISC-V
		EM_BPF	0xF7	Berkeley Packet Filter
		EM_65816	0x101	WDC 65C816
4	e_version	Номер версии формата. На данный момент корректным считается только одно значение. **Название** / **Значение** / **Описание** EV_NONE / 0 / Некорректное значение EV_CURRENT / 1 / Текущая версия		

4	e_version	Номер версии формата. На данный момент корректным считается только одно значение. **Название** / **Значение** / **Описание** EV_NONE / 0 / Некорректное значение EV_CURRENT / 1 / Текущая версия		

4	8	e_entry	Виртуальный адрес точки входа, которому система передает управление при запуске процесса. Если у файла нет точки входа, это поле содержит 0.
4	8	e_phoff	Смещение таблицы заголовков программы от начала файла в байтах. Если у файла нет таблицы заголовков программы, это поле содержит 0.
4	8	e_shoff	Смещение таблицы заголовков секций от начала файла в байтах. Если у файла нет таблицы заголовков секций, это поле содержит 0.
4		e_flags	Связанные с файлом флаги, зависящие от процессора. При их отсутствии это поле содержит 0.
2		e_ehsize	Размер заголовка файла в байтах (52 для 32-битных файлов и 64 для 64-битных).
2		e_phentsize	Размер одного заголовка программы. Все заголовки программы имеют одинаковый размер (32 для 32-битных файлов и 56 для 64-битных).
2		e_phnum	Число заголовков программы. Если у файла нет таблицы заголовков программы, это поле содержит 0.
2		e_shentsize	Размер одного заголовка секции. Все заголовки секций имеют одинаковый размер (40 для 32-битных файлов и 64 для 64-битных).
2		e_shnum	Число заголовков секций. Если у файла нет таблицы заголовков секций, это поле содержит 0.
2		e_shstrndx	Индекс записи в таблице заголовков секций, описывающей таблицу названий секций (обычно эта таблица называется **.shstrtab** и представляет собой отдельную секцию). Если файл не содержит таблицы названий секций, это поле содержит 0.

Функция symtab

```
def symtab(self):
    answer_ = "\n.symtab\n"
    symtb = section("")
```

```
        address = 0
        is_found = False
        for sh in self.page_title:
            if int.from_bytes(sh.sh_type, 'little') == 2:
                symtb = sh
            if (int.from_bytes(sh.sh_type, 'little') == 3) and (not is_found):
                address = int.from_bytes(sh.sh_offset, 'little')
                is_found = True

         if int.from_bytes(symtb.sh_offset, 'little') + int.from_bytes(symtb.sh_size,
'little') > len(self.our_bytes):
            raise Exception
             information  =  self.our_bytes[int.from_bytes(symtb.sh_offset,  'little'):
int.from_bytes(symtb.sh_offset,
                                                                                'little')
+ int.from_bytes(
            symtb.sh_size, 'little')]
        size_str = int.from_bytes(symtb.sh_entsize, 'little')

        answer_ += (('\n%6s %-17s %5s %-8s %-8s %-8s %6s %s\n' % (
            "Symbol", "Value", "Size", "Type", "Bind", "Vis", "Index", "Name")))

        for i in range(len(information) // size_str):
            value = hex(int.from_bytes(information[4 + i * size_str:8 + size_str * i],
'little'))
              size = int.from_bytes(information[8 + i * size_str:12 + size_str * i],
'little')
            bind = dict_bind[information[12 + i * size_str] // 16]
            type_ = dict_type[information[12 + i * size_str] % 16]
            vis = dict_vis[information[13 + i * size_str]]
             index = int.from_bytes(information[14 + i * size_str:16 + i * size_str],
'little')
            if index in dict_index.keys():
                index = dict_index[index]
            name = ""
           ind = address + int.from_bytes(information[i * size_str: 4 + i * size_str],
'little')
            while self.our_bytes[ind] != 0:
                name += chr(self.our_bytes[ind])
                ind += 1
          answer_ += ('[%4i] 0x%-15s %5i %-8s %-8s %-8s %6s %s\n' % (i, value[2:].upper(),
size, type_, bind, vis, index, name))
            if type_ == "FUNC":
                names_func[value] = name
                if int(value, 16) < self.min_address or self.min_address == 0:
                    self.min_address = int(value, 16)
        return answer_
```

Данная функция возвращает строку, где написано содержимое .symtab. Чтобы найти в таблице self.page_title нужную секцию, мы смотрим на переменную заголовка sh.sh_typ. Она должна равняться 2. Далее, найдя адрес начала секции, конца и размера каждой линии, мы раскодируем каждую строку. Информация о каждой строке содержится в переменных, соответсвующим названиям столбцов: value, size, type, bind, vis, index, name. В этой же функции заполняется переменная self.min_address.

