САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA. Ассемблер, дизассемблер»

Выполнил: Андосов Герман Андреевич

Номер ИСУ: 334875

студ. гр. M3139

Санкт-Петербург

2021

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

Инструментарий и требования к работе: работа может быть выполнена на любом из следующих языков: C/C++, Python, Java.

Теоретическая часть

Итак, задача состоит в том, чтобы программу-транслятор, с помощью которой можно преобразовывать машинный код в текст программы на языке ассемблера. Поскольку нужно раскодировать лишь часть файла, я буду говорить в отчёте лишь о том, что мне понадобилось при написании программы.

Начнем со структуры ELF-файла. Он представляет из себя набор байтов. Главные составные части файла – заголовок, таблица заголовков программы и таблица заголовков секций. По большому счету, они определяют положение содержательной части файла – секций и не только. Все части файла, кроме заголовка, могут быть расположены в произвольном месте файла. Заголовок имеет фиксированное положение – в самом начале файла.

Итак, первые 52 байта 32-битного ELF-файла – это заголовок. В свою очередь, он состоит из 16 первых идентификационных байтов, а остальные байты хранят некоторые полезные поля. Первые 4 идентификационных байта должны быть такими – 0x7f 0x45 0x4c 0x46. Их ещё называют «магическими числами», по большому счету они определяют, что данный файл имеет формат ELF. Также стоит подметить, что далее мне будет удобно записывать значение байта в шестнадцатеричной системе счисления (так же, как я записал магические числа). Следующий, пятый, байт хранит класс объектного файла.

Поскольку в рамках данной задачи файл может быть только 32-битным, то корректное значение лишь одно – 0x01. Шестой байт содержит метод кодирования данных. Снова, в рамках данной задачи есть лишь один вариант – Little Endian, которому соответствует значение 0x01. При кодировании данных методом Little Endian, байты расположены в порядке «от меньшего к большему». Пример такого кодирования приведен на рисунке №1.

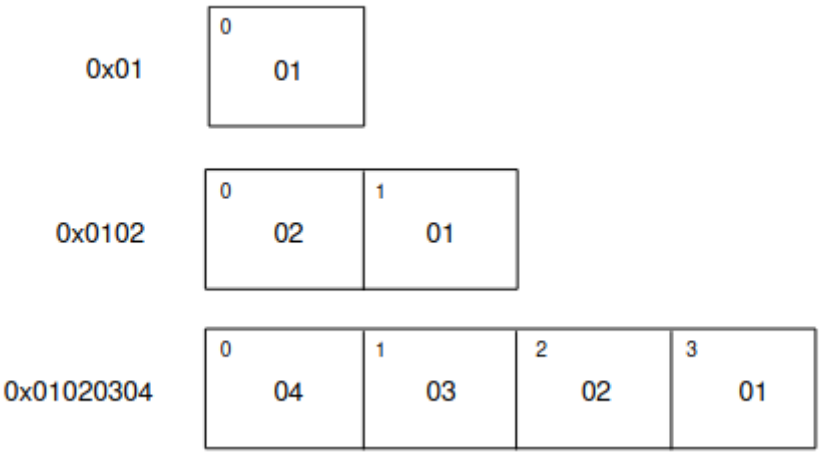


Рисунок №1 – Пример кодирования в Little Endian.

Седьмой байт содержит версию ELF-заголовка. В настоящее время существует лишь одно корректное значение – 0x01. Остальные идентификационные байты в рамках задачи не важны. Примечание: далее под словом «не важно» будет подразумеваться «не важно в рамках данной задачи».

В заголовке также хранятся некоторые полезные поля. Байты 32-35 содержат поле e_shoff – смещение таблицы заголовков секций от начала файла в байтах. Байты 46-47 содержат e_shentsize – размер одного заголовка секции. На самом деле он должен быть всегда равен 40 для 32-битных файлов. Байты 48-49 содержат e_shnum – число заголовков

секций. Наконец, байты 50-51 содержат e_shstrndx – индекс записи в таблице заголовков секций, описывающей таблицу названий секций.

Следующая важная часть файла называется таблицей заголовков секций. Она содержит атрибуты секций файла и представляет собой массив из идущих подряд заголовков секций, содержащих несколько полей. Как было сказано ранее, размер каждого заголовка секций равен 40. Он содержит 10 4-битных полей. Первое поле содержит sh_name – смещение строки, содержащей название данной секции, относительно начала таблицы названий секций. Второе поле sh_type – содержит тип заголовка. Этих типов существует много, но в рамках задачи нужны только три – тип SHT_PROGBITS означает, что секция содержит информацию, определенную программой, её формат и значение определяется программой единолично. Он кодируется значением 0x01. Тип SHT_SYMTAB означает, что секция содержит таблицу символов. Он кодируется значением 0x02. Тип SHT_STRTAB означает, что секция содержит таблицу строк. Он кодируется значением 0x03.

Четвертое поле sh_addr – содержит адрес, начиная с которого секция будет загружена. Пятое поле sh_offset – содержит смещение секции от начала файла в байтах. Шестое поле sh_size – содержит размер секции в файле. Остальные поля не важны.

В рамках данной задачи нам потребуются только три секции:

1) .symtab (тип SHT_SYMTAB)
2) .strtab (тип SHT_STRTAB)
3) .text (тип SHT_PROGBITS)

Начнем с .symtab (таблицы символов). Она содержит полезную информацию о некоторых частях программы по адресу, например, имена, размеры данных частей и так далее. Представляет собой массив из подряд записанных элементов – символов, хранящих информацию о строках таблицы. Каждый элемент состоит из шести полей:

1) st_name: хранит индекс в таблице названий секций, отвечающий за имя символа.
2) st_value: хранит адрес символа.
3) st_size: хранит размер символа.
4) st_info: особое поле, которое на самом деле содержит в себе два полезных поля – st_bind и st_type. Для получения данных полей нужно проделать следующие вычисления:

   $st\_bind = (st\_info >> 4)$

   $st\_type = st\_info \,\&\, 0xf$

   Здесь $>>$ – операция побитового сдвига вправо, & – побитовое И. st_bind определяет поведение и видимость символа при сборке. st_type хранит тип символа.

5) st_other: хранит тип видимости символа.
6) st_shndx: каждый символ связан с какой-то секцией. Данное поле хранит индекс соответствующей секции.

Типы st_bind приведены на рисунке №2.

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

Рисунок №2 – Поддерживаемые константы st_bind.

Типы st_type приведены на рисунке №3.

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

Рисунок №3 – Поддерживаемые константы st_type.

Типы st_other приведены в таблице №1:

Таблица №1 – Поддерживаемые константы st_other.

st_other	значение
STV_DEFAULT	0
STV_INTERNAL	1
STV_HIDDEN	2
STV_PROTECTED	3

Следующая интересующая нас секция – .strtab. Она хранит набор строк, разделенных специальным нулевым символом. Пример такой секции изображен на рисунке №4.

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	V	a	r
10	i	a	b	l	e	\0	a	b	l	e
20	\0	\0	x	x	\0					

Рисунок №4 – Пример .strtable.

Для того, чтобы получить какую-либо строку из такой таблицы, достаточно одного числа – сдвига начала строки относительно нулевого символа. Изначально строка будет пустой. Затем мы будем жадно брать символы и дописывать их в конец строки, пока не встретим разделяющий ноль. Примеры приведены на рисунке №5.

Index	String
0	*none*
1	`name.`
7	`Variable`
11	`able`
16	`able`
24	*null string*

Рисунок №5 – Пример получения строк из .strtable по индексу.

Наконец, последняя интересующая нас секция имеет название .text. Она содержит инструкции, выполняемые программой, которая закодирована в ELF-файле. Из условия задачи нам известно, что эти инструкции закодированы в системе RISC-V, поэтому нужно изучить

структуру данной системы кодирования. (Примечание: все дальнейшие сведения касаются 32-битной версии RISC-V).

RISC-V – это архитектура набора команд (сокращенно ISA) типа Reg-Reg (регистровая-регистровая). В распоряжении процессора есть 32 регистра, значения в которые можно записать с помощью инструкций типа LD (load), а забрать значение из регистра можно с помощью инструкций типа ST (store). Остальные инструкции отвечают за разного рода вычисления, производимые между регистрами. По умолчанию регистры обозначаются как «x#», где # – порядковый номер регистра. Но существует набор соглашений ABI (Двоичный Интерфейс Приложений), согласно которому регистры имеют другие имена (см. рис. 6).

