САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA. Ассемблер, дизассемблер»

Выполнил: Трофимов Максим Владимирович

Номер ИСУ: 334948

студ. гр. M3135

Санкт-Петербург

2021

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

Инструментарий и требования к работе: Python 3.10.

Теоретическая часть

Система кодирования RISC-V

RISC-V – открытая ISA (Instruction Set Architecture) основанная на концепции RISC. Данная архитектура имеет сокращенный набор команд, в нее входит относительно небольшой набор простых инструкций (порядка 50-ти (но существуют расширения “M”, “F”, “C”, “A” и т.д.)), при том они все имеют одинаковый размер. RISC – load/store архитектура, т.е. для выполнения операции над данными, их необходимо предварительно разместить в регистровом файле (все операции происходят только с данными в регистровом файле, а для доступа к основной памяти существуют операции load и store). RISC-V работает на 32 регистрах (для кодирования регистра используется 5 бит). Регистры называют xi – где i это число [0; 31]. При этом есть зарезервированный регистр x0, в котором хранится ноль. Существует соглашение о названии определенных индексов, тем самым обозначая их роли.

Имя	Регистр	Описание
a0 – a7	x10 – x17	Аргументы для функции
a0, a1	x10, x11	Возвращаемые значения
ra	x1	Адрес возврата
t0 – t6	x5 – x7, x28 – x31	Временные регистры
s0 – s11	x8 – x9, x18 – x27	Сохраняемые (оберегаемые) регистры
sp	x2	Указатель на вершину стека
gp	x3	Указатель на глобальные переменные
tp	x4	Указатель потока
zero	x0	Аппаратный ноль

Рисунок №1 – таблица имен регистров.

Базовая rv32i имеет длину инструкции – 32 бита. Все команды делятся на 6 типов – R, S, I, B, J, U. Младшие 7 бит кода являются так называемым opcode, который отвечает за распознавание определенного подмножества функций, а также существует специальные значения func3, func7, определяющие конкретную функцию в данном подмножестве.

Instr	Функция	Формат	Opcode	Func3	Func7	Пример использования
add	Сложение	R	0110011	0x0	0x00	op rd, rs1, rs2 xor x2, x5, x6 sll x7, x11, x12
sub	Вычитание			0x0	0x20	
xor	Исключающее ИЛИ			0x4	0x00	
or	Логическое ИЛИ			0x6	0x00	
and	Логическое И			0x7	0x00	
sll	Логический сдвиг влево			0x1	0x00	
srl	Логический сдвиг вправо			0x5	0x00	
sra	Арифметический сдвиг вправо			0x5	0x20	
slt	Результат сравнения A < B			0x2	0x00	
sltu	Беззнаковое сравнение A < B			0x3	0x00	
addi	Сложение с константой	I	0010011	0x0	–	op rd, rs1, imm addi x6, x3, -12 ori x3, x1, 0x8F
xori	Исключающее ИЛИ с константой			0x4		
ori	Логическое ИЛИ с константой			0x6		
andi	Логическое И с константой			0x7		
slli	Логический сдвиг влево			0x1	0x00	
srli	Логический сдвиг вправо			0x5	0x00	
srai	Арифметический сдвиг вправо			0x5	0x20	
slti	Результат сравнения A < B			0x2	–	
sltiu	Беззнаковое сравнение A < B			0x3		
lb	Загрузить байт из памяти	I	0000011	0x0	–	op rd, imm(rs1) lh x1, 8(x5)
lh	Загрузить полуслово из памяти			0x1		
lw	Загрузить слово из памяти			0x2		
lbu	Загрузить беззнаковый байт из памяти			0x4		
lbh	Загрузить беззнаковое полуслово из памяти			0x5		
sb	Сохранить байт в память	S	0100011	0x0	–	op rs2, imm(rs1) sw x1, 0xCF(x12)
sh	Сохранить полуслово в память			0x1		
sw	Сохранить слово в память			0x2		
beq	Перейти, если A == B	B	1100011	0x0	–	comp rs1, rs2, imm beq x8, x9, offset bltu x20, x21, 0xFC
bne	Перейти, если A != B			0x1		
blt	Перейти, если A < B			0x4		
bge	Перейти, если A >= B			0x5		
bltu	Перейти, если A < B беззнаковое			0x6		
bgeu	Перейти, если A >= B беззнаковое			0x7		
jal	Переход с сохранением адреса возврата	J	1101111	–	–	jal x1, offset
jalr	Переход по регистру с сохранением адреса возврата	I	1100111	0x0		jalr x1, 0(x5)
lui	Загрузить константу в сдвинутую на 12	U	0110111	–	–	lui x3, 0xFFFFF
auipc	Сохранить счетчик команд в сумме с константой << 12		0010111			auipc x2, 0x000FF
ecall	Передача управления операционной системе	I	1110011	–	–	–
ebreak	Передача управления отладчику					

Рисунок №2 – таблица команд rv32i

Также каждый тип команд различается «маской», по которой идет кодирование/раскодирование команды и ее аргументов.

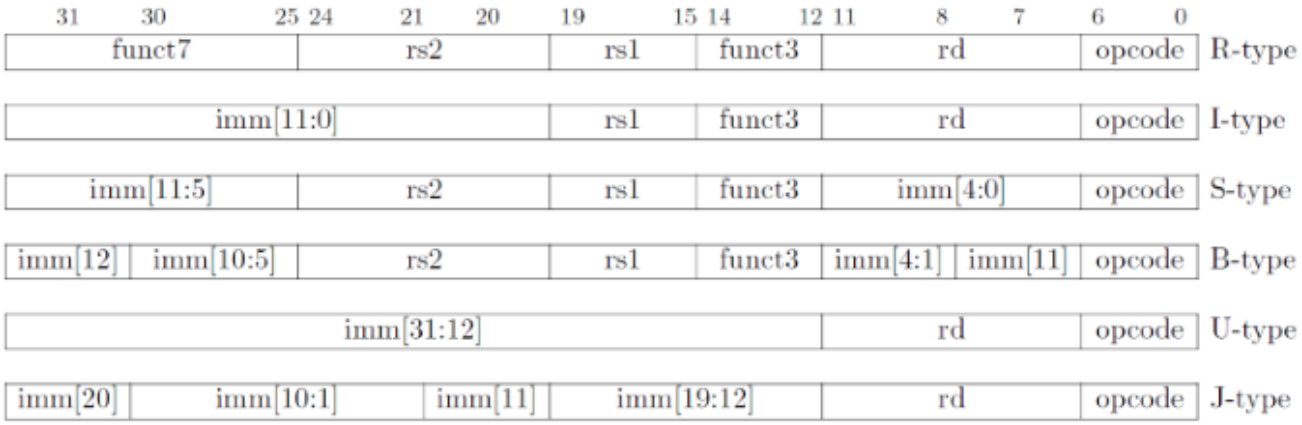

Рисунок №3 – маска различных типов

Тип R – команды работающие только на регистрах. Содержит 3 аргумента – rd, rs1, rs2. (rd – для записи, rs1, rs2 – для чтения).

Тип S – команды, записывающие значения в память. Работает почти, как R, но вместо func7 и rd используется imm (является дополнительным сдвигом для адреса в памяти).

Тип I – команды, использующие временное значение imm (immediate). В маске располагается вместо func7 и rs2.

Тип B – команды, реализующие условные переходы.

Тип J – команды, реализующие прыжок в другое место программы.

Тип U – команды, записывающие в какой-либо регистр верхние 20 бит.

Команды rv32m раскодируются по такому же принципу, единственное их отличие от rv32i – это то что это дополнение к rv32i, содержащее операции умножения, деления и т.п.

Elf -файлы

ELF – формат двоичных файлов. Структура ELF файла в плане дизайна очень гибка и расширяема. К примеру, она позволяет указывать размер адресов и т.д.

ELF файл состоит из заголовка (ELF header). Заголовок содержит в себе множество параметров. Например, указание количества бит на адрес (32 или 64),

целевая ISA, e_entry – адрес, где начинает выполняться программа, e_phoff, e_shoff - адрес, по которому находятся заголовок программы и заголовок (таблица) секций соответственно и многие другие.