Функция text

```
    def text(self):
        answer = []
        index_in_list = int.from_bytes(self.e_shstrndx, 'little')
        our_title = self.page_title[index_in_list]
        address = int.from_bytes(our_title.sh_offset, 'little')
        text_section = title("", 0)

        for sh in self.page_title:
            name = ""
            ind = int.from_bytes(sh.sh_name, 'little')
            while self.our_bytes[ind + address] != 0:
                name += chr(self.our_bytes[ind + address])
                ind += 1
            if name == ".text":
                text_section = sh
                break

        offset = int.from_bytes(text_section.sh_offset, 'little')
        size = int.from_bytes(text_section.sh_size, 'little')
        link = int.from_bytes(text_section.sh_link, 'little')

        for i in range(size // 4):
            tb2 = transfer_hex(self.our_bytes[offset + i * 4:offset + 4 * (i + 1)])
            s = str(bin(int.from_bytes(self.our_bytes[offset + i * 4:offset + 4 * (i +
1)], 'little')))[2:]
            while len(s) < 32:
                s = "0" + s
            address = self.min_address + 4 * i
            answer.append(
              ['   %05s:\t%08s\t%s' % (str(hex(address))[2:], str(tb2), command_parce(s,
address)), address])
        answer_str = ".text\n"
        for i in range(len(answer)):
            if hex(answer[i][1]) in names_func:
                adr = hex(answer[i][1])[2:]
                while (len(adr)) < 8:
                    adr = "0" + adr
                  answer_str += ('\n%08s \t<%s>:\n' % (adr, names_func[hex(answer[i]
[1])]))
            answer_str += answer[i][0]
        return answer_str
```