Register	ABI Name
x0	zero
x1	ra
x2	sp
x3	gp
x4	tp
x5	t0
x6–7	t1–2
x8	s0/fp
x9	s1
x10–11	a0–1
x12–17	a2–7
x18–27	s2–11
x28–31	t3–6
f0–7	ft0–7
f8–9	fs0–1
f10–11	fa0–1
f12–17	fa2–7
f18–27	fs2–11
f28–31	ft8–11

Рисунок №6 – Названия регистров в ABI.

Помимо этого, есть еще один регистр, который называется pc (program counter) – он хранит адрес текущей инструкции.

RISC-V состоит из нескольких множеств различных инструкций, но нас в рамках задачи будут интересовать только 3:

1) RV32I
2) RV32M
3) RVC

RV32I является базовым набором команд, а RV32M и RVC – это расширения. RV32I и RV32M имеют фиксированный размер закодированной команды – 32 бита. RVC тоже имеет фиксированный размер команды – 16 бит. Как различать, к какому набору относится команда? Для этого достаточно посмотреть лишь на первый байт команды. Если два младших бита равны единице, то это команда RV32I или RV32M, а иначе – RVC.

Начнем с RV32I. Это базовый набор команд, содержащий простые арифметические операции. Все команды относятся к одному из шести типов, которые приведены на рисунке №7.

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

Рисунок №7 – Все типы инструкций RV32I и их структура.

Числа в самом вверху нужны для удобства – они показывают границы между частями одной команды. Причем здесь показано, что

команды кодируются как привычное 32-битное число, хотя на самом деле стоит помнить, что все команды закодированы в Little Endian.

Часть opcode нужна для того, чтобы различать команды. Зачастую команды со схожим предназначением имеют одинаковый opcode, поэтому их можно также отличить по блокам funct7 или funct3. rd, rs1, rs2 хранят номера регистров в двоичной системе счисления. rd – это регистр, в который будет записан результат исполнения инструкции, а rs1, rs2 – регистры, над которыми будут производиться те или иные действия. imm хранит числовой аргумент инструкции (immediate). В отличие от остальных блоков, биты в imm могут быть неупорядочены, и это хорошо видно на рисунке №7. Например, запись вида imm[12|10:5] означает, что в этом блоке хранятся 12-й, затем с 10-го по 5-й биты числового аргумента. Для каждого типа команд есть схема, по которой можно составить числовой аргумент (см. рис. 8).

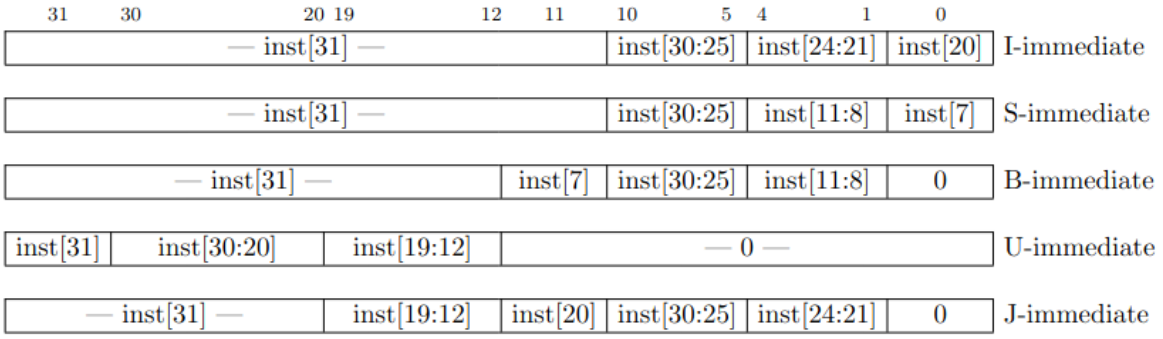


Рисунок №8 – Схема составления числовых аргументов (immediate) для различных типов команд в RV32I.

Здесь inst[x] означает бит, который стоит на месте x в закодированной инструкции. Важное замечание: вид кодирования immediate’ов – дополнение до двух. То есть, для любого разряда i от 0 до

30, вес этого разряда равен 2^i, а вес последнего, 31-го разряда равняется -2^{31}.

А так выглядит полный список команд RV32I (см. рис. 9).

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
0000 | pred	succ	00000	000	00000	0001111	FENCE
0000 | 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

Рисунок №9 – Все команды RV32I.

Команды FENCE и FENCE.I в рамках задачи нас не интересуют. Осталось лишь понять, что такое shamt в командах SLLI, SRLI, SRAI, а также что такое csr и zimm в командах вида CSR*.

Блоки zimm и shamt, в отличие от imm, хранят беззнаковые двоичные числа в привычном формате (биты идут слева направо от четвертому к нулевому). А csr кодирует так называемые «регистры контроля и статуса», которые приведены на рисунке №10.

Number	Privilege	Name
0x001	Read/write	fflags
0x002	Read/write	frm
0x003	Read/write	fcsr
0xC00	Read-only	cycle
0xC01	Read-only	time
0xC02	Read-only	instret
0xC80	Read-only	cycleh
0xC81	Read-only	timeh
0xC82	Read-only	instreth

Рисунок №10 – Поддерживаемые регистры CSR.

Первый столбец на рисунке говорит о том, как закодирован регистр, а третий столбец содержит имя регистра.

На самом деле регистров CSR больше, и в разных источниках я находил различные множества CSR-регистров. Я решил поддерживать лишь какую-то основную их часть, а если не удается распознать, что это за регистр, то по дефолту ему будет присвоено имя unknown_csr_reg.

Наконец, осталось определить, в каком формате будут выводиться раскодированные команды:

1) Load/store команды (к которым относятся sb, sh, sw, lb, lh, lw, lbu, lhu), а также jalr выводятся так: command rd (imm)rs1;
2) Команды csr*: command rd csr rs1;
3) ecall, ebreak – просто одним словом;
4) Команды типа R (add, sub, sll, slt, sltu, xor, srl, sra, or, and): command rd, rs1, rs2;
5) Команды типа I (addi, slti, sltiu, xori, ori, andi): command rd, rs1, imm;
6) Команды типа B (beq, bne, blt, bge, bltu, bgeu): command rs1, rs2, imm;
7) jal: jal rd, imm;
8) Команды типа U (lui, auipc): command rd, imm.

Все случаи разобраны.

Перейдем к множеству команд RV32M. Оно содержит дополнительные арифметические операции, которых нет в RV32I, а именно: умножение, деление и взятие остатка. Все команды имеют одинаковую структуру (такую же, как команды R-типа в RV32I), это видно на рисунке №11.

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

Рисунок №11 – Все команды RV32M.

Раскодированные команды выглядят так: command rd, rs1, rs2.

Перейдем к множеству команд RVC. Как было сказано ранее, все команды имеют фиксированный размер – 16 бит. Это множество команд

нужно для оптимизации некоторых наиболее часто используемых команд из RV32I. Среди них сложнее выделить какие-либо классы команд, поэтому сразу приведу весь набор команд на рисунках 12-14.

15–13	12–11–10	9–8–7	6–5	4–3–2	1–0	
000	0			0	00	*Illegal instruction*
000	nzuimm[5:4|9:6|2|3]			rd′	00	C.ADDI4SPN *(RES, nzuimm=0)*
001	uimm[5:3]	rs1′	uimm[7:6]	rd′	00	C.FLD *(RV32/64)*
001	uimm[5:4|8]	rs1′	uimm[7:6]	rd′	00	C.LQ *(RV128)*
010	uimm[5:3]	rs1′	uimm[2|6]	rd′	00	C.LW
011	uimm[5:3]	rs1′	uimm[2|6]	rd′	00	C.FLW *(RV32)*
011	uimm[5:3]	rs1′	uimm[7:6]	rd′	00	C.LD *(RV64/128)*
100	—				00	*Reserved*
101	uimm[5:3]	rs1′	uimm[7:6]	rs2′	00	C.FSD *(RV32/64)*
101	uimm[5:4|8]	rs1′	uimm[7:6]	rs2′	00	C.SQ *(RV128)*
110	uimm[5:3]	rs1′	uimm[2|6]	rs2′	00	C.SW
111	uimm[5:3]	rs1′	uimm[2|6]	rs2′	00	C.FSW *(RV32)*
111	uimm[5:3]	rs1′	uimm[7:6]	rs2′	00	C.SD *(RV64/128)*

Рисунок №12 – Команды RVC, у которых два младших бита нули.