Таблица секций содержит всю информацию о секциях в данном файле. В ней хранятся имена секций (точнее адрес имени в специальном блоке, хранящем все имена), их типы, адрес в файле, а также размер. Одними из таких секций являются “.text” и “.symtab” – хранящие закодированные команды и метки соответственно.

Описание работы кода

Программа написана на Python 3.10, для корректной работы необходимо установить последнюю версию Python. Для запуска программы через консоль необходимо ввести: “python Elf_Disassembler.py <имя входного файла> <имя выходного файла>”.

1. Открываем файл по названию (с побайтовым чтением), данному первым аргументом при запуске программы.
2. Проверяем, что файл корректен, т.е. он существует, его первые 4 байта являются “magic elf bytes”, а также он является 32-битным little endian.
3. Начинаем парсинг ELF header (первые 52 байта). Находим в хедере нужные нам значения (e_phoff, e_shoff, e_phentsize, e_shnum, e_shstrndx). Находим в файле “section header table” по адресу e_shoff. Находим адрес начала и размер секции “section names” используя “section neader table” и e_shstrndx . Достаем все имена из данной секции и составляем словарь (в секции “section names” слова разделены нулевым байтом).
4. Теперь достаем из “section header table” адреса и размеры “.text”, “.symtab” и “.strtab” (пригодится позже).
5. Далее запускаем функцию generate_symtab(symtab, strtab). Пройдемся по блоку “.symtab” с шагом 16 байт (т.к. одна строка занимаем 16 байт). На каждой итерации соберем нужные значения (st_name – имя метки, st_value – адрес строки программы, у которой стоит данная метка, st_size – длина кода функции, st_info, st_other, st_shndx). При этом значение st_info и st_other содержат значения параметров st_bind, st_visibility, а также st_type, где каждое из них может быть как обычным числом, так и зарезервированным значением (для st_type: {00 – NOTYPE, 01 – OBJECT, 02 – FUNC, 03 – SECTION и т.д.}, для st_bind: {00 – LOCAL, 01 – GLOBAL И Т.Д.}, для st_visibility: {00 – DEFAULT, 01 – INTERNAL и т.д.}). Значение st_shndx также может содержать специальное значение: {0000 – UNDEF,

FF00 – BEFORE, FF01 – AFTER, FFF1 – ABS, FFF2 – COMMON, FFFF - XINDEX}, при этом если значение не является ни одним из данных и входит в отрезок [0xff00; 0xff1f], [0xff20; 0xff3f], [0xff00; 0x ffff], то ему присуждается значение “SPEC_PROC”, “SPEC_OS”, “RESERVED_INDEX” соответственно.

6. Далее запускаем функцию decode_command(text, offset, symtab). Проходимся по всему блоку .text, читая по 4, либо по 2 байта (это становится понятно по первым двум битам, если они не равны 11, то это сокращенная команда RVC и соответственно необходимо читать 2 байта), раскодируем команду и добавим в текущий список commands. Если у нас 4-байтовая команда, то берем opcode (младшие 7 бит), и по нему понимаем, в каком подмножестве команд лежит данная, далее расшифровываем значения аргументов и т.д. пользуясь маской текущего типа (см. Теоретическую часть), если же не под одну команду не подошли условия маски, то выводим “unknown_command”. Если же у нас сокращенная команда RVC, то запускаем функцию build_rvc_command(command, address, symtab, queue, commands). В ней производятся все те же самые действия, только для RVC команд (берем биты на специальных местах и по ним понимаем текущую команды). При этом параллельно смотрим входит ли данный адрес команды в таблицу .symtab, если же входит, и его TYPE = FUNC, то печатаем имя метки (если имя – пустая строка, то ее имя выводится по формату “LOC_%05x” % address). Также параллельно ведем множество queue, в которое будем добавлять адрес, если на него хочет совершить прыжок другая команда (если же команда хочет прыгнуть назад, то идем в уже непустой список commands по данному адресу и вставляем туда метку, если ее не было до этого)
7. Открываем или создаем файл с именем, указанным в аргументах программы при запуске. Далее печатаем все команды из списка commands, а также все строки .symtab.