Данная функция возвращает строку, хранящую в себе .text. Найдя адрес начала и конца данной секции(нужная секция из таюлицы находится, благодаря сравнению имени секции с .text), мы раскодируем каждую команду. Этим занимается функция command_parce(). Первый столбец, вычисляется с помощью переменной self.min_address. Второй столбец получается из байтов, взятых на этой итерации. Раскодировка каждой строки таблицы хаписывается в массив answer. При проходе по всей секции заполняется словарь names_func. Это словарь, который по адресу функции выдаёт её имя. А значению «0» в нём соответствует количество неизвестных меток, то есть меток $< Li >$.

```
def command_parce(bit_str, address):
    opcode = bit_str[25:32]
    reg_rd = bit_str[20:25]
    if opcode == "0110111":
                        return   '%7s\t%s,   %s\n'   %   ("lui",   reg[int(reg_rd,   2)],
str(conv_10to16(int((bit_str[0:20]), 2) - 2 ** 20 * int(bit_str[0]))))
    elif opcode == "0010111":
        return '%7s\t%s, %s\n' % ("auipc", reg[int(reg_rd, 2)],
                                            str(hex(int((bit_str[0:20]), 2) - 2 ** 20 *
int(bit_str[0]))))
    elif opcode == "1101111":
        jump = int(bit_str[0] + bit_str[12:20] + bit_str[11] + bit_str[1:11] + "0", 2)
- (2 ** 21) * int(
            bit_str[0])  # это прыжок
        address += jump
        if hex(address) not in names_func:
            name = "L" + str(names_func["0"])
            names_func[hex(address)] = name
            names_func["0"] += 1
       return '%7s\t%s, %s <%s>\n' % ("jal", reg[int(bit_str[20:25], 2)], hex(address),
names_func[hex(address)])
    elif opcode == "1100111":
              return  '%7s\t%s,  %d(%s)\n'  %  ("jalr",  reg[int(bit_str[20:25],  2)],
int(bit_str[:12], 2) - 2 ** 12 * int(bit_str[0]),
                                          reg[int(bit_str[12:17], 2)])
    elif opcode == "1100011":
        im = int(bit_str[0] + bit_str[24] + bit_str[1:7] + bit_str[20:24], 2) * 2 - 2
** 13 * int(bit_str[0])
        im += address
        if hex(im) in names_func:
            name = names_func[hex(im)]
        else:
            name = "L" + str(names_func["0"])
            names_func[hex(im)] = name
            names_func["0"] += 1

        r = reg[int(bit_str[7:12], 2)]
        comm = ""
        if bit_str[17:20] == "000":
            comm = "beq"
        elif bit_str[17:20] == "001":
            comm = "bne"
        elif bit_str[17:20] == "100":
            comm = "blt"
        elif bit_str[17:20] == "101":
            comm = "bge"
        elif bit_str[17:20] == "110":
            comm = "bltu"
        elif bit_str[17:20] == "111":
            comm = "bgeu"
          return '%7s\t%s, %s, %s, <%s>\n' % (comm, reg[int(bit_str[12:17], 2)], r,
hex(im), name)
    elif opcode == "0110011":
        r = reg[int(bit_str[7:12], 2)]
        command = ""
        if bit_str[:7] == "0000001":
```

```
            if bit_str[17:20] == "000":
                command = "mul"
            elif bit_str[17:20] == "001":
                command = "mulh"
            elif bit_str[17:20] == "010":
                command = "mulhsu"
            elif bit_str[17:20] == "011":
                command = "mulhu"
            elif bit_str[17:20] == "100":
                command = "div"
            elif bit_str[17:20] == "101":
                command = "divu"
            elif bit_str[17:20] == "110":
                command = "rem"
            elif bit_str[17:20] == "111":
                command = "remu"
        else:
            if bit_str[17:20] == "000":
                if bit_str[1] == "0":
                    command = "add"
                else:
                    command = "sub"
            elif bit_str[17:20] == "001":
                command = "sll"
            elif bit_str[17:20] == "010":
                command = "slt"
            elif bit_str[17:20] == "011":
                command = "sltu"
            elif bit_str[17:20] == "100":
                command = "xor"
            elif bit_str[17:20] == "101":
                if bit_str[1] == "0":
                    command = "srl"
                else:
                    command = "sra"
            elif bit_str[17:20] == "110":
                command = "or"
            elif bit_str[17:20] == "111":
                command = "and"
            return '%7s\t%s, %s, %s\n' % (command, reg[int(bit_str[20:25], 2)],
reg[int(bit_str[12:17], 2)], r)
    elif opcode == "0010011":
        im = int(bit_str[:12], 2) - int(bit_str[0]) * 2 ** 12
        command = ""
        if bit_str[17:20] == "000":
            command = "addi"
        elif bit_str[17:20] == "001":
            im = int(bit_str[7:12], 2)
            command = "slli"
        elif bit_str[17:20] == "010":
            command = "slti"
        elif bit_str[17:20] == "011":
            command = "sltiu"
        elif bit_str[17:20] == "100":
            command = "xori"
        elif bit_str[17:20] == "101":
```

```
            im = int(bit_str[7:12], 2)
            if bit_str[1] == "0":
                command = "srli"
            else:
                command = "srai"
        elif bit_str[17:20] == "110":
            command = "ori"
        elif bit_str[17:20] == "111":
            command = "andi"
              return '%7s\t%s, %s, %s\n' % (command, reg[int(bit_str[20:25], 2)],
reg[int(bit_str[12:17], 2)], im)
    elif opcode == "0000011":
        command = ""
        if bit_str[17:20] == "000":
            command = "lb"
        elif bit_str[17:20] == "001":
            command = "lh"
        elif bit_str[17:20] == "010":
            command = "lw"
        elif bit_str[17:20] == "100":
            command = "lbu"
        elif bit_str[17:20] == "101":
            command = "lhu"
              return '%7s\t%s, %d(%s)\n' % (command, reg[int(bit_str[20:25], 2)],
int(bit_str[:12], 2) - 2 ** 12 * int(bit_str[0]),
                                      reg[int(bit_str[12:17], 2)])
    elif opcode == "0100011":
        command = ""
        im = int(bit_str[:7] + bit_str[20:25], 2) - 2 ** 12 * int(bit_str[0])
        if bit_str[17:20] == "000":
            command = "sb"
        elif bit_str[17:20] == "001":
            command = "sh"
        elif bit_str[17:20] == "010":
            command = "sw"
            return '%7s\t%s, %d(%s)\n' % (command, reg[int(bit_str[7:12], 2)], im,
reg[int(bit_str[12:17], 2)])
    elif opcode == "1110011":
        if bit_str[11] == "0":
            return '%7s\n' % "ecall"
        else:
            return '%7s\n' % "ebreak"
    elif bit_str == "00000001000000000000000000001111":
        return '%7s\n' % "pause"
    elif bit_str == "10000011001100000000000000001111":
        return '%7s\n' % "fence.tso"
    elif opcode == "0001111":
        p_fence = ("i" if bit_str[4] == "1" else "") + ("o" if bit_str[5] == "1" else
"") + (
            "r" if bit_str[6] == "1" else "") + ("w" if bit_str[7] == "1" else "")
        s_fence = ("i" if bit_str[8] == "1" else "") + ("o" if bit_str[9] == "1" else
"") + (
            "r" if bit_str[10] == "1" else "") + ("w" if bit_str[11] == "1" else "")
        return '%7s\t%s, %s\n' % ("fence", p_fence, s_fence)
    return "invalid_instruction"
```

Данная функция раскодирует команды в соответствии со следующими таблицами, взятыми из интернета.