15–14–13	12	11–10	9–8–7	6–5	4–3–2	1–0	
000	0	0		0		01	C.NOP
000	nzimm[5]	rs1/rd≠0		nzimm[4:0]		01	C.ADDI *(HINT, nzimm=0)*
001	imm[11|4|9:8|10|6|7|3:1|5]					01	C.JAL *(RV32)*
001	imm[5]	rs1/rd≠0		imm[4:0]		01	C.ADDIW *(RV64/128; RES, rd=0)*
010	imm[5]	rd≠0		imm[4:0]		01	C.LI *(HINT, rd=0)*
011	nzimm[9]	2		nzimm[4|6|8:7|5]		01	C.ADDI16SP *(RES, nzimm=0)*
011	nzimm[17]	rd≠{0, 2}		nzimm[16:12]		01	C.LUI *(RES, nzimm=0; HINT, rd=0)*
100	nzuimm[5]	00	rs1′/rd′	nzuimm[4:0]		01	C.SRLI *(RV32 NSE, nzuimm[5]=1)*
100	0	00	rs1′/rd′	0		01	C.SRLI64 *(RV128; RV32/64 HINT)*
100	nzuimm[5]	01	rs1′/rd′	nzuimm[4:0]		01	C.SRAI *(RV32 NSE, nzuimm[5]=1)*
100	0	01	rs1′/rd′	0		01	C.SRAI64 *(RV128; RV32/64 HINT)*
100	imm[5]	10	rs1′/rd′	imm[4:0]		01	C.ANDI
100	0	11	rs1′/rd′	00	rs2′	01	C.SUB
100	0	11	rs1′/rd′	01	rs2′	01	C.XOR
100	0	11	rs1′/rd′	10	rs2′	01	C.OR
100	0	11	rs1′/rd′	11	rs2′	01	C.AND
100	1	11	rs1′/rd′	00	rs2′	01	C.SUBW *(RV64/128; RV32 RES)*
100	1	11	rs1′/rd′	01	rs2′	01	C.ADDW *(RV64/128; RV32 RES)*
100	1	11	—	10	—	01	*Reserved*
100	1	11	—	11	—	01	*Reserved*
101	imm[11|4|9:8|10|6|7|3:1|5]					01	C.J
110	imm[8|4:3]	rs1′		imm[7:6|2:1|5]		01	C.BEQZ
111	imm[8|4:3]	rs1′		imm[7:6|2:1|5]		01	C.BNEZ

Рисунок №13 – Команды RVC, у которых два младших бита «01».

15 14 13	12	11 10 9 8 7	6 5 4 3 2	1 0	
000	nzuimm[5]	rs1/rd≠0	nzuimm[4:0]	10	C.SLLI *(HINT, rd=0; RV32 NSE, nzuimm[5]=1)*
000	0	rs1/rd≠0	0	10	C.SLLI64 *(RV128; RV32/64 HINT; HINT, rd=0)*
001	uimm[5]	rd	uimm[4:3|8:6]	10	C.FLDSP *(RV32/64)*
001	uimm[5]	rd≠0	uimm[4|9:6]	10	C.LQSP *(RV128; RES, rd=0)*
010	uimm[5]	rd≠0	uimm[4:2|7:6]	10	C.LWSP *(RES, rd=0)*
011	uimm[5]	rd	uimm[4:2|7:6]	10	C.FLWSP *(RV32)*
011	uimm[5]	rd≠0	uimm[4:3|8:6]	10	C.LDSP *(RV64/128; RES, rd=0)*
100	0	rs1≠0	0	10	C.JR *(RES, rs1=0)*
100	0	rd≠0	rs2≠0	10	C.MV *(HINT, rd=0)*
100	1	0	0	10	C.EBREAK
100	1	rs1≠0	0	10	C.JALR
100	1	rs1/rd≠0	rs2≠0	10	C.ADD *(HINT, rd=0)*
101	uimm[5:3|8:6]		rs2	10	C.FSDSP *(RV32/64)*
101	uimm[5:4|9:6]		rs2	10	C.SQSP *(RV128)*
110	uimm[5:2|7:6]		rs2	10	C.SWSP
111	uimm[5:2|7:6]		rs2	10	C.FSWSP *(RV32)*
111	uimm[5:3|8:6]		rs2	10	C.SDSP *(RV64/128)*

Рисунок №14 – Команды RVC, у которых два младших бита «10».

На самом деле, не все из этих команд нужны нам в рамках задачи, а лишь те, которые являются сжатой версией команд из RV32I. К таковым относятся: c.addi4spn, c.lw, c.sw, c.nop, c.addi, c.jal, c.li, c.addi16sp, c.lui, c.srli, c.srai, c.addi, c.sub, c.xor, c.or, c.and, c.j, c.beqz, c.bnez, c.slli, c.lwsp, c.jr, c.mv, c.ebreak, c.jalr, c.add, c.swsp. Раскодированные версии команд во многом аналогичны их оригинальной версии в RV32I.

Осталось лишь расшифровать некоторые обозначения, использованные в рисунках 12-14. rd’, rs1’, rs2’ – это тоже регистры, но закодированные не пятью битами, а тремя. Это всё сделано потому, что 8 регистров являются наиболее используемыми – а именно с восьмого по пятнадцатый. То есть, справедлива формула:

$$r = r' + 8,$$

где r – настоящий номер регистра, а r' – закодированный в RVC-команде регистр.

Появились новые сокращения: uimm – беззнаковый числовой аргумент, nzimm – ненулевой числовой аргумент, nzuimm – ненулевой беззнаковый числовой аргумент.

Практическая часть

Программа написана на языке Java версии 11.0.11. Она состоит из 5 классов:

1) Main (занимается открытием/закрытием файлов, считыванием входных данных, обработкой ошибок. Соответственно, для запуска всей программы нужно скомпилировать этот класс и запустить его следующим образом: java Main <имя входного ELF-файла> <имя выходного файла>);
2) Source (просто интерфейс источника данных для парсера);
3) ByteSource (источник байтовых данных для парсера. Хранит и передает парсеру входные данные, а также имеет несколько полезных методов, которые может вызвать парсер);
4) ByteParser (базовый абстрактный класс парсера, умеет считывать данные, перемещать указатель в произвольное место файла, а также кидать ошибки);
5) ElfParser (самая содержательная часть, парсер ELF-файлов).

ByteSource нужен парсеру для удобной работы с входными данными. Он содержит всего два поля – массив content содержит байты исходного файла, а pos – это указатель на текущий байт (следующий байт, который прочитает парсер.

ByteSource имеет набор методов, показанный в таблице №2.

Таблица №2 – Методы класса ByteSource.

Название метода	Описание метода
ByteSource (конструктор)	Принимает массив байтов, конвертирует его в массив long'ов. К сожалению, в языке Java нет беззнаковых типов, а работать с 32-битным знаковым типом int довольно проблематично. Поэтому приходится в программе в основном работать с 64-битным типом long.
hasNext	Проверяет, правда ли, что байты исходного файла не закончились и можно вызвать метод getNext.
getNext	Возвращает текущий байт, сдвигает указатель на 1 вправо.
getPos	Возвращает порядковый номер байта, на которой сейчас находится указатель.
setPos	Переставляет указатель на другое место (иногда парсер может захотеть так сделать).
error	Бросает исключение типа ParseException в случае ошибки. Класс Main умеет ловить такие исключения.

ByteParser имеет лишь одно поле source для доступа к входным данным. ByteParser имеет набор методов, указанный в таблице №3.

Таблица №3 – Набор методов класса ByteParser.

Название метода	Описание метода
1	2
read	Читает 1 байт.
read2	Читает 2 байта и переводит их привычный вид из Little Endian'a.
read4	Читает 4 байта и переводит их привычный вид из Little Endian'a.

Продолжение таблицы №3.

1	2
expect	«Ожидает» некоторый символ. Если символ не равен ожидаемому, то вызывает ошибку.
moveTo	Переставляет указатель source’а на желаемую позицию.
jumpOver	«Перескакивает» через некоторое количество байт и ставит указатель source’а туда.
getPos	Возвращает текущую позицию указателя source’а.
error	Сообщает об ошибке.
parse	Абстрактный метод, который должен быть переопределен у всех потомков (в частности, у ElfParser).

Перейдем к ElfParser. Помимо ранее описанных в теоретической полезных полей заголовка по типу e_shoff, e_shentsize и так далее, ElfParser имеет поля, указанные в таблице №4.

Таблица №4 – Поля класса ElfParser.