Пример работы кода

```
.text
00010074 register_fini ADDI a5, zero, 0
00010078            BEQ a5, zero, 16 0x00010088 LOC_10088
0001007c            LUI a0, 65536
00010080            ADDI a0, a0, 1164
00010084            JAL zero, 1012 0x00010478 atexit
00010088 LOC_10088: JALR zero, 0(ra)
0001008c     _start AUIPC gp, 8192
00010090            ADDI gp, gp, -684
00010094            ADDI a0, gp, -972
00010098            ADDI a2, gp, -944
0001009c            SUB a2, a2, a0
000100a0            ADDI a1, zero, 0
000100a4            JAL ra, 472 0x0001027c memset
000100a8            AUIPC a0, 0
000100ac            ADDI a0, a0, 976
000100b0            BEQ a0, zero, 16 0x000100c0 LOC_100c0
000100b4            AUIPC a0, 0
000100b8            ADDI a0, a0, 984
000100bc            JAL ra, 956 0x00010478 atexit
000100c0 LOC_100c0: JAL ra, 288 0x000101e0 __libc_init_array
000100c4            LW a0, 0(sp)
000100c8            ADDI a1, sp, 4
000100cc            ADDI a2, zero, 0
000100d0            JAL ra, 116 0x00010144 main
000100d4            JAL zero, 220 0x000101b0 exit
000100d8 __do_global_dtors_aux LBU a4, -972(gp)
000100dc            BNE a4, zero, 68 0x00010120 LOC_10120
000100e0            ADDI sp, sp, -16
000100e4            SW s0, 8(sp)
000100e8            ADDI s0, a5, 0
000100ec            SW ra, 12(sp)
000100f0            ADDI a5, zero, 0
000100f4            BEQ a5, zero, 20 0x00010108 LOC_10108
000100f8            LUI a0, 69632
000100fc            ADDI a0, a0, 1484
00010100            AUIPC ra, 0
```

```
00010104            JALR ra, 0(zero)
00010108 LOC_10108: ADDI a5, zero, 1
0001010c            LW ra, 12(sp)
00010110            SB a5, -972(gp)
00010114            LW s0, 8(sp)
00010118            ADDI sp, sp, 16
0001011c            JALR zero, 0(ra)
00010120 LOC_10120: JALR zero, 0(ra)
00010124 frame_dummy ADDI a5, zero, 0
00010128            BEQ a5, zero, 24 0x00010140 LOC_10140
0001012c            LUI a0, 69632
00010130            ADDI a1, gp, -968
00010134            ADDI a0, a0, 1484
00010138            AUIPC t1, 0
0001013c            JALR zero, 0(zero)
00010140 LOC_10140: JALR zero, 0(ra)
00010144       main ADDI sp, sp, -32
00010148            SW s0, 28(sp)
0001014c            ADDI s0, sp, 32
00010150            ADDI a5, zero, 2
00010154            SW a5, -28(s0)
00010158            ADDI a5, zero, 3
0001015c            SW a5, -32(s0)
00010160            SW zero, -20(s0)
00010164            SW zero, -24(s0)
00010168            JAL zero, 32 0x00010188 LOC_10188
0001016c LOC_1016c: LW a4, -20(s0)
00010170            LW a5, -24(s0)
00010174            ADD a5, a4, a5
00010178            SW a5, -20(s0)
0001017c            LW a5, -24(s0)
00010180            ADDI a5, a5, 1
00010184            SW a5, -24(s0)
00010188 LOC_10188: LW a4, -28(s0)
0001018c            LW a5, -32(s0)
00010190            MUL a5, a4, a5
00010194            LW a4, -24(s0)
00010198            BLT a4, a5, -44 0x0001016c LOC_1016c
0001019c            ADDI a5, zero, 0
```

```
000101a0            ADDI a0, a5, 0
000101a4            LW s0, 28(sp)
000101a8            ADDI sp, sp, 32
000101ac            JALR zero, 0(ra)
000101b0       exit ADDI sp, sp, -16
000101b4            ADDI a1, zero, 0
000101b8            SW s0, 8(sp)
000101bc            SW ra, 12(sp)
000101c0            ADDI s0, a0, 0
000101c4            JAL ra, 404 0x00010358 __call_exitprocs
000101c8            LW a0, -984(gp)
000101cc            LW a5, 60(a0)
000101d0            BEQ a5, zero, 8 0x000101d8 LOC_101d8
000101d4            JALR ra, 0(a5)
000101d8 LOC_101d8: ADDI a0, s0, 0
000101dc            JAL ra, 932 0x00010580 _exit
000101e0 __libc_init_array ADDI sp, sp, -16
000101e4            SW s0, 8(sp)
000101e8            SW s2, 0(sp)
000101ec            LUI s0, 69632
000101f0            LUI s2, 69632
000101f4            ADDI a5, s0, 1488
000101f8            ADDI s2, s2, 1488
000101fc            SUB s2, s2, a5
00010200            SW ra, 12(sp)
00010204            SW s1, 4(sp)
00010208            SRAI s2, s2, 2
0001020c            BEQ s2, zero, 32 0x0001022c LOC_1022c
00010210            ADDI s0, s0, 1488
00010214            ADDI s1, zero, 0
00010218 LOC_10218: LW a5, 0(s0)
0001021c            ADDI s1, s1, 1
00010220            ADDI s0, s0, 4
00010224            JALR ra, 0(a5)
00010228            BNE s2, s1, -16 0x00010218 LOC_10218
0001022c LOC_1022c: LUI s0, 69632
00010230            LUI s2, 69632
00010234            ADDI a5, s0, 1488
00010238            ADDI s2, s2, 1496
```

```
0001023c            SUB s2, s2, a5
00010240            SRAI s2, s2, 2
00010244            BEQ s2, zero, 32 0x00010264 LOC_10264
00010248            ADDI s0, s0, 1488
0001024c            ADDI s1, zero, 0
00010250 LOC_10250: LW a5, 0(s0)
00010254            ADDI s1, s1, 1
00010258            ADDI s0, s0, 4
0001025c            JALR ra, 0(a5)
00010260            BNE s2, s1, -16 0x00010250 LOC_10250
00010264 LOC_10264: LW ra, 12(sp)
00010268            LW s0, 8(sp)
0001026c            LW s1, 4(sp)
00010270            LW s2, 0(sp)
00010274            ADDI sp, sp, 16
00010278            JALR zero, 0(ra)
0001027c     memset ADDI t1, zero, 15
00010280            ADDI a4, a0, 0
00010284            BGEU t1, a2, 60 0x000102c0 LOC_102c0
00010288            ANDI a5, a4, 15
0001028c            BNE a5, zero, 160 0x0001032c LOC_1032c
00010290 LOC_10290: BNE a1, zero, 132 0x00010314 LOC_10314
00010294 LOC_10294: ANDI a3, a2, -16
00010298            ANDI a2, a2, 15
0001029c            ADD a3, a3, a4
000102a0 LOC_102a0: SW a1, 0(a4)
000102a4            SW a1, 4(a4)
000102a8            SW a1, 8(a4)
000102ac            SW a1, 12(a4)
000102b0            ADDI a4, a4, 16
000102b4            BLTU a4, a3, -20 0x000102a0 LOC_102a0
000102b8            BNE a2, zero, 8 0x000102c0 LOC_102c0
000102bc            JALR zero, 0(ra)
000102c0 LOC_102c0: SUB a3, t1, a2
000102c4            SLLI a3, a3, 2
000102c8            AUIPC t0, 0
000102cc            ADD a3, a3, t0
000102d0            JALR zero, 12(a3)
000102d4            SB a1, 14(a4)
```

```
000102d8            SB a1, 13(a4)
000102dc            SB a1, 12(a4)
000102e0            SB a1, 11(a4)
000102e4            SB a1, 10(a4)
000102e8            SB a1, 9(a4)
000102ec            SB a1, 8(a4)
000102f0            SB a1, 7(a4)
000102f4            SB a1, 6(a4)
000102f8            SB a1, 5(a4)
000102fc            SB a1, 4(a4)
00010300            SB a1, 3(a4)
00010304            SB a1, 2(a4)
00010308            SB a1, 1(a4)
0001030c            SB a1, 0(a4)
00010310            JALR zero, 0(ra)
00010314 LOC_10314: ANDI a1, a1, 255
00010318            SLLI a3, a1, 8
0001031c            OR a1, a1, a3
00010320            SLLI a3, a1, 16
00010324            OR a1, a1, a3
00010328            JAL zero, -148 0x00010294 LOC_10294
0001032c LOC_1032c: SLLI a3, a5, 2
00010330            AUIPC t0, 0
00010334            ADD a3, a3, t0
00010338            ADDI t0, ra, 0
0001033c            JALR ra, -96(a3)
00010340            ADDI ra, t0, 0
00010344            ADDI a5, a5, -16
00010348            SUB a4, a4, a5
0001034c            ADD a2, a2, a5
00010350            BGEU t1, a2, -144 0x000102c0 LOC_102c0
00010354            JAL zero, -196 0x00010290 LOC_10290
00010358 __call_exitprocs ADDI sp, sp, -48
0001035c            SW s4, 24(sp)
00010360            LW s4, -984(gp)
00010364            SW s2, 32(sp)
00010368            SW ra, 44(sp)
0001036c            LW s2, 328(s4)
00010370            SW s0, 40(sp)
```

```
00010374            SW s1, 36(sp)
00010378            SW s3, 28(sp)
0001037c            SW s5, 20(sp)
00010380            SW s6, 16(sp)
00010384            SW s7, 12(sp)
00010388            SW s8, 8(sp)
0001038c            BEQ s2, zero, 64 0x000103cc LOC_103cc
00010390            ADDI s6, a0, 0
00010394            ADDI s7, a1, 0
00010398            ADDI s5, zero, 1
0001039c            ADDI s3, zero, -1
000103a0 LOC_103a0: LW s1, 4(s2)
000103a4            ADDI s0, s1, -1
000103a8            BLT s0, zero, 36 0x000103cc LOC_103cc
000103ac            SLLI s1, s1, 2
000103b0            ADD s1, s2, s1
000103b4 LOC_103b4: BEQ s7, zero, 72 0x000103fc LOC_103fc
000103b8            LW a5, 260(s1)
000103bc            BEQ a5, s7, 64 0x000103fc LOC_103fc
000103c0 LOC_103c0: ADDI s0, s0, -1
000103c4            ADDI s1, s1, -4
000103c8            BNE s0, s3, -20 0x000103b4 LOC_103b4
000103cc LOC_103cc: LW ra, 44(sp)
000103d0            LW s0, 40(sp)
000103d4            LW s1, 36(sp)
000103d8            LW s2, 32(sp)
000103dc            LW s3, 28(sp)
000103e0            LW s4, 24(sp)
000103e4            LW s5, 20(sp)
000103e8            LW s6, 16(sp)
000103ec            LW s7, 12(sp)
000103f0            LW s8, 8(sp)
000103f4            ADDI sp, sp, 48
000103f8            JALR zero, 0(ra)
000103fc LOC_103fc: LW a5, 4(s2)
00010400            LW a3, 4(s1)
00010404            ADDI a5, a5, -1
00010408            BEQ a5, s0, 92 0x00010464 LOC_10464
