Лабораторная работа №4	M3137	2023
ISA	Сафин Булат Рамилевич	

Репозиторий

https://github.com/skkv-itmo2/itmo-comp-arch-2023-riscv-targang

Инструментарий

- Java (Temurin-17)

Результат работы

Вывод в stdout:

```
Disassembled successfully
```

Были реализованы наборы команд: RV32I, RV32M, RV32A и расширения Zifencei, Zihintpause

Разбор elf файла

За разбор elf файла отвечает класс ElfParser. Он считывает заголовок elf файла и считывает секции.
Источник: https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format

header

Первые 52 байта файла содержат заголовок elf файла. Он содержит информацию о формате файла, архитектуре, типе, версии и другие данные. Моя программа считывает следующие поля:

- `e_ident` - массив из 16 байт, содержащий общую характеристику файла.
- `e_type` - тип файла (executable, relocatable, shared object, core file). Должен быть 2 (executable).
- `e_machine` - архитектура процессора. Должен быть 243 (RISC-V).
- `e_version` - версия формата файла.
- `e_entry` - адрес точки входа.
- `e_phoff` - смещение в файле до таблицы заголовков программ.
- `e_shoff` - смещение в файле до таблицы заголовков секций.
- `e_flags` - флаги процессора.
- `e_ehsize` - размер заголовка elf файла. Для 32-битных файлов равен 52 байтам.
- `e_phentsize` - размер одного заголовка программы.
- `e_phnum` - количество заголовков программ.
- `e_shentsize` - размер одного заголовка секции.
- `e_shnum` - количество заголовков секций.
- `e_shstrndx` - индекс секции, содержащей имена секций.

После считывания заголовка, программа считывает таблицу заголовков секций.

Section headers

Таблица заголовков секций содержит информацию о каждой секции. Каждый заголовок содержит следующие поля:

- `sh_name` - смещение в таблице имен секций до имени секции.
- `sh_type` - тип секции. Если равен 2, то секция является таблицей символов. Такая секция может быть только одна.
- `sh_flags` - флаги секции.
- `sh_addr` - адрес секции в памяти.
- `sh_offset` - смещение секции в файле.
- `sh_size` - размер секции.
- `sh_link` - индекс секции, на которую ссылается эта секция.
- `sh_info` - дополнительная информация.
- `sh_addralign` - выравнивание адреса.

- `sh_entsize` - размер записи в секции.

Для каждого заголовка эта информация хранится в отдельном экземпляре класса `SectionHeader`. Программы считывает все заголовки секций и сохраняет их в массив. Если секция является таблицей символов, то программа сохраняет ее заголовок в переменную symtab.

`.shstrtab`

Таблица имен секций. Содержит имена всех секций. Её заголовок хранится в массиве заголовков секций по индексу `e_shstrndx`. Записываем в массив байтов содержимое этой секции. Далее для каждой секции считываем имя из этой таблицы по `sh_name`. Названия разделяются нулевым байтом. После этого заголовки секций помещаются в хэш-таблицу, где ключом является имя секции.

`.symtab`

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

Каждый символ в таблице символов занимает 16 байт и содержит следующие поля:

- `st_name` - смещение в таблице имен `.strtab` до имени символа. Таблицу `.strtab` можно найти по `sh_link` заголовка секции `.symtab`. Имя символа можно найти по этому смещению.
- `st_value` - значение символа. Это адрес, если символ является функцией или переменной.
- `st_size` - размер символа.
- `st_info` - информация о типе и привязке символа. Первые 4 бита - тип символа, остальные 4 - привязка.

  Возможны следующие типы символов:

  - 0 - `NOTYPE`
  - 1 - `OBJECT`
  - 2 - `FUNC`
  - 3 - `SECTION`
  - 4 - `FILE`
  - 5 - `COMMON`
  - 6 - `TLS`
  - 10 - `LOOS`
  - 12 - `HIOS`
  - 13 - `LOPROC`
  - 15 - `HIPROC`
  - `UNKNOWN`

  Если тип символа - `FUNC`, то сохраняем адрес и символ в специально заведенную хэш-таблицу меток.

  Привязка символа может быть следующей:

  - 0 - `LOCAL`
  - 1 - `GLOBAL`
  - 2 - `WEAK`
  - 10 - `LOOS`
  - 12 - `HIOS`
  - 13 - `LOPROC`
  - 15 - `HIPROC`

- `st_other` - дополнительная информация о символе. Определяет только visibility последними двумя битами.
  - 0 - `DEFAULT`
  - 1 - `INTERNAL`
  - 2 - `HIDDEN`
  - 3 - `PROTECTED`
- `st_shndx` - индекс секции, к которой относится символ.