Название поля	Описание поля
1	2
shstr_offset	Сдвиг в байтах относительно начала файла таблицы строк, хранящей имена секций .symtab’а.
strtab_offset	Сдвиг в байтах относительно начала файла секции .strtab.
text_begin	Сдвиг в байтах относительно начала файла секции .text.
pc	program counter, о котором было сказано в теор. части. Хранит адрес текущей инструкции.

Продолжение таблицы №4.

1	2
addrToName	HashMap, с помощью которого можно получить название метки по адресу. Эти названия хранятся в .symtab и имеют тип FUNC.
unknown_command	Просто константа, которая возвращается, если не удалось определить, что за команду подали на вход.
immOrderX	Здесь X - целое число от 1 до 7. Эти методы содержат порядок immediate'ов для некоторых команд RVC.

ElfParser имеет набор методов, указанный в таблице №5.

Таблица №5 – Набор методов класса ElfParser.

Название метода	Описание метода
1	2
TO_STT	Преобразует число st_type в тип символа.
TO_STB	Преобразует число st_bind в тип поведения и видимости символа при сборке.
TO_STV	Преобразует число st_other в тип видимости символа.
TO_SHN	Преобразует st_shndx в индекс соответствующей данному символу секции.
TO_REG	Преобразует порядковый номер регистра в название регистра в ABI.
TO_CREG	Преобразует порядковый номер сокращенного регистра в название регистра в ABI. Сокращенные регистры – это те, которые закодированы тремя битами в командах RVC.
TO_CSR	Преобразует идентификатор CSR-регистра в его имя.

Продолжение таблицы №5.

1	2
parse	Определяет, в каком порядке нужно парсить отдельные части файла и возвращает дизассемблированные команды.
parseHeader	Проверяет, что переданный файл – это действительно ELF-файл и считывает полезные константы, которыми в дальнейшем будут пользоваться другие методы.
parseSection HeaderTable	Находит сдвиг в байтах от начала файла нужных нам секций: .text, .symtab и .strtab, запускает парсинг .text и .symtab, возвращает их в дизассемблированном виде.
parseText	Возвращает дизассемблированный .text.
parseSymbolTable	Возвращает дизассемблированный .symtab.
getName	Возвращает строку в .strtab по сдвигу ее начала относительно начала .strtab.
getSectionName	Возвращает имя секции.
expectMagic	Проверяет, что первые 4 байта файла – это «магические числа».
parseRV32IOrM	Определяет, к какому набору команд относится данная 32-битная команда и возвращает ее дизассемблированную версию.
bitSubstr	Вспомогательная команда, которая захватывает некоторый подотрезок бит в числе.
normalView	Возвращает дизассемблированную команду, которая не относится к командам load/store.

Продолжение таблицы №5.

1	2
normalViewAddr	Возвращает дизассемблированную команду, которая не относится к командам load/store. В отличие от normalView, последний аргумент этого метода содержит адрес, поэтому метод дополнительно ищет метку в .symtab и дописывает название метки в конец строки. Если в .symtab по такому адресу ничего не было найдено, то записывает название в формате LOC_%05x.
loadStoreView	Возвращает дизассемблированную команду, относящуюся к командам load/store.
parseRV32I	Возвращает дизассемблированную команду, относящуюся к множеству команд RV32I.
getImmediateI	Возвращает числовой аргумент команды RV32I I-типа.
getImmediateS	Возвращает числовой аргумент команды RV32I S-типа.
getImmediateB	Возвращает числовой аргумент команды RV32I B-типа.
getImmediateU	Возвращает числовой аргумент команды RV32I U-типа.
getImmediateJ	Возвращает числовой аргумент команды RV32I J-типа.
parseRV32M	Возвращает дизассемблированную команду, относящуюся к множеству команд RV32M.
unshuffle	Переставляет биты числового аргумента в нужном порядке. Очень удобна и полезна, поскольку в RVC биты числового аргумента могут идти в хаотичном порядке.
parseRVC	Возвращает дизассемблированную команду, относящуюся к множеству команд RVC.

Общий алгоритм работы парсера такой:

1) Вызывается метод parse, который, в свою очередь, сначала вызывает parseHeader.
2) parseHeader убеждается в том, что на вход подали корректный ELF-файл и считывает полезные константы из заголовка: e_shoff, e_shentsize, e_shnum, e_shstrndx.
3) Потом parse вызывает метод parseSectionHeaderTable, который находит, где начинаются интересующие нас секции: .text, .symtab и .strtab.
4) Сначала парсится .symtab. В этом нет ничего необычного – цикл считывает все строки, составляет дизассемблированные строки, которые кладутся в StringBuilder. Помимо этого, если тип очередного символа равен FUNC, то мы кладем его в addrToName. Накопленный StringBuilder возвращается методом – это и есть дизассемблированный .symtab.
5) Далее парсится .text. В цикле определяется длина команды и в зависимости от нее вызывается метод parseRV32IOrM или parseRVC.
6) parseRV32IOrM и parseRVC пытаются понять, что это за команда с помощью огромного количества if'ов и switch'ей и некоторых вспомогательных методов. Для этого оказались очень кстати такие вспомогательные методы, как bitSubstr, unshuffle и getImmediateX. Если не удается понять, что это за команда, то по дефолту считается, что это unknown_command. Наличие такой команды в выходном файле говорит об одном из двух. Либо автор этой программы (то есть я) где-то набажил (хотя в процессе тестирования я таких команд не обнаруживал), либо файл содержит неподдерживаемые команды.

7) Как только удается понять, что это за команда, а также выделены все immediate’ы, возвращается результат одного из трех методов в зависимости от команды: normalView, normalViewAddr или loadStoreView, которые приводят команду к дизассемблированному виду в нужном формате.
8) Основной цикл .text получает дизассемблированные команды и копит их в StringBuilder’е. Когда все команды готовы, он возвращает накопленные в StringBuilder’е дизассемблированные команды.
9) Наконец, parseSectionHeaderTable берет полученные дизассемблированные секции .text и .symtab и выводит их в нужном порядке.

Пример работы программы и сравнение ее вывода с выводом некоторых стандартных утилит приведены на рисунках 15-18.

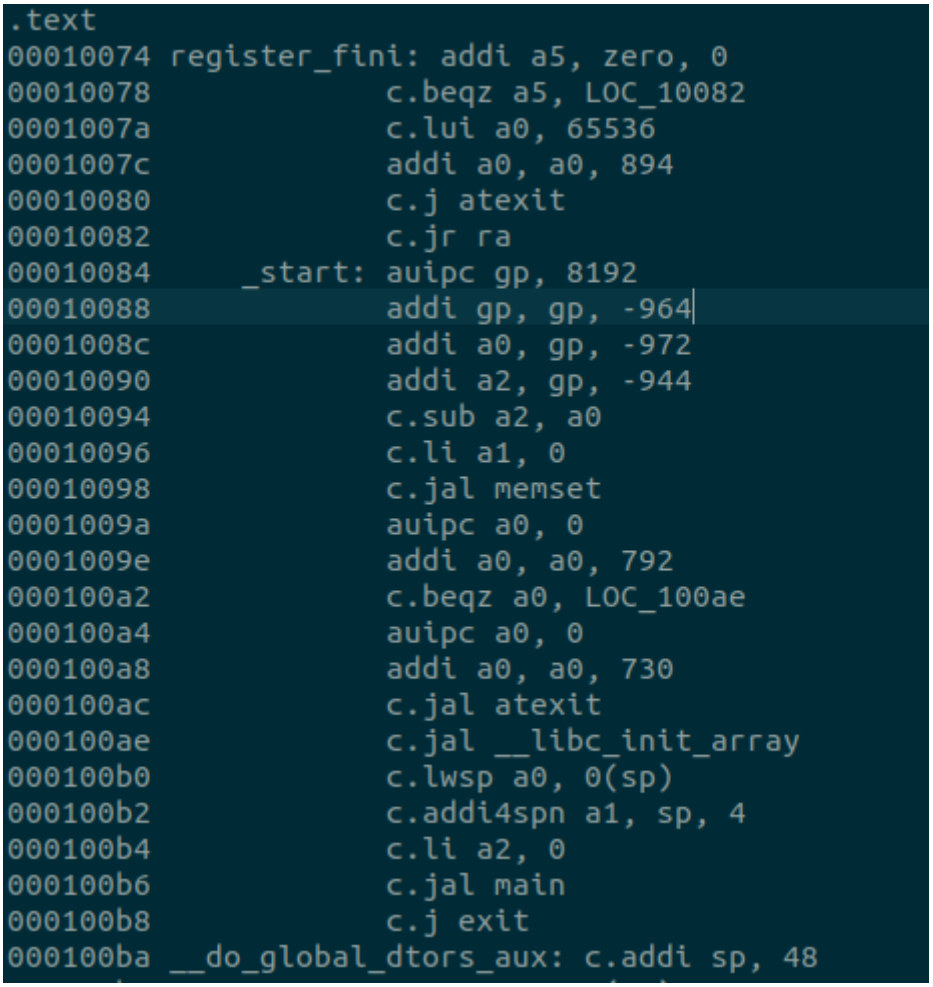


Рисунок №15 – Пример вывода .text моей программой.