0001040c            SW zero, 4(s1)
```

```
00010410 LOC_10410: BEQ a3, zero, -80 0x000103c0 LOC_103c0
00010414            LW a5, 392(s2)
00010418            SLL a4, s5, s0
0001041c            LW s8, 4(s2)
00010420            AND a5, a4, a5
00010424            BNE a5, zero, 36 0x00010448 LOC_10448
00010428            JALR ra, 0(a3)
0001042c LOC_1042c: LW a4, 4(s2)
00010430            LW a5, 328(s4)
00010434            BNE a4, s8, 8 0x0001043c LOC_1043c
00010438            BEQ a5, s2, -120 0x000103c0 LOC_103c0
0001043c LOC_1043c: BEQ a5, zero, -112 0x000103cc LOC_103cc
00010440            ADDI s2, a5, 0
00010444            JAL zero, -164 0x000103a0 LOC_103a0
00010448 LOC_10448: LW a5, 396(s2)
0001044c            LW a1, 132(s1)
00010450            AND a4, a4, a5
00010454            BNE a4, zero, 24 0x0001046c LOC_1046c
00010458            ADDI a0, s6, 0
0001045c            JALR ra, 0(a3)
00010460            JAL zero, -52 0x0001042c LOC_1042c
00010464 LOC_10464: SW s0, 4(s2)
00010468            JAL zero, -88 0x00010410 LOC_10410
0001046c LOC_1046c: ADDI a0, a1, 0
00010470            JALR ra, 0(a3)
00010474            JAL zero, -72 0x0001042c LOC_1042c
00010478     atexit ADDI a1, a0, 0
0001047c            ADDI a3, zero, 0
00010480            ADDI a2, zero, 0
00010484            ADDI a0, zero, 0
00010488            JAL zero, 96 0x000104e8 __register_exitproc
0001048c __libc_fini_array ADDI sp, sp, -16
00010490            SW s0, 8(sp)
00010494            LUI a5, 69632
00010498            LUI s0, 69632
0001049c            ADDI s0, s0, 1496
000104a0            ADDI a5, a5, 1500
000104a4            SUB a5, a5, s0
000104a8            SW s1, 4(sp)
```

```
000104ac            SW ra, 12(sp)
000104b0            SRAI s1, a5, 2
000104b4            BEQ s1, zero, 32 0x000104d4 LOC_104d4
000104b8            ADDI a5, a5, -4
000104bc            ADD s0, a5, s0
000104c0 LOC_104c0: LW a5, 0(s0)
000104c4            ADDI s1, s1, -1
000104c8            ADDI s0, s0, -4
000104cc            JALR ra, 0(a5)
000104d0            BNE s1, zero, -16 0x000104c0 LOC_104c0
000104d4 LOC_104d4: LW ra, 12(sp)
000104d8            LW s0, 8(sp)
000104dc            LW s1, 4(sp)
000104e0            ADDI sp, sp, 16
000104e4            JALR zero, 0(ra)
000104e8 __register_exitproc LW a4, -984(gp)
000104ec            LW a5, 328(a4)
000104f0            BEQ a5, zero, 88 0x00010548 LOC_10548
000104f4 LOC_104f4: LW a4, 4(a5)
000104f8            ADDI a6, zero, 31
000104fc            BLT a6, a4, 124 0x00010578 LOC_10578
00010500            SLLI a6, a4, 2
00010504            BEQ a0, zero, 44 0x00010530 LOC_10530
00010508            ADD t1, a5, a6
0001050c            SW a2, 136(t1)
00010510            LW a7, 392(a5)
00010514            ADDI a2, zero, 1
00010518            SLL a2, a2, a4
0001051c            OR a7, a7, a2
00010520            SW a7, 392(a5)
00010524            SW a3, 264(t1)
00010528            ADDI a3, zero, 2
0001052c            BEQ a0, a3, 40 0x00010554 LOC_10554
00010530 LOC_10530: ADDI a4, a4, 1
00010534            SW a4, 4(a5)
00010538            ADD a5, a5, a6
0001053c            SW a1, 8(a5)
00010540            ADDI a0, zero, 0
00010544            JALR zero, 0(ra)
```

```
00010548 LOC_10548: ADDI a5, a4, 332
0001054c            SW a5, 328(a4)
00010550            JAL zero, -92 0x000104f4 LOC_104f4
00010554 LOC_10554: LW a3, 396(a5)
00010558            ADDI a4, a4, 1
0001055c            SW a4, 4(a5)
00010560            OR a2, a3, a2
00010564            SW a2, 396(a5)
00010568            ADD a5, a5, a6
0001056c            SW a1, 8(a5)
00010570            ADDI a0, zero, 0
00010574            JALR zero, 0(ra)
00010578 LOC_10578: ADDI a0, zero, -1
0001057c            JALR zero, 0(ra)
00010580      _exit ADDI a1, zero, 0
00010584            ADDI a2, zero, 0
00010588            ADDI a3, zero, 0
0001058c            ADDI a4, zero, 0
00010590            ADDI a5, zero, 0
00010594            ADDI a7, zero, 93
00010598            ECALL
0001059c            BLT a0, zero, 8 0x000105a4 LOC_105a4
000105a0 LOC_105a0: JAL zero, 0 0x000105a0 LOC_105a0
000105a4 LOC_105a4: ADDI sp, sp, -16
000105a8            SW s0, 8(sp)
000105ac            ADDI s0, a0, 0
000105b0            SW ra, 12(sp)
000105b4            SUB s0, zero, s0
000105b8            JAL ra, 12 0x000105c4 __errno
000105bc            SW s0, 0(a0)
000105c0 LOC_105c0: JAL zero, 0 0x000105c0 LOC_105c0
000105c4    __errno LW a0, -976(gp)
000105c8            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] 0x115cc               0 SECTION  LOCAL    DEFAULT       2
```

```
[   3] 0x115d0              0 SECTION  LOCAL    DEFAULT       3
[   4] 0x115d8              0 SECTION  LOCAL    DEFAULT       4
[   5] 0x115e0              0 SECTION  LOCAL    DEFAULT       5
[   6] 0x11a08              0 SECTION  LOCAL    DEFAULT       6
[   7] 0x11a14              0 SECTION  LOCAL    DEFAULT       7
[   8] 0x0                  0 SECTION  LOCAL    DEFAULT       8
[   9] 0x0                  0 SECTION  LOCAL    DEFAULT       9
[  10] 0x0                  0 FILE     LOCAL    DEFAULT     ABS __call_atexit.c
[  11] 0x10074             24 FUNC     LOCAL    DEFAULT       1 register_fini
[  12] 0x0                  0 FILE     LOCAL    DEFAULT     ABS crtstuff.c
[  13] 0x115cc              0 OBJECT   LOCAL    DEFAULT       2
[  14] 0x100d8              0 FUNC     LOCAL    DEFAULT       1 __do_global_dtors_aux
[  15] 0x11a14              1 OBJECT   LOCAL    DEFAULT       7 completed.1
[  16] 0x115d8              0 OBJECT   LOCAL    DEFAULT       4 __do_global_dtors_aux_fini_array_entry
[  17] 0x10124              0 FUNC     LOCAL    DEFAULT       1 frame_dummy
[  18] 0x11a18             24 OBJECT   LOCAL    DEFAULT       7 object.0
[  19] 0x115d4              0 OBJECT   LOCAL    DEFAULT       3 __frame_dummy_init_array_entry
[  20] 0x0                  0 FILE     LOCAL    DEFAULT     ABS test.c
[  21] 0x0                  0 FILE     LOCAL    DEFAULT     ABS exit.c
[  22] 0x0                  0 FILE     LOCAL    DEFAULT     ABS impure.c
[  23] 0x115e0           1064 OBJECT   LOCAL    DEFAULT       5 impure_data
[  24] 0x0                  0 FILE     LOCAL    DEFAULT     ABS init.c
[  25] 0x0                  0 FILE     LOCAL    DEFAULT     ABS atexit.c
[  26] 0x0                  0 FILE     LOCAL    DEFAULT     ABS fini.c
[  27] 0x0                  0 FILE     LOCAL    DEFAULT     ABS __atexit.c
[  28] 0x0                  0 FILE     LOCAL    DEFAULT     ABS sys_exit.c
[  29] 0x0                  0 FILE     LOCAL    DEFAULT     ABS errno.c
[  30] 0x0                  0 FILE     LOCAL    DEFAULT     ABS crtstuff.c
[  31] 0x115cc              0 OBJECT   LOCAL    DEFAULT       2 __FRAME_END__
[  32] 0x0                  0 FILE     LOCAL    DEFAULT     ABS
[  33] 0x115dc              0 NOTYPE   LOCAL    DEFAULT       4 __fini_array_end
[  34] 0x115d8              0 NOTYPE   LOCAL    DEFAULT       4 __fini_array_start
[  35] 0x115d8              0 NOTYPE   LOCAL    DEFAULT       3 __init_array_end
[  36] 0x115d0              0 NOTYPE   LOCAL    DEFAULT       3 __preinit_array_end
[  37] 0x115d0              0 NOTYPE   LOCAL    DEFAULT       3 __init_array_start
[  38] 0x115d0              0 NOTYPE   LOCAL    DEFAULT       3 __preinit_array_start
[  39] 0x11de0              0 NOTYPE   GLOBAL   DEFAULT     ABS __global_pointer$
[  40] 0x105c4              8 FUNC     GLOBAL   DEFAULT       1 __errno
[  41] 0x11a08              0 NOTYPE   GLOBAL   DEFAULT       6 __SDATA_BEGIN__
```

```
[  42] 0x11a0c               0 OBJECT   GLOBAL   HIDDEN        6 __dso_handle
[  43] 0x11a08               4 OBJECT   GLOBAL   DEFAULT       6 _global_impure_ptr
[  44] 0x101e0             156 FUNC     GLOBAL   DEFAULT       1 __libc_init_array
[  45] 0x1048c              92 FUNC     GLOBAL   DEFAULT       1 __libc_fini_array
[  46] 0x10358             288 FUNC     GLOBAL   DEFAULT       1 __call_exitprocs
[  47] 0x1008c              76 FUNC     GLOBAL   DEFAULT       1 _start
[  48] 0x104e8             152 FUNC     GLOBAL   DEFAULT       1 __register_exitproc
[  49] 0x11a30               0 NOTYPE   GLOBAL   DEFAULT       7 __BSS_END__
[  50] 0x11a14               0 NOTYPE   GLOBAL   DEFAULT       7 __bss_start
[  51] 0x1027c             220 FUNC     GLOBAL   DEFAULT       1 memset
[  52] 0x10144             108 FUNC     GLOBAL   DEFAULT       1 main
[  53] 0x10478              20 FUNC     GLOBAL   DEFAULT       1 atexit
[  54] 0x11a10               4 OBJECT   GLOBAL   DEFAULT       6 _impure_ptr
[  55] 0x115e0               0 NOTYPE   GLOBAL   DEFAULT       5 __DATA_BEGIN__
[  56] 0x11a14               0 NOTYPE   GLOBAL   DEFAULT       6 _edata
[  57] 0x11a30               0 NOTYPE   GLOBAL   DEFAULT       7 _end
[  58] 0x101b0              48 FUNC     GLOBAL   DEFAULT       1 exit
[  59] 0x10580              68 FUNC     GLOBAL   DEFAULT       1 _exit
```