Программа считывает таблицу символов и сохраняет ее в массив. Для каждого символа программа считывает имя из таблицы имен по `st_name` и сохраняет его вместе с остальной информацией.

Уже на этом этапе программа может вывести информацию о всех символах в таблице согласно ТЗ.

`.text`

Секция `.text` содержит исполняемый код. Первый байт секции находится по `sh_offset` заголовка. Адрес начала секции в памяти - `sh_addr`.

Программа считывает ее и сохраняет в массив байтов. Каждая инструкция занимает 32 бита. Поскольку используется кодирование little endian, значение каждой инструкции преобразуется следующим образом:

```
// bytes - массив байтов секции .text, i - индекс начала инструкции
int instruction = (bytes[i] & 0xFF)
  | (bytes[i + 1] & 0xFF) << 8
  | (bytes[i + 2] & 0xFF) << 16
  | (bytes[i + 3] & 0xFF) << 24;
```

Помещаем инструкции в хэш-таблицу, где ключом является адрес инструкции и переходим к разбору команд.

Набор команд RISC-V

32 целочисленных регистра			
x0	zero	zero	Всегда ноль
x1	ra	ra	Адрес возврата (return address)
x2	sp	sp	Указатель стека (stack pointer)
x3	gp	gp	Глобальный указатель (global pointer)
x4	tp	tp	Потоковый указатель (thread pointer)
x5	t0	t0	Temporary / альтернативный адрес возврата
x6	s3	t1	Temporary
x7	s4	t2	Temporary
x8	s0/fp	s0/fp	Saved register / frame pointer
x9	s1	s1	Saved register
x10	a0	a0	Аргумент (argument) / возвращаемое значение
x11	a1	a1	Аргумент (argument) / возвращаемое значение
x12	a2	a2	Аргумент (argument)
x13	a3	a3	Аргумент (argument)
x14	s2	a4	Аргумент (argument)
x15	t1	a5	Аргумент (argument)
x16	s5	a6	Аргумент (argument)
x17	s6	a7	Аргумент (argument)
x18-27	s7-16	s2-11	Saved register
x28-31	s17-31	t3-6	Temporary
32 дополнительных регистра с плавающей точкой			
f0-7		ft0-7	Floating-point temporaries
f8-9		fs0-1	Floating-point saved registers
f10-11		fa0-1	Floating-point arguments/return values
f12-17		fa2-7	Floating-point arguments
f18-27		fs2-11	Floating-point saved registers
f28-31		ft8-11	Floating-point temporaries

Рис. 1. регистры

Тип	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Регистр/регистр	funct7							rs2					rs1					funct3			rd					код операции					1	1
С операндом	±	imm[10:0]											rs1					funct3			rd					код операции					1	1
С длинным операндом	±	imm[30:12]																			rd					код операции					1	1
Сохранение	±	imm[10:5]						rs2					rs1					funct3			imm[4:0]					код операции					1	1
Ветвление	±	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	код операции					1	1
Переход	±	imm[10:1]										[11]	imm[19:12]								rd					код операции					1	1

Рис. 2. Типы хранения инструкций

Как видно на рис. 2, каждая инструкция в RISC-V имеет длину 32 бита. Она может состоять из следующих полей:

- `opcode` - код операции. Определяет тип инструкции.
- `rd` - номер регистра, в который записывается результат.
- `funct3` - дополнительное поле, используемое для определения типа инструкции.
- `rs1` - номер первого регистра.
- `rs2` - номер второго регистра.
- `funct7` - дополнительное поле, используемое для определения типа инструкции.
- `imm` - непосредственное значение.
- `shamt` - сдвиг.

В программе каждая инструкция хранится в своем экземпляре класса `InstructionField`. В нем содержится значение инструкции, ее адрес, ее строковое представление. Так же реализованы методы `getOpCode()`, `getRd()`, `getFunct3()`, `getRs1()`, `getRs2()`, `getFunct7()`, `getShamt()` которые возвращают соответствующие поля инструкции и `getSegment(int low, int high)`, который возвращает биты `[low, high)`.

Хэш-таблица инструкций, полученная из `.text` секции передается в класс `InstructionManager` вместе с хэш-таблицей известных из `.symtab` меток. В этом классе происходит разбор инструкций. и хранение объектов `InstructionField` в массиве, а так же хранится хэш-таблица `REGISTER_MAP`, хранящия расшифровки названий регистров.