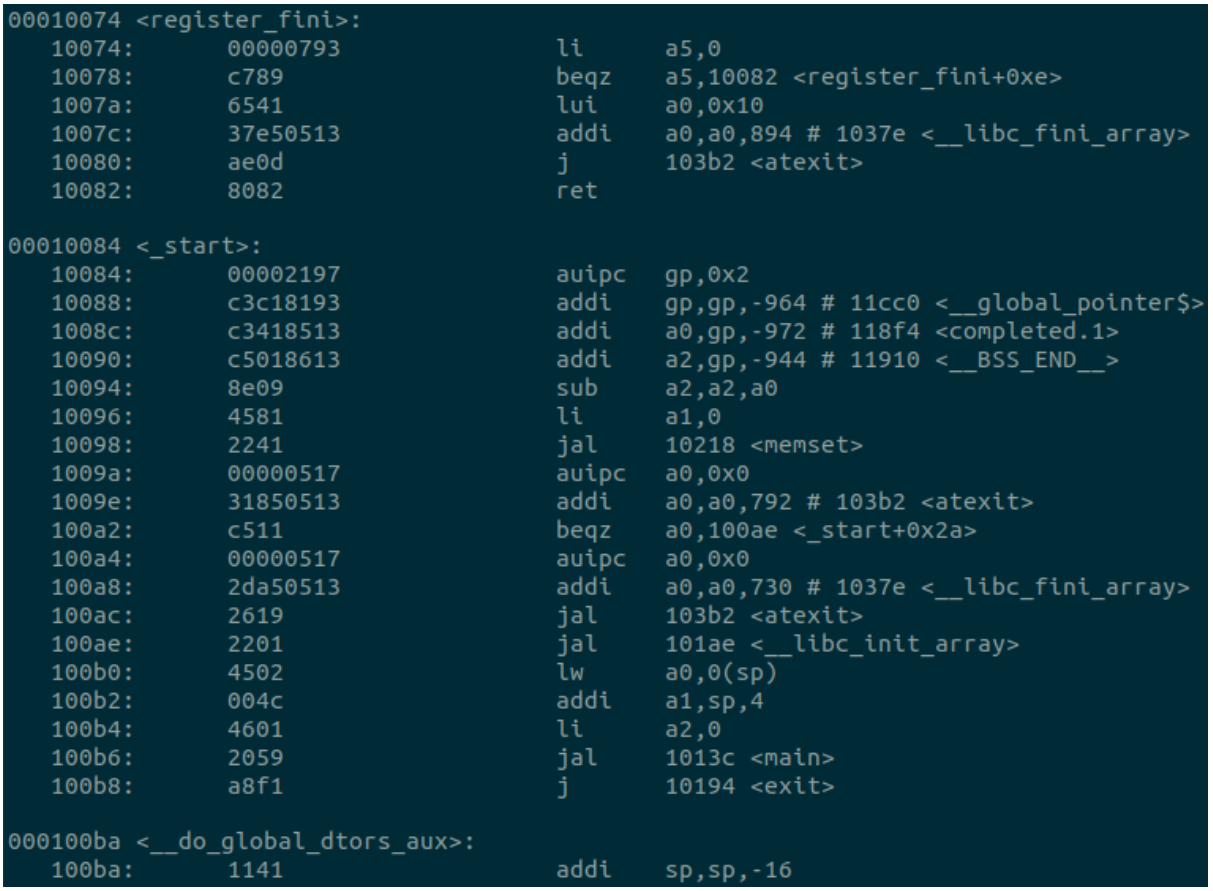
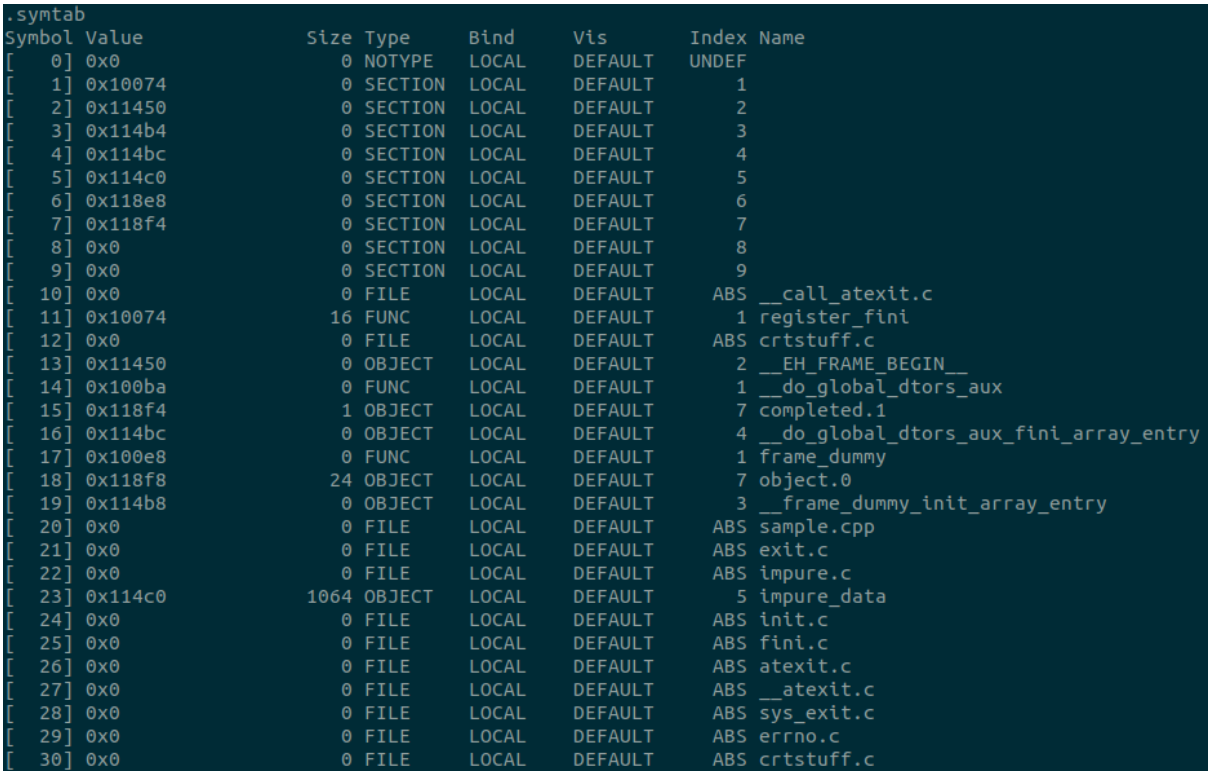


Рисунок №16 – Пример дизассемблинга того же файла, что и на рисунке 15 утилитой objdump.



```
.symtab
Symbol Value              Size Type     Bind     Vis       Index Name
[   0] 0x0                   0 NOTYPE   LOCAL    DEFAULT   UNDEF
[   1] 0x10074               0 SECTION  LOCAL    DEFAULT       1
[   2] 0x11450               0 SECTION  LOCAL    DEFAULT       2
[   3] 0x114b4               0 SECTION  LOCAL    DEFAULT       3
[   4] 0x114bc               0 SECTION  LOCAL    DEFAULT       4
[   5] 0x114c0               0 SECTION  LOCAL    DEFAULT       5
[   6] 0x118e8               0 SECTION  LOCAL    DEFAULT       6
[   7] 0x118f4               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              16 FUNC     LOCAL    DEFAULT       1 register_fini
[  12] 0x0                   0 FILE     LOCAL    DEFAULT     ABS crtstuff.c
[  13] 0x11450               0 OBJECT   LOCAL    DEFAULT       2 __EH_FRAME_BEGIN__
[  14] 0x100ba               0 FUNC     LOCAL    DEFAULT       1 __do_global_dtors_aux
[  15] 0x118f4               1 OBJECT   LOCAL    DEFAULT       7 completed.1
[  16] 0x114bc               0 OBJECT   LOCAL    DEFAULT       4 __do_global_dtors_aux_fini_array_entry
[  17] 0x100e8               0 FUNC     LOCAL    DEFAULT       1 frame_dummy
[  18] 0x118f8              24 OBJECT   LOCAL    DEFAULT       7 object.0
[  19] 0x114b8               0 OBJECT   LOCAL    DEFAULT       3 __frame_dummy_init_array_entry
[  20] 0x0                   0 FILE     LOCAL    DEFAULT     ABS sample.cpp
[  21] 0x0                   0 FILE     LOCAL    DEFAULT     ABS exit.c
[  22] 0x0                   0 FILE     LOCAL    DEFAULT     ABS impure.c
[  23] 0x114c0            1064 OBJECT   LOCAL    DEFAULT       5 impure_data
[  24] 0x0                   0 FILE     LOCAL    DEFAULT     ABS init.c
[  25] 0x0                   0 FILE     LOCAL    DEFAULT     ABS fini.c
[  26] 0x0                   0 FILE     LOCAL    DEFAULT     ABS atexit.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
```