Листинг

Elf_Disassembler.py (Python 3.10)

```
import sys



"""
@author Trofimov Maxim (tg: @trofik00777)
"""



def bp(a):
    for i in range(0, len(a), 16):
        print(" ".join(a[i:i + 16]),
              ''.join(chr(int(j, 16)) for j in a[i:i + 16]))
```

```
def generate_dict_section_names(section):
    names = dict()

    current_name = ""
    offset = 0
    for b in section:
        if b == "00":
            names[offset] = current_name
            offset += len(current_name) + 1
            current_name = ""
        else:
            current_name += chr(int(b, 16))

    return names



def find_name_in_strtab(strtab, offset):
    name = ""
    for i in range(offset, len(strtab)):
        if strtab[i] == "00":
            return name
        name += chr(int(strtab[i], 16))
    return name



def generate_symtab(symtab, strtab):
    to_type = {
        '00': 'NOTYPE',
        '01': 'OBJECT',
        '02': 'FUNC',
        '03': 'SECTION',
```

```
    '04': 'FILE',
    '05': 'COMMON',
    '06': 'TLS',
    '07': 'NUM',
    '10': 'LOOS',
    '12': 'HIOS',
    '13': 'LOPROC',
    '15': 'HIPROC'
}
to_bind = {
    '00': 'LOCAL',
    '01': 'GLOBAL',
    '02': 'WEAK',
    '03': 'NUM',
    '10': 'LOOS',
    '12': 'HIOS',
    '13': 'LOPROC',
    '15': 'HIPROC'
}
to_visibility = {
    '00': 'DEFAULT',
    '01': 'INTERNAL',
    '02': 'HIDDEN',
    '03': 'PROTECTED'
}
to_index = {
    '0000': 'UNDEF',
    'ff00': 'BEFORE',
    'ff01': 'AFTER',
    # 'ff1f': 'HIPROC',
    # 'ff20': 'LOOS',
    # 'ff3f': 'HIOS',
```

```
    'fff1': 'ABS',
    'fff2': 'COMMON',
    'ffff': 'XINDEX'  # SHN_SPEC для диапазона
}

# print("%s %-15s %7s %-8s %-8s %-8s %6s %s" %
#       ("Symbol", "Value", "Size", "Type", "Bind", "Vis", "Index", "Name"))
gen_symtab = []

for i in range(0, len(symtab), 16):
    st_name = find_name_in_strtab(strtab, int(''.join(reversed(symtab[i:i + 4])), 16))
    st_value = hex(int(''.join(reversed(symtab[i + 4:i + 8])), 16))
    st_size = int(''.join(reversed(symtab[i + 8:i + 12])), 16)
    st_info = bin(int(symtab[i + 12], 16))[2:].rjust(8, '0')
    st_other = symtab[i + 13]
    st_shndx = ''.join(reversed(symtab[i + 14:i + 16]))

    key_to_type = hex(int(st_info[-4:], 2))[2:].rjust(2, '0')
    if key_to_type in to_type:
        st_type = to_type[key_to_type]
    else:
        st_type = "UNKNOWN_TYPE"
    st_bind = to_bind.get(hex(int(st_info[:4], 2))[2:].rjust(2, '0'))
    st_visibility = to_visibility.get(hex(int(st_other[-2:], 16))[2:].rjust(2, '0'))
    key_to_index = hex(int(st_shndx, 16))[2:].rjust(4, '0')
    if key_to_index in to_index:
        st_index = to_index[key_to_index]
    else:
        if 0xff00 <= int(key_to_index, 16) <= 0xff1f:
            st_index = "SPEC_PROC"
        elif 0xff20 <= int(key_to_index, 16) <= 0xff3f:
            st_index = "SPEC_OS"
```

```
            elif 0xff00 <= int(key_to_index, 16):
                st_index = "RESERVED_INDEX"
            else:
                st_index = int(key_to_index, 16)

        gen_symtab.append((i >> 4, st_value[2:], st_size, st_type, st_bind, st_visibility, st_index,
st_name))
        # print("[%4i] 0x%-15s %5i %-8s %-8s %-8s %6s %s" %
        #       (i >> 4, st_value[2:], st_size, st_type, st_bind, st_visibility, st_index, st_name))

    return gen_symtab


def convert_reg_names(rd):
    if rd == 0:
        rd = "zero"
    elif rd == 1:
        rd = "ra"
    elif rd == 2:
        rd = "sp"
    elif rd == 3:
        rd = "gp"
    elif rd == 4:
        rd = "tp"
    elif rd <= 7:
        rd = f"t{rd - 5}"
    elif rd <= 9:
        rd = f"s{rd - 8}"
    elif rd <= 17:
        rd = f"a{rd - 10}"
    elif rd <= 27:
        rd = f"s{rd - 16}"
```

```
    elif rd <= 31:
        rd = f"t{rd - 25}"

    return rd



def convert_reg_names_rvc(rd):
    # "000", "s0",
    # "001", "s1",
    # "010", "a0",
    # "011", "a1",
    # "100", "a2",
    # "101", "a3",
    # "110", "a4",
    # "111", "a5"
    match rd:
        case 0:
            return "s0"
        case 1:
            return "s1"
        case 2:
            return "a0"
        case 3:
            return "a1"
        case 4:
            return "a2"
        case 5:
            return "a3"
        case 6:
            return "a4"
        case 7:
            return "a5"
```

```
        case _:
            return ""


def make_new_address(address, imm, symtab, queue, commands):
    new_address = hex(address + imm)[2:]
    new_address_symtab = find_in_symtab_by_address(symtab, new_address, True)
    if imm > 0:
        queue.add(new_address)
    elif imm < 0:
        for ind in range(len(commands)):
            if commands[ind][1][0] == int(new_address, 16):
                if commands[ind][1][1] == "":
                    commands[ind][1][1] = new_address_symtab
    else:
        return new_address, new_address_symtab, False
    return new_address, new_address_symtab, True


def build_rvc_command(command, address, symtab, queue, commands):
    code1_0 = command[-2:]
    str_address = hex(address)[2:]
    match code1_0:
        case "00":
            code15_13 = command[-16:-13]
            match code15_13:
                case '000':
                    nzuimm = int(command[-11:-7] + command[-13:-11] + command[-6] + command[-7] + "00", 2)
                    rd = int(command[-5:-2], 2)
                    rd = convert_reg_names_rvc(rd)
                    name = "c.addi4spn"
```

```
                    return [ "%08x %10s %s %s, %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, "sp", nzuimm]]

                case '001':

                    uimm = int(command[-7:-5] + command[-13:-10] + "000", 2)

                    rd = int(command[-5:-2], 2)

                    rd = convert_reg_names_rvc(rd)

                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.fld"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, uimm, rs1]]

                case '010':

                    uimm = int(command[-6] + command[-13:-10] + command[-7] + "00", 2)

                    rd = int(command[-5:-2], 2)

                    rd = convert_reg_names_rvc(rd)

                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.lw"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, uimm, rs1]]

                case '011':

                    uimm = int(command[-6] + command[-13:-10] + command[-7] + "00", 2)

                    rd = int(command[-5:-2], 2)

                    rd = convert_reg_names_rvc(rd)

                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.flw"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, uimm, rs1]]

                case '101':

                    uimm = int(command[-7:-5] + command[-13:-10] + "000", 2)

                    rs2 = int(command[-5:-2], 2)

                    rs2 = convert_reg_names_rvc(rs2)
```

```
                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.fsd"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, rs2, uimm]]

                case '110':

                    uimm = int(command[-6] + command[-13:-10] + command[-7] + "00", 2)

                    rs2 = int(command[-5:-2], 2)

                    rs2 = convert_reg_names_rvc(rs2)

                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.sw"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, uimm, rs2]]

                case '111':

                    uimm = int(command[-6] + command[-13:-10] + command[-7] + "00", 2)

                    rs2 = int(command[-5:-2], 2)

                    rs2 = convert_reg_names_rvc(rs2)

                    rs1 = int(command[-10:-7], 2)

                    rs1 = convert_reg_names_rvc(rs1)

                    name = "c.fsw"

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, uimm, rs2]]

                case _:

                    commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])

                    return

                    # assert 0 == 1, "RVC 00 commander Error!"

        case "01":

            code15_13 = command[-16:-13]

            match code15_13:

                case '000':

                    if command[-12:-7] == "00000":

                        name = "c.nop"
```

```
                        return [ "%08x %10s %s" , [address, find_in_symtab_by_address(symtab, str_address,
str_address in queue), name.upper()]]

                    else:

                        rs1 = int(command[-12:-7], 2)

                        rs1 = convert_reg_names(rs1)

                        nzimm = bin_to_int_with_sign(command[-13] + command[-7:-2])

                        name = "c.addi"

                        return [ "%08x %10s %s %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, nzimm]]

                case '001':

                    name = "c.jal"