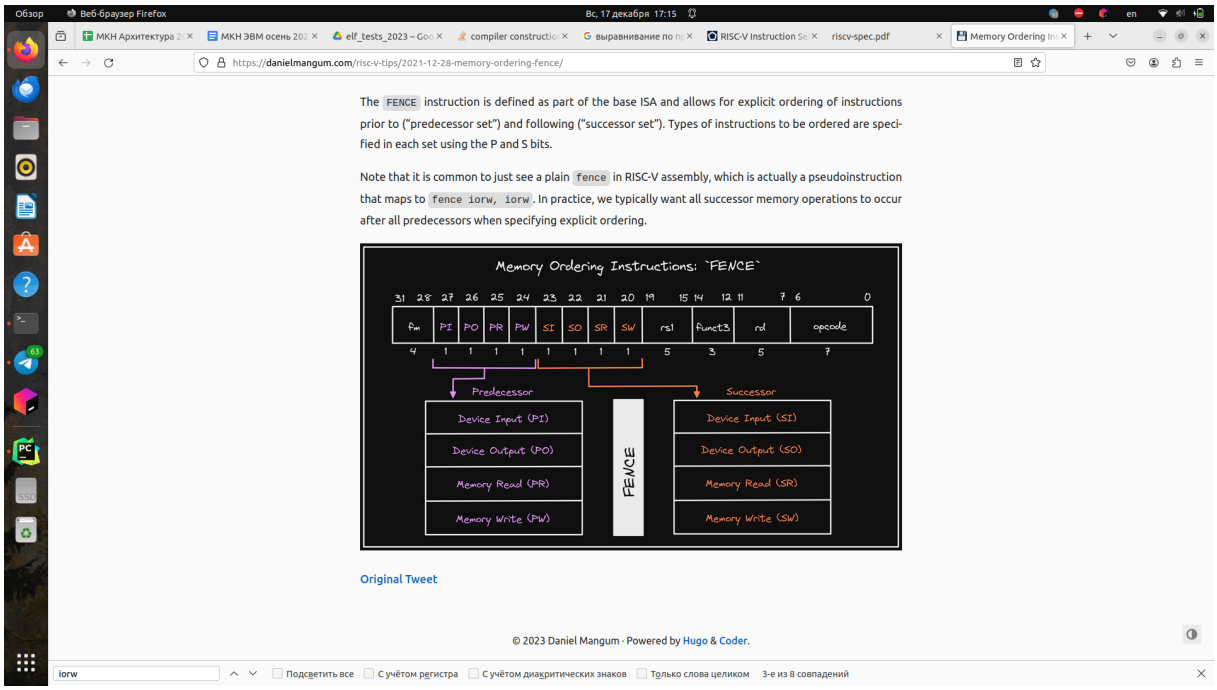
RV32I Base Instruction Set						
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	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

The RISC-V Instruction Set Manual Volume I | © RISC-V

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



Подробнее о том, как заполняется словарь names_func. При вызове функции `symtab()` мы проходимся по всей таблице symtab. И в конце каждой строки, проверяем тип обрабатываемой переменной. Если тип переменной - функция (то есть type_ == FUNC), мы добавляем данную переменную в словарь(вместе с её адресом).

Впоследствии, когда по адресу нужно получить имя функции, мы проверяем, находится ли её адрес в ключах нашего словаря. Если находится, то просто передаём имя. Если не находится, то добавляем новый регистр, под названием $< Li >$, где i - количество уже имеющихся регистров такого типа.

Словарики в самом начале файла взяты с сайта
https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-94076.html#chapter6-tbl-16

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

STT_NOTYPE

A symbol's visibility is determined from its `st_other` field. This visibility can be specified in a relocatable object. This visibility defines how that symbol can be accessed once the symbol has become part of an executable or shared object.

Table 12-20 ELF Symbol Visibility

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

STV_DEFAULT

The visibility of symbols with the `STV_DEFAULT` attribute is as specified by the symbol's binding type. Global symbols and weak symbols are visible outside of their defining component, the executable file or shared object. Local symbols are hidden. Global symt symbols can also be preempted. These symbols can by interposed by definitions of the same name in another component.

Словарик регистров взят с гитхаба
https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md