Рисунок №17 – Пример вывода .symtab моей программой.

```
Symbol table '.symtab' contains 61 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00010074     0 SECTION LOCAL  DEFAULT    1
     2: 00011450     0 SECTION LOCAL  DEFAULT    2
     3: 000114b4     0 SECTION LOCAL  DEFAULT    3
     4: 000114bc     0 SECTION LOCAL  DEFAULT    4
     5: 000114c0     0 SECTION LOCAL  DEFAULT    5
     6: 000118e8     0 SECTION LOCAL  DEFAULT    6
     7: 000118f4     0 SECTION LOCAL  DEFAULT    7
     8: 00000000     0 SECTION LOCAL  DEFAULT    8
     9: 00000000     0 SECTION LOCAL  DEFAULT    9
    10: 00000000     0 FILE    LOCAL  DEFAULT  ABS __call_atexit.c
    11: 00010074    16 FUNC    LOCAL  DEFAULT    1 register_fini
    12: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    13: 00011450     0 OBJECT  LOCAL  DEFAULT    2 __EH_FRAME_BEGIN__
    14: 000100ba     0 FUNC    LOCAL  DEFAULT    1 __do_global_dtors_aux
    15: 000118f4     1 OBJECT  LOCAL  DEFAULT    7 completed.1
    16: 000114bc     0 OBJECT  LOCAL  DEFAULT    4 __do_global_dtors_aux_fin
    17: 000100e8     0 FUNC    LOCAL  DEFAULT    1 frame_dummy
    18: 000118f8    24 OBJECT  LOCAL  DEFAULT    7 object.0
    19: 000114b8     0 OBJECT  LOCAL  DEFAULT    3 __frame_dummy_init_array_
    20: 00000000     0 FILE    LOCAL  DEFAULT  ABS sample.cpp
    21: 00000000     0 FILE    LOCAL  DEFAULT  ABS exit.c
    22: 00000000     0 FILE    LOCAL  DEFAULT  ABS impure.c
    23: 000114c0  1064 OBJECT  LOCAL  DEFAULT    5 impure_data
    24: 00000000     0 FILE    LOCAL  DEFAULT  ABS init.c
    25: 00000000     0 FILE    LOCAL  DEFAULT  ABS fini.c
    26: 00000000     0 FILE    LOCAL  DEFAULT  ABS atexit.c
    27: 00000000     0 FILE    LOCAL  DEFAULT  ABS __atexit.c
    28: 00000000     0 FILE    LOCAL  DEFAULT  ABS sys_exit.c
    29: 00000000     0 FILE    LOCAL  DEFAULT  ABS errno.c
    30: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
```

Рисунок №18 – Пример вывода .symtab утилитой readelf.

Листинг

Компилятор: Java SDK 11.0.11.

ByteParser.java

```
import java.text.ParseException;


public abstract class ByteParser {

   protected final ByteSource source;


   protected ByteParser(final ByteSource source) {

       this.source = source;

   }
```

```
    protected long read() throws ParseException {
        if (!source.hasNext()) {
            error("Unexpected end of file");
        }
        return source.getNext();
    }

    protected long read2() throws ParseException {
        long first = read();
        return (read() << 8) | first;
    }

    protected long read4() throws ParseException {
        long[] bytes = new long[4];
        for (int i = 0; i < 4; i++) {
            bytes[i] = read();
        }
          return (bytes[3] << 24) | (bytes[2] << 16) | (bytes[1] << 8) |
bytes[0];
    }

    protected void expect(long expected) throws ParseException {
        long taken = read();
        if (taken != expected) {
            error("Expected \"" + expected + "\", but found \"" + taken);
        }
    }

    protected void moveTo(long pos) {
        source.setPos(pos);
    }

    protected void jumpOver(long length) {
```

```
        source.setPos(source.getPos() + length);
    }

    protected long getPos() {
        return source.getPos();
    }

    protected void error(String message) throws ParseException {
        source.error(message);
    }

    public abstract String parse() throws ParseException;
}
```

ByteSource.java

```
import java.text.ParseException;

public class ByteSource implements Source {
    private final long[] content;
    private long pos;

    public ByteSource(byte[] bytes) {
        long[] longs = new long[bytes.length];
        for (int i = 0; i < bytes.length; i++) {
            longs[i] = bytes[i];
            if (longs[i] < 0) {
                longs[i] += 256;
            }
        }
        this.content = longs;
    }

    public boolean hasNext() {
        return pos < content.length;
```

```
    }

    public long getNext() {
        return content[(int) pos++];
    }

    public long getPos() {
        return pos;
    }

    public void setPos(long newPos) {
        pos = newPos;
    }

    public void error(String message) throws ParseException {
            throw new ParseException("Error while parsing: " + message +
"\nPosition #", (int) pos);
    }
}
```