                    imm = bin_to_int_with_sign(command[-13] + command[-9] + command[-11:-9] + command[-7]
+ command[-8] + command[-3] + command[-12] + command[-6:-3] + '0')

                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)



                    if not result:

                        commands.append(["%08x %10s %s %s 0x%08x %s",

                                         [address, new_address_symtab,

                                          name.upper(), imm, int(new_address, 16),

                                          new_address_symtab.rstrip(":")]])  # bug

                        return None



                    return [ "%08x %10s %s %s 0x%08x %s",

                             [address, find_in_symtab_by_address(symtab, str_address, str_address in
queue), name.upper(), imm, int(new_address, 16), new_address_symtab.rstrip(":")]]

                case '010':

                    name = "c.li"

                    rd = int(command[-12:-7], 2)

                    rd = convert_reg_names(rd)

                    imm = bin_to_int_with_sign(command[-13] + command[-7:-2])

                    return [ "%08x %10s %s %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, imm]]

                case '011':
```

```
                    rd = int(command[-12:-7], 2)
                    if rd == 2:
                        name = "c.addi16sp"
                        nzimm = bin_to_int_with_sign(command[-13] + command[-5:-3] + command[-6] +
command[-3] + command[-7] + "0000")
                    else:
                        name = "c.lui"
                        nzimm = bin_to_int_with_sign(command[-13] + command[-7:-2] + "0" * 12)
                    return [ "%08x %10s %s %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), convert_reg_names(rd), nzimm]]
                case '100':
                    code11_10 = command[-12:-10]
                    match code11_10:
                        case '00':
                            nzuimm = int(command[-13] + command[-7:-2], 2)
                            rs1 = int(command[-10:-7], 2)
                            rs1 = convert_reg_names_rvc(rs1)
                            if nzuimm == 0:
                                name = "c.srli64"
                                return [ "%08x %10s %s %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1]]
                            else:
                                name = "c.srli"
                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, nzuimm]]
                        case '01':
                            nzuimm = int(command[-13] + command[-7:-2], 2)
                            rs1 = int(command[-10:-7], 2)
                            rs1 = convert_reg_names_rvc(rs1)
                            if nzuimm == 0:
                                name = "c.srai64"
                                return [ "%08x %10s %s %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1]]
                            else:
```

```
                                name = "c.srai"

                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, nzuimm]]

                            case '10':

                                imm = bin_to_int_with_sign(command[-13] + command[-7:-2], 2)

                                rs1 = int(command[-10:-7], 2)

                                rs1 = convert_reg_names_rvc(rs1)

                                name = "c.andi"

                                return [ "%08x %10s %s %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, imm]]

                            case '11':

                                rs1 = int(command[-10:-7], 2)

                                rs1 = convert_reg_names_rvc(rs1)

                                rs2 = int(command[-5:-2], 2)

                                rs2 = convert_reg_names_rvc(rs2)

                                code6_5 = command[-7:-5]

                                if command[-13] == '0':

                                    match code6_5:

                                        case '00':

                                            name = "c.sub"

                                        case '01':

                                            name = "c.xor"

                                        case '10':

                                            name = "c.or"

                                        case '11':

                                            name = "c.and"

                                        case _:

                                            commands.append(["%08x %10s %s", [address, "",
"unknown_command".upper()]])

                                            return

                                            # assert 0 == 1, "c.sub-c.and assertion"

                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, rs2]]
```

```
                            else:
                                match code6_5:
                                    case '00':
                                        name = "c.subw"
                                    case '01':
                                        name = "c.addw"
                                    case '10' | '11':
                                        name = "reserved"
                                        return [ "%08x %10s %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper()]]
                                    case _:
                                        commands.append(["%08x %10s %s", [address, "",
"unknown_command".upper()]])
                                        return
                                        # assert 0 == 1, "c.subw,c.addw assertion"
                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, rs2]]
                        case _:
                            commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])
                            return
                            # assert 0 == 1, "RVC 100 assertion"
                case '101':
                    name = "c.j"
                    imm = bin_to_int_with_sign(command[-13] + command[-9] + command[-11:-9] + command[-7]
+ command[-8] + command[-3] + command[-12] + command[-6:-3] + '0', 2)
                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)

                    if not result:
                        commands.append(["%08x %10s %s %s 0x%08x %s",
                                         [address, new_address_symtab,
                                          name.upper(), imm, int(new_address, 16),
                                          new_address_symtab.rstrip(":")]])  # bug
                        return None
```

```
                    return [ "%08x %10s %s %s 0x%08x %s" ,
                             [address, find_in_symtab_by_address(symtab, str_address, str_address in
queue), name.upper(), imm, int(new_address, 16), new_address_symtab.rstrip(":")]]
                case '110':
                    name = "c.beqz"
                    rs1 = int(command[-10:-7], 2)
                    rs1 = convert_reg_names_rvc(rs1)
                    imm = bin_to_int_with_sign(command[-13] + command[-7:-5] + command[-3] + command[-12:-
10] + command[-5:-3] + '0', 2)
                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)

                    if not result:
                        commands.append(["%08x %10s %s %s, %s 0x%08x %s",
                                         [address, new_address_symtab,
                                          name.upper(), rs1, imm, int(new_address, 16),
                                          new_address_symtab.rstrip(":")]])  # bug
                        return None

                    return [ "%08x %10s %s %s, %s 0x%08x %s" ,
                             [address, find_in_symtab_by_address(symtab, str_address, str_address in
queue), name.upper(), rs1, imm, int(new_address, 16), new_address_symtab.rstrip(":")]]
                case '111':
                    name = "c.bnez"
                    rs1 = int(command[-10:-7], 2)
                    rs1 = convert_reg_names_rvc(rs1)
                    imm = bin_to_int_with_sign(command[-13] + command[-7:-5] + command[-3] + command[-12:-
10] + command[-5:-3] + '0', 2)
                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)

                    if not result:
                        commands.append(["%08x %10s %s %s, %s 0x%08x %s",
```

```
                                        [address, new_address_symtab,
                                         name.upper(), rs1, imm, int(new_address, 16),
                                         new_address_symtab.rstrip(":")]])  # bug
                        return None


                    return [ "%08x %10s %s %s, %s 0x%08x %s" ,