Анализ очередной инструкции

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

RV32I Base Instruction Set

imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20|10:1|11|19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
1000	0011	0011	00000	000	00000	0001111	FENCE.TSO
0000	0001	0000	00000	000	00000	0001111	PAUSE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

Рис. 3. Набор RV32I

Для начала создаем объект `InstructionField` и передаем ему значение инструкции и адрес.
Далее проверяем на соответствие инструкции каждому типу (сверяемся с таблицей выше). Взята из https://five-embeddev.com/riscv-isa-manual/latest/instr-table.html

Integer Register-Immediate Instructions

Этому типу принадлежат инструкции `ADDI`, `SLTI`, `SLTIU`, `XORI`, `ORI`, `ANDI` (с тремя аргументами: 2 регистра и `imm`;

`SLLI`, `SRLI`, `SRAI` (с тремя аргументами: 2 регистра и `shamt`).

`LUI`, `AUIPC` (с двумя аргументами: 1 регистр и `imm`).

Integer Register-Register Instructions

Этому типу принадлежат инструкции `ADD`, `SUB`, `SLL`, `SLT`, `SLTU`, `XOR`, `SRL`, `SRA`, `OR`, `AND` (с тремя регистрами в качестве аргументов). Так же сюда можно отнести инструкции RV32M `MUL`, `MULH`, `MULHSU`, `MULHU`, `DIV`, `DIVU`, `REM`, `REMU`.

Unconditional Jumps

- `JAL` (с одним регистром и `imm`). Переходит на адрес со смещением, определенном в `imm` относительно текущего адреса. В случае, если нового адреса нет в таблице меток, то создается новая метка.
- `JALR` (с двумя регистрами и `imm`). Переходит на адрес, хранящийся в `rs1` со смещением, определенном в `imm`.

Conditional Branches

- `BEQ`, `BNE`, `BLT`, `BGE`, `BLTU`, `BGEU` (с тремя регистрами и адресом со смещением, определенном в `imm` относительно текущего адреса). В случае, если нового адреса нет в таблице меток, то создается новая метка.

Load/Store Instructions

Load: `LB`, `LH`, `LW`, `LBU`, `LHU` (с двумя регистрами и `imm`).

Store: `SB`, `SH`, `SW` (с двумя регистрами и `imm`).

Fence Instructions

- `FENCE` – `pred` и `succ` занимают 4 бита каждый, порядок `iorw`.
- `FENCE.TSO` – без аргументов).
- `FENCE.I` – без аргументов, расширение Zifencei.
- `PAUSE` – без аргументов, расширение Zihintpause.

Environment Call and Breakpoints

- `ECALL` – без аргументов.
- `EBREAK` – без аргументов.

RV32A

RV32A Standard Extension								
00010	aq	rl	00000	rs1	010	rd	0101111	LR.W
00011	aq	rl	rs2	rs1	010	rd	0101111	SC.W
00001	aq	rl	rs2	rs1	010	rd	0101111	AMOSWAP.W
00000	aq	rl	rs2	rs1	010	rd	0101111	AMOADD.W
00100	aq	rl	rs2	rs1	010	rd	0101111	AMOXOR.W
01100	aq	rl	rs2	rs1	010	rd	0101111	AMOAND.W
01000	aq	rl	rs2	rs1	010	rd	0101111	AMOOR.W
10000	aq	rl	rs2	rs1	010	rd	0101111	AMOMIN.W
10100	aq	rl	rs2	rs1	010	rd	0101111	AMOMAX.W
11000	aq	rl	rs2	rs1	010	rd	0101111	AMOMINU.W
11100	aq	rl	rs2	rs1	010	rd	0101111	AMOMAXU.W

Рис. 4. Набор RV32A

- `LR.W` с двумя регистрами, выводятся в формате `lr.w rd, rs1`.
- `SC.W` с тремя регистрами, выводятся в формате `sc.w rd, rs1, rs2`.
- Остальные инструкции выводятся в формате `amoswap.w rd, rs2, (rs1)`. https://msyksphinz-self.github.io/riscv-isadoc/html/rva.html

Если инструкция не соответствует ни одному типу, то программа присваивает ее строковому представлению значение `invalid_instruction`.

Строковое представление инструкции записывается в класс InstructionField при обработке.

После обработки всех инструкций программа проходится по массиву инструкций и записывает их в выходной файл согласно ТЗ, затемм записывыет `.symtab`.

Используемые источники

1. https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format
2. https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html
3. https://five-embeddev.com/riscv-isa-manual/latest/instr-table.html
4. https://msyksphinz-self.github.io/riscv-isadoc/html/rva.html
5. https://ru.wikipedia.org/wiki/RISC-V
6. https://elfy.io/