ElfParser.java

```
import java.text.ParseException;
import java.util.HashMap;
import java.util.Map;

public class ElfParser extends ByteParser {
    private long e_shoff;
    private long e_shentsize;
    private long e_shnum;
    private long e_shstrndx;
    private long shstr_offset;
    private long strtab_offset;
    private long textBegin;
    private long pc;
```

```
private final Map<Long, String> addrToName = new HashMap<>();

private static String TO_STT(long num) {
    switch ((int) num) {
        case 0:
            return "NOTYPE";
        case 1:
            return "OBJECT";
        case 2:
            return "FUNC";
        case 3:
            return "SECTION";
        case 4:
            return "FILE";
        default:
            if (13 <= num && num <= 15) {
                return "PROC";
            }
            return "UNKNOWN_TYPE";
    }
}

private static String TO_STB(long num) {
    switch ((int) num) {
        case 0:
            return "LOCAL";
        case 1:
            return "GLOBAL";
        case 2:
            return "WEAK";
        default:
            if (13 <= num && num <= 15) {
                return "PROC";
            }
```

```
            return "UNKNOWN_BIND";
        }
    }

    private static String TO_STV(long num) {
        switch ((int) num) {
            case 0:
                return "DEFAULT";
            case 1:
                return "INTERNAL";
            case 2:
                return "HIDDEN";
            case 3:
                return "PROTECTED";
            default:
                return "UNKNOWN_VALUE";
        }
    }

    private static String TO_SHN(long num) {
        if (num == 0) {
            return "UNDEF";
        } else if (num == 0xff00L) {
            return "BEFORE";
        } else if (num == 0xff01L) {
            return "AFTER";
        } else if (num == 0xfff1L) {
            return "ABS";
        } else if (num == 0xfff2L) {
            return "COMMON";
        } else if (num == 0xffffL) {
            return "XINDEX";
        }
        return Long.toString(num);
```

```
}

private static String TO_REG(long num) {
    switch ((int) num) {
        case 0:
            return "zero";
        case 1:
            return "ra";
        case 2:
            return "sp";
        case 3:
            return "gp";
        case 4:
            return "tp";
        case 5:
            return "t0";
        case 6:
        case 7:
            return "t" + (num - 5);
        case 8:
            return "s0";
        case 9:
            return "s1";
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
        case 16:
        case 17:
            return "a" + (num - 10);
        case 18:
        case 19:
```

```
            case 20:
            case 21:
            case 22:
            case 23:
            case 24:
            case 25:
            case 26:
            case 27:
                return "s" + (num - 16);
            case 28:
            case 29:
            case 30:
            case 31:
                return "t" + (num - 25);
            default:
                return "unkown_reg";
        }
    }

    private static String TO_CREG(long num) {
        return TO_REG(num + 8);
    }

    private static String TO_CSR(long num) {
        if (num == 0x001L) {
            return "fflags";
        } else if (num == 0x002L) {
            return "frm";
        } else if (num == 0x003L) {
            return "fcsr";
        } else if (num == 0xc00L) {
            return "cycle";
        } else if (num == 0xc01L) {
            return "time";
```

```
        } else if (num == 0xc02L) {
            return "instret";
        } else if (num == 0xc80L) {
            return "cycleh";
        } else if (num == 0xc81L) {
            return "timeh";
        } else if (num == 0xc82L) {
            return "instreth";
        }
        return "unknown_csr_reg";
    }

    private final static String unknown_command = "unknown_command";

    public ElfParser(ByteSource source) {
        super(source);
    }

    public String parse() throws ParseException {
        parseHeader();
        return parseSectionHeaderTable();
    }

    private void parseHeader() throws ParseException {
        moveTo(0);
        expectMagic();
        expect(1); // EI_CLASS = 32-bit file
        expect(1); // EI_DATA = little endian
        expect(1); // EI_VERSION = 1
        moveTo(32);
        e_shoff = read4();
        moveTo(46);
        e_shentsize = read2();
        e_shnum = read2();
```

```
        e_shstrndx = read2();
    }

    private String parseSectionHeaderTable() throws ParseException {
        moveTo(e_shoff + e_shstrndx * e_shentsize + 16);
        shstr_offset = read4();
        moveTo(e_shoff);
        // Finding .text, .symtab and .strtab
        long symtabPos = 0;
        long symtabSize = 0;
        long textPos = 0;
        long textSize = 0;
        for (int i = 0; i < e_shnum; i++) {
            long sh_name = read4();
            long sh_type = read4();
            if (sh_type == 2 && ".symtab".equals(getSectionName(sh_name))) {
                jumpOver(8);
                symtabPos = read4();
                symtabSize = read4();
                jumpOver(16);
                continue;
                                    } else if (sh_type == 1 &&
".text".equals(getSectionName(sh_name))) {
                jumpOver(4);
                textBegin = read4();
                textPos = read4();
                textSize = read4();
                jumpOver(16);
                continue;
                                    } else if (sh_type == 3 &&
".strtab".equals(getSectionName(sh_name))) {
                jumpOver(8);
                strtab_offset = read4();
                jumpOver(20);
                continue;
```

```
            }
            jumpOver(32);
        }
        StringBuilder result = new StringBuilder();
        String symTabView;
        if (symtabPos != 0) {
            symTabView = parseSymbolTable(symtabPos, symtabSize);
        } else {
            symTabView = "";
        }
        result.append(".text\n");
        if (textPos != 0) {
            result.append(parseText(textPos, textSize));
        }
        result.append("\n.symtab\n").append(symTabView);
        return result.toString();
    }

        private String parseText(long textPos, long textSize) throws
ParseException {
        long prevPos = getPos();
        moveTo(textPos);
        StringBuilder sb = new StringBuilder();
        pc = textBegin;
        for (int i = 0; i < textSize; i += 2) {
            long first = read2();
            if ((first & 3) == 3) {
                long second = read2();
                i += 2;
                String command = parseRV32IOrM((second << 16) | first);
                    String name = addrToName.getOrDefault(textBegin + i - 2,
"");
                if (name.length() > 0) {
                     sb.append(String.format("%08x %10s: %s\n", textBegin + i
- 2, name, command));
```

```
                } else {
                    sb.append(String.format("%08x %11s %s\n", textBegin + i
- 2, name, command));
                }
                pc += 4;
            } else {
                String command = parseRVC(first);
                String name = addrToName.getOrDefault(textBegin + i, "");
                if (name.length() > 0) {
                    sb.append(String.format("%08x %10s: %s\n", textBegin +
i, name, command));
                } else {
                    sb.append(String.format("%08x %11s %s\n", textBegin + i,
name, command));
                }
                pc += 2;
            }
        }
        moveTo(prevPos);
        return sb.toString();
    }


    private String parseSymbolTable(long symtabPos, long symtabSize) throws
ParseException {
        long prevPos = getPos();
        moveTo(symtabPos);
        StringBuilder symtab = new StringBuilder();
        symtab.append(String.format("%s %-15s %7s %-8s %-8s %-8s %6s %s\n",
"Symbol", "Value", "Size",
                "Type", "Bind", "Vis", "Index", "Name"));
        for (int i = 0; i < symtabSize; i += 16) {
            long st_name = read4();
            long st_value = read4();
            long st_size = read4();
            long st_info = read();
            long st_other = read();
```

```
            long st_shndx = read2();

            long st_bind = st_info >> 4;
            long st_type = st_info & 0xf;
            String index = TO_SHN(st_shndx);
            String name = getName(st_name);

               symtab.append(String.format("[%4d] 0x%-15s %5d %-8s %-8s %-8s
%6s %s\n", i / 16, Long.toHexString(st_value),
                                    st_size, TO_STT(st_type), TO_STB(st_bind),
TO_STV(st_other & 3), index, name));

            if (TO_STT(st_type).equals("FUNC")) {
                addrToName.put(st_value, name);
            }
        }
        moveTo(prevPos);
        return symtab.toString();
    }

    String getName(long offset) throws ParseException {
        long prevPos = getPos();
        moveTo(strtab_offset);
        jumpOver(offset);
        StringBuilder name = new StringBuilder();
        do {
            name.append((char) read());
        } while (name.charAt(name.length() - 1) != 0);
        name.deleteCharAt(name.length() - 1);
        moveTo(prevPos);
        return name.toString();
    }

    String getSectionName(long offset) throws ParseException {
        long prevPos = getPos();
```

```
        moveTo(shstr_offset);
        jumpOver(offset);
        StringBuilder name = new StringBuilder();
        do {
            name.append((char) read());
        } while (name.charAt(name.length() - 1) != 0);
        name.deleteCharAt(name.length() - 1);
        moveTo(prevPos);
        return name.toString();
    }

    private void expectMagic() throws ParseException {
        // Magic 7f 45 4c 46
        expect(0x7f);
        expect(0x45);
        expect(0x4c);
        expect(0x46);
    }

    private String parseRV32IOrM(long mask) {
        long opcode = bitSubstr(mask, 6, 0);
        long funct7 = bitSubstr(mask, 31, 25);
        if (opcode == 0b0110011 && funct7 == 1) {
            return parseRV32M(mask);
        }
        return parseRV32I(mask);
    }

    private long bitSubstr(long value, int r, int l) {
        long andMask = (1L << (r + 1)) - 1 - ((1L << l) - 1);
        return (andMask & value) >> l;
    }

    private String normalView(String[] command) {
```

```
        StringBuilder sb = new StringBuilder();
        if (command.length == 1) {
            return command[0];
        }
        sb.append(command[0]).append(' ');
        for (int i = 1; i < command.length - 1; i++) {
            sb.append(command[i]).append(", ");
        }
        sb.append(command[command.length - 1]);
        return sb.toString();
    }

    private String normalViewAddr(String[] command) {
        StringBuilder sb = new StringBuilder();
        sb.append(command[0]).append(' ');