                             [address, find_in_symtab_by_address(symtab, str_address, str_address in
queue), name.upper(), rs1, imm, int(new_address, 16), new_address_symtab.rstrip(":")]]
                case _:
                    commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])
                    return
                    # assert 0 == 1, "RVC 01 assertion Error!"
        case "10":
            code15_13 = command[-16:-13]
            rs1 = int(command[-12:-7], 2)
            rs1 = convert_reg_names(rs1)
            match code15_13:
                case '000':
                    nzuimm = int(command[-13] + command[-7:-2], 2)
                    if nzuimm == 0:
                        name = "c.slli64"
                        return [ "%08x %10s %s %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1]]
                    else:
                        name = "c.slli"
                        return [ "%08x %10s %s %s, %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, nzuimm]]
                case '001':
                    name = "c.fldsp"
                    uimm = int(command[-5:-2] + command[-13] + command[-7:-5] + "000", 2)
                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, uimm, "sp"]]
                case '010':
```

```
                    name = "c.lwsp"

                    uimm = int(command[-4:-2] + command[-13] + command[-7:-4] + "00", 2)

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, uimm, "sp"]]

                case '011':

                    name = "c.flwsp"

                    uimm = int(command[-4:-2] + command[-13] + command[-7:-4] + "00", 2)

                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1, uimm, "sp"]]

                case '100':

                    match command[-13]:

                        case '0':

                            rs2 = int(command[-7:-2], 2)

                            if rs2 == 0:

                                name = "c.jr"

                                return [ "%08x %10s %s %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs1]]

                            else:

                                name = "c.mv"

                                rs2 = convert_reg_names(rs2)

                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, rs2]]

                        case '1':

                            if rs1 == "zero":

                                name = "c.ebreak"

                                return [ "%08x %10s %s" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper()]]

                            else:

                                rs2 = int(command[-7:-2], 2)

                                if rs2 == 0:

                                    name = "c.jalr"

                                    return [ "%08x %10s %s %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1]]

                                else:
```

```
                                name = "c.add"
                                rs2 = convert_reg_names(rs2)
                                return [ "%08x %10s %s %s, %s" , [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs1, rs2]]
                        case _:
                            commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])
                            return
                            # assert 0 == 1, "RVC 100 assert"
                case '101':
                    name = "c.fsdsp"
                    uimm = int(command[-10:-7] + command[-13:-10] + "000", 2)
                    rs2 = int(command[-7:-2], 2)
                    rs2 = convert_reg_names(rs2)
                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs2, uimm, "sp"]]
                case '110':
                    name = "c.swsp"
                    uimm = int(command[-9:-7] + command[-13:-9] + "00", 2)
                    rs2 = int(command[-7:-2], 2)
                    rs2 = convert_reg_names(rs2)
                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs2, uimm, "sp"]]
                case '111':
                    name = "c.fswsp"
                    uimm = int(command[-9:-7] + command[-13:-9] + "00", 2)
                    rs2 = int(command[-7:-2], 2)
                    rs2 = convert_reg_names(rs2)
                    return [ "%08x %10s %s %s, %s(%s)" , [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rs2, uimm, "sp"]]
                case _:
                    commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])
                    return
                    # assert 0 == 1, "RVC 10 assertion Error! Maybe"
```

```
        case _:
            commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])
            return
            # assert 0 == 1, f"Incorrect RVC command {code1_0}"



def find_in_symtab_by_address(symtab, address, must=False):
    # result = []
    for i in symtab:
        if i[1] == address:
            if i[3] == "FUNC":
                # if i[2] != 0:  # but size==0?
                if i[-1] != "":
                    return i[-1]
                    # result.append(i[-1])
                else:
                    return "LOC_%05x" % int(address, 16)
                    # result.append("LOC_%05x" % int(address, 16))
    # if result == []:
    if must:
        return "LOC_%05x" % int(address, 16) + ":"
    return ""




def bin_to_int_with_sign(n, *ars):
    len_n = len(n) - 1
    result = (2**(len_n)) * (-int(n[0]))
    pos = 0
    step = 1
    while pos < len_n:
```

```
        result += step * int(n[-(pos + 1)])
        step *= 2
        pos += 1
    return result




def decode_commands(text, offset, symtab):
    rv32im = {
        '18': [
            'B',
            {
                '0': 'beq',
                '1': 'bne',
                '4': 'blt',
                '5': 'bge',
                '6': 'bltu',
                '7': 'bgeu'
            }
        ],
        '04': [
            'I',
            {
                '0': 'addi',
                '1': 'slli',
                '2': 'slti',
                '3': 'sltiu',
                '4': 'xori',
                '5': 'srli',  # or srai, i don't know :(
                '6': 'ori',
                '7': 'andi'
            }
```

```
],
'0c': [
    'R',
    {
        '0': {
            '0': 'add',
            '1': 'sll',
            '2': 'slt',
            '3': 'sltu',
            '4': 'xor',
            '5': 'srl',
            '6': 'or',
            '7': 'and'
        },
        '32': {
            '0': 'sub',
            '5': 'sra'
        },
        # rv32m
        '1': {
            '0': 'mul',
            '1': 'mulh',
            '2': 'mulhsu',
            '3': 'mulhu',
            '4': 'div',
            '5': 'divu',
            '6': 'rem',
            '7': 'remu'
        }
    }

],
```

```
'00': [
    'I',
    {
        '0': 'lb',
        '1': 'lh',
        '2': 'lw',
        '4': 'lbu',
        '5': 'lhu'
    }
],
'08': [
    'S',
    {
        '0': 'sb',
        '1': 'sh',
        '2': 'sw'
    }
],
'0d': [
    'U',
    'lui'
],
'05': [
    'U',
    'auipc'
],
'19': [
    'I',
    {
        '0': 'jalr'
    }
],
```

```
    '1b': [
        'J',
        'jal'
    ],
    '1c': [
        'I',
        {
            '0': ['ecall', 'ebreak'],
            '1': 'csrrw',
            '2': 'csrrs',
            '3': 'csrrc',
            '5': 'csrrwi',
            '6': 'csrrsi',
            '7': 'csrrci'
        }
    ]
}


commands = []


index = 0
queue = set()
while index < len(text):
    command = bin(int(''.join(reversed(text[index:index + 4])), 16))[2:].rjust(32, '0')
    if command[-2:] != "11":
        command = bin(int(''.join(reversed(text[index:index + 2])), 16))[2:].rjust(16, '0')
        index += 2
        command_rvc = build_rvc_command(command, offset + index - 2, symtab, queue, commands)
        if command_rvc is not None:
            commands.append(command_rvc)
    else:
        index += 4
```

```
code1_0 = command[-2:]

code6_2 = hex(int(command[-7:-2], 2))[2:].rjust(2, '0')

address = offset + index - 4

str_address = hex(address)[2:]


if code6_2 in rv32im:

    form, name = rv32im[code6_2]

    if isinstance(name, str):

        pass

    else:

        code14_12 = str(int(command[-15:-12], 2))

        if code6_2 == "0c":

            code31_25 = str(int(command[-32:-25], 2))

            name = name[code31_25]

            name = name[code14_12]

        elif code6_2 == "1c":

            name = name[code14_12]

            if isinstance(name, list):

                name = name[int(command[-21])]

        else:

            name = name[code14_12]

    if name == "srli":

        if command[-31] == '1':

            name = "srai"

else:

    form = name = None



match form:

    case 'R':

        rd = int(command[-12:-7], 2)

        rd = convert_reg_names(rd)
```

```
                    rs1 = int(command[-20:-15], 2)

                    rs1 = convert_reg_names(rs1)

                    rs2 = int(command[-25:-20], 2)

                    rs2 = convert_reg_names(rs2)

                    commands.append(["%08x %10s %s %s, %s, %s", [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rd, rs1, rs2]])

                case 'I':

                    rd = int(command[-12:-7], 2)

                    rd = convert_reg_names(rd)

                    rs1 = int(command[-20:-15], 2)

                    rs1 = convert_reg_names(rs1)

                    imm = bin_to_int_with_sign(command[-32:-20])

                    match name[0]:

                        case 'l' | 'j':

                            commands.append(["%08x %10s %s %s, %s(%s)", [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rd, imm, rs1]])

                        case 'e':

                            commands.append(["%08x %10s %s", [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper()]])

                        case _:

                            if name == "srai":

                                imm -= 1024

                            commands.append(["%08x %10s %s %s, %s, %s", [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rd, rs1, imm]])

                case 'S':

                    rs2 = int(command[-25:-20], 2)

                    rs2 = convert_reg_names(rs2)

                    rs1 = int(command[-20:-15], 2)

                    rs1 = convert_reg_names(rs1)

                    imm = bin_to_int_with_sign(command[-32:-25] + command[-12:-7])

                    commands.append(["%08x %10s %s %s, %s(%s)", [address,
find_in_symtab_by_address(symtab, str_address, str_address in queue), name.upper(), rs2, imm, rs1]]) # bug

                case 'B':

                    rs2 = int(command[-25:-20], 2)
```

```
                    rs2 = convert_reg_names(rs2)

                    rs1 = int(command[-20:-15], 2)

                    rs1 = convert_reg_names(rs1)

                    imm = bin_to_int_with_sign(command[-32] + command[-8] + command[-31:-25] + command[-
12:-8] + '0')

                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)



                    if not result:

                        commands.append(["%08x %10s %s %s, %s, %s 0x%08x %s",

                                         [address, new_address_symtab,

                                          name.upper(), rs1, rs2, imm, int(new_address, 16),
new_address_symtab.rstrip(":")]])  # bug

                        continue



                    commands.append(["%08x %10s %s %s, %s, %s 0x%08x %s",

                                     [address, find_in_symtab_by_address(symtab, str_address, str_address
in queue), name.upper(), rs1, rs2, imm, int(new_address, 16), new_address_symtab.rstrip(":")]])

                case 'J':

                    rd = int(command[-12:-7], 2)

                    rd = convert_reg_names(rd)

                    imm = bin_to_int_with_sign(command[-32] + command[-20:-12] + command[-21] + command[-
31:-21] + '0')

                    new_address, new_address_symtab, result = make_new_address(address, imm, symtab,
queue, commands)



                    if not result:

                        commands.append(["%08x %10s %s %s, %s 0x%08x %s",

                                         [address, new_address_symtab,

                                          name.upper(), rd, imm, int(new_address, 16),
new_address_symtab.rstrip(":")]])  # bug

                        continue



                    commands.append(["%08x %10s %s %s, %s 0x%08x %s",
```

```
                                    [address, find_in_symtab_by_address(symtab, str_address, str_address
in queue), name.upper(), rd, imm, int(new_address, 16), new_address_symtab.rstrip(":")]])

                case 'U':

                    rd = int(command[-12:-7], 2)

                    rd = convert_reg_names(rd)

                    imm = bin_to_int_with_sign(command[-32:-12] + "0" * 12) # bug

                    commands.append(["%08x %10s %s %s, %s", [address, find_in_symtab_by_address(symtab,
str_address, str_address in queue), name.upper(), rd, imm]])

                case _:

                    commands.append(["%08x %10s %s", [address, "", "unknown_command".upper()]])

                    # assert 0 == 1, "match-case assertion"


    return commands



def main(args=("test_elfRVC.elf", "out.txt"), log=False):

    filename = args[0]

    # assert filename.endswith(".elf"), "This format file not correct (It must be *.elf)"

    with open(filename, 'rb') as elf:

        test = elf.read()

        byte = [hex(i)[2:].rjust(2, '0') for i in test]


    assert byte[:7] == ['7f', '45', '4c', '46', '01', '01', '01'], "It's not correct elf file..."  #
проверяем что это elf32 with little endian


    # e_entry = ''.join(reversed(byte[24:28]))

    e_phoff = ''.join(reversed(byte[28:32]))

    e_shoff = ''.join(reversed(byte[32:36]))

    e_phentsize = ''.join(reversed(byte[42:44]))

    e_shnum = int(''.join(reversed(byte[48:50])), 16)

    e_shstrndx = int(''.join(reversed(byte[50:52])), 16)
```

```
    section_header_table = byte[int(e_shoff, 16):]
    if log:
        print(test)
        print(byte)
        print(e_phoff, e_shoff, e_phentsize, '\n')
        bp(section_header_table)
        print()
        print(len(section_header_table) // 40)

    sh_offset_shrtrtab = int(''.join(reversed(section_header_table[40 * e_shstrndx + 16:40 * e_shstrndx + 16 + 4])), 16)
    sh_size_shrtrtab = int(''.join(reversed(section_header_table[40 * e_shstrndx + 20:40 * e_shstrndx + 20 + 4])), 16)

    section_names = byte[sh_offset_shrtrtab:sh_offset_shrtrtab + sh_size_shrtrtab]
    names = generate_dict_section_names(section_names)
    if log:
        print(names)
        bp(section_header_table)

    sections = dict()

    for index in range(e_shnum):
        sh_name = int(''.join(reversed(section_header_table[40 * index: 40 * index + 4])), 16)
        if log:
            print(f"{sh_name = }")
        name = names.get(sh_name)
        if name in [".text", ".symtab", ".strtab"]:
            sections[name] = dict()
            sections[name]["offset"] = int(''.join(reversed(section_header_table[40 * index + 16: 40 * index + 16 + 4])), 16)
```

```
            sections[name]["size"] = int(''.join(reversed(section_header_table[40 * index + 20: 40 * index + 20 + 4])), 16)
            sections[name]["address"] = int(''.join(reversed(section_header_table[40 * index + 12: 40 * index + 12 + 4])), 16)

    if log:
        print(sections)

    assert len(sections) == 3, "No such .text or .symtab"

    symtab = byte[sections[".symtab"]["offset"]:sections[".symtab"]["offset"] + sections[".symtab"]["size"]]
    if log:
        bp(symtab)
        print()

    text = byte[sections[".text"]["offset"]:sections[".text"]["offset"] + sections[".text"]["size"]]
    strtab = byte[sections[".strtab"]["offset"]:sections[".strtab"]["offset"] + sections[".strtab"]["size"]]

    del byte

    if log:
        bp(strtab)

        strtab_names = generate_dict_section_names(strtab)
        print(strtab_names)

        for i in range(0, len(symtab), 16):
            print(i, end=' ')
            print(find_name_in_strtab(strtab, int(''.join(reversed(symtab[i:i + 4])), 16)))

        print(hex(sections[".text"]["offset"]), hex(sections[".text"]["offset"] + sections[".text"]["size"]))
```

```
        print()
    generated_symtab = generate_symtab(symtab, strtab)
    commands = decode_commands(text, sections[".text"]["address"], generated_symtab)

    if log:
        print(*commands, sep='\n')

    with open(args[1], 'w', encoding="utf-8") as f:
        f.write(".text\n")
        for i in commands:
            f.write((i[0] % tuple(i[1])) + '\n')
        f.write('\n.symtab\n')
        f.write("%s %-15s %7s %-8s %-8s %-8s %6s %s\n" %
          ("Symbol", "Value", "Size", "Type", "Bind", "Vis", "Index", "Name"))
        for i in generated_symtab:
            f.write("[%4i] 0x%-15s %5i %-8s %-8s %-8s %6s %s\n" % i)
    return



if __name__ == "__main__":
    args = ["", ""]
    try:
        args = sys.argv
        assert len(args) > 2, "Number of args is less than 2"
        args = args[1:]
    except AssertionError as e:
        print(e)
        print("So enter names files now there:")
        args = input().split()
    except Exception as e:
        print(f"Error! Program arguments isn't correct. Try Again! (Error message: {e})")
        exit(0)
```

```
try:
    main(args)
except FileNotFoundError as fnfe:
    print("Error! Filename is incorrect! Exception message:", fnfe)
except Exception as e:
    print("Other Error! Exception message:", e)
```