        for (int i = 1; i < command.length - 2; i++) {
            sb.append(command[i]).append(", ");
        }
        if (command.length > 2) {
            sb.append(command[command.length - 2]).append(", ");
        }
        long addr = Long.parseLong(command[command.length - 1], 16);
           sb.append(addrToName.getOrDefault(addr, String.format("LOC_%05x",
addr)));
        return sb.toString();
    }

    private String loadStoreView(String[] command) {
        StringBuilder sb = new StringBuilder();
        sb.append(command[0]).append(' ');
        for (int i = 1; i < command.length - 2; i++) {
            sb.append(command[i]).append(", ");
        }
                                        sb.append(command[command.length      -
2]).append('(').append(command[command.length - 1]).append(')');
```

```
        return sb.toString();
    }

    private String parseRV32I(long mask) {
        long rd = bitSubstr(mask, 11, 7);
        long rs1 = bitSubstr(mask, 19, 15);
        long rs2 = bitSubstr(mask, 24, 20);
        long funct7 = bitSubstr(mask,31, 25);
        long funct3 = bitSubstr(mask, 14, 12);
        long opcode = bitSubstr(mask, 6, 0);
        long imm110 = bitSubstr(mask, 31, 20);
        String rdReg = TO_REG(rd);
        String rs1Reg = TO_REG(rs1);
        String rs2Reg = TO_REG(rs2);
        String offset;
        switch ((int) opcode) {
            case 0b0110011:
                if (funct7 == 0) {
                    switch ((int) funct3) {
                        case 0b000:
                                return normalView(new String[]{"add", rdReg,
rs1Reg, rs2Reg});
                        case 0b001:
                                return normalView(new String[]{"sll", rdReg,
rs1Reg, rs2Reg});
                        case 0b010:
                                return normalView(new String[]{"slt", rdReg,
rs1Reg, rs2Reg});
                        case 0b011:
                               return normalView(new String[]{"sltu", rdReg,
rs1Reg, rs2Reg});
                        case 0b100:
                                return normalView(new String[]{"xor", rdReg,
rs1Reg, rs2Reg});
                        case 0b101:
```

```
                                return normalView(new String[]{"srl", rdReg,
rs1Reg, rs2Reg});
                        case 0b110:
                                 return normalView(new String[]{"or", rdReg,
rs1Reg, rs2Reg});
                        case 0b111:
                                return normalView(new String[]{"and", rdReg,
rs1Reg, rs2Reg});
                        default:
                            return unknown_command;
                    }
                } else if (funct7 == (1L << 5)) {
                    if (funct3 == 0b000) {
                        return normalView(new String[]{"sub", rdReg, rs1Reg,
rs2Reg});
                    } else if (funct3 == 0b101) {
                        return normalView(new String[]{"sra", rdReg, rs1Reg,
rs2Reg});
                    }
                    return unknown_command;
                } else {
                    return unknown_command;
                }
            case 0b0010011:
                switch ((int) funct3) {
                    case 0b000:
                                return normalView(new String[]{"addi", rdReg,
rs1Reg, getImmediateI(mask)});
                    case 0b010:
                                return normalView(new String[]{"slti", rdReg,
rs1Reg, getImmediateI(mask)});
                    case 0b011:
                               return normalView(new String[]{"sltiu", rdReg,
rs1Reg, getImmediateI(mask)});
                    case 0b100:
                                return normalView(new String[]{"xori", rdReg,
rs1Reg, getImmediateI(mask)});
                    case 0b110:
```

```
                        return normalView(new String[]{"ori", rdReg, rs1Reg,
getImmediateI(mask)});
                    case 0b111:
                                return normalView(new String[]{"andi", rdReg,
rs1Reg, getImmediateI(mask)});
                    case 0b001:
                        if (funct7 == 0) {
                                return normalView(new String[]{"slli", rdReg,
rs1Reg, Long.toString(rs2)});
                        }
                        return unknown_command;
                    case 0b101:
                        if (funct7 == (1L << 5)) {
                                return normalView(new String[]{"srai", rdReg,
rs1Reg, Long.toString(rs2)});
                        } else if (funct7 == 0) {
                                return normalView(new String[]{"srli", rdReg,
rs1Reg, Long.toString(rs2)});
                        }
                        return unknown_command;
                    default:
                        return unknown_command;
                }
            case 0b0100011:
                switch ((int) funct3) {
                    case 0b000:
                                return loadStoreView(new String[]{"sb", rs2Reg,
getImmediateS(mask), rs1Reg});
                    case 0b001:
                                return loadStoreView(new String[]{"sh", rs2Reg,
getImmediateS(mask), rs1Reg});
                    case 0b010:
                                return loadStoreView(new String[]{"sw", rs2Reg,
getImmediateS(mask), rs1Reg});
                    default:
                        return unknown_command;
                }
```

```
            case 0b0000011:
                switch ((int) funct3) {
                    case 0b000:
                                return loadStoreView(new String[]{"lb", rdReg,
getImmediateI(mask), rs1Reg});
                    case 0b001:
                                return loadStoreView(new String[]{"lh", rdReg,
getImmediateI(mask), rs1Reg});
                    case 0b010:
                                return loadStoreView(new String[]{"lw", rdReg,
getImmediateI(mask), rs1Reg});
                    case 0b100:
                               return loadStoreView(new String[]{"lbu", rdReg,
getImmediateI(mask), rs1Reg});
                    case 0b101:
                               return loadStoreView(new String[]{"lhu", rdReg,
getImmediateI(mask), rs1Reg});
                    default:
                        return unknown_command;
                }
            case 0b1100011:
                                          offset = Long.toHexString(pc +
Long.parseLong(getImmediateB(mask)));
                switch ((int) funct3) {
                    case 0b000:
                          return normalViewAddr(new String[]{"beq", rs1Reg,
rs2Reg, offset});
                    case 0b001:
                          return normalViewAddr(new String[]{"bne", rs1Reg,
rs2Reg, offset});
                    case 0b100:
                          return normalViewAddr(new String[]{"blt", rs1Reg,
rs2Reg, offset});
                    case 0b101:
                          return normalViewAddr(new String[]{"bge", rs1Reg,
rs2Reg, offset});
                    case 0b110:
```

```
                        return normalViewAddr(new String[]{"bltu", rs1Reg,
rs2Reg, offset});
                    case 0b111:
                        return normalViewAddr(new String[]{"bgeu", rs1Reg,
rs2Reg, offset});
                    default:
                        return unknown_command;
                }
            case 0b1100111:
                if (funct3 == 0b000) {
                    return loadStoreView(new String[]{"jalr", rdReg,
getImmediateI(mask), rs1Reg});
                }
                return unknown_command;
            case 0b1101111:
                                        offset = Long.toHexString(pc +
Long.parseLong(getImmediateJ(mask)));
                return normalViewAddr(new String[]{"jal", rdReg, offset});
            case 0b0010111:
                    return normalView(new String[]{"auipc", rdReg,
getImmediateU(mask)});
            case 0b0110111:
                    return normalView(new String[]{"lui", rdReg,
getImmediateU(mask)});
            case 0b1110011:
                switch ((int) funct3) {
                    case 0b000:
                        if (rs1 == 0 && rd == 0) {
                            if (imm110 == 0) {
                                return normalView(new String[]{"ecall"});
                            } else if (imm110 == 1) {
                                return normalView(new String[]{"ebreak"});
                            }
                            return unknown_command;
                        }
                    case 0b001:
```

```
                            return normalView(new String[]{"csrrw", rdReg,
TO_CSR(imm110), rs1Reg});
                    case 0b010:
                            return normalView(new String[]{"csrrs", rdReg,
TO_CSR(imm110), rs1Reg});
                    case 0b011:
                            return normalView(new String[]{"csrrc", rdReg,
TO_CSR(imm110), rs1Reg});
                    case 0b101:
                           return normalView(new String[]{"csrrwi", rdReg,
TO_CSR(imm110), Long.toString(rs1)});
                    case 0b110:
                           return normalView(new String[]{"csrrsi", rdReg,
TO_CSR(imm110), Long.toString(rs1)});
                    case 0b111:
                           return normalView(new String[]{"csrrci", rdReg,
TO_CSR(imm110), Long.toString(rs1)});
                }
            default:
                return unknown_command;
        }
    }

    private String getImmediateI(long mask) {
        long res = 0;
        if (bitSubstr(mask, 31, 31) > 0) {
            res = 0b011_111_111_111_111_111_111_000_000_000_000L;
        }
        res |= bitSubstr(mask, 31, 20);
        if (bitSubstr(res, 31, 31) > 0) {
            res -= (1L << 32);
        }
        return Long.toString(res);
    }

    private String getImmediateS(long mask) {
```

```
        long res = 0;
        if (bitSubstr(mask, 31, 31) > 0) {
            res = 0b011_111_111_111_111_111_111_000_000_000_000L;
        }
        res |= (bitSubstr(mask, 31, 25) << 5);
        res |= bitSubstr(mask, 11, 7);
        if (bitSubstr(res, 31, 31) > 0) {
            res -= (1L << 32);
        }
        return Long.toString(res);
    }

    private String getImmediateB(long mask) {
        long res = 0;
        if (bitSubstr(mask, 31, 31) > 0) {
            res = 0b011_111_111_111_111_111_111_000_000_000_000L;
        }
        res |= (bitSubstr(mask, 7, 7) << 11);
        res |= (bitSubstr(mask, 30, 25) << 5);
        res |= (bitSubstr(mask, 11, 8) << 1);
        if (bitSubstr(res, 31, 31) > 0) {
            res -= (1L << 32);
        }
        return Long.toString(res);
    }

    private String getImmediateU(long mask) {
        long res = bitSubstr(mask, 31, 12) << 12;
        if (bitSubstr(res, 31, 31) > 0) {
            res -= (1L << 32);
        }
        return Long.toString(res);
    }
```

```
    private String getImmediateJ(long mask) {
        long res = 0;
        if (bitSubstr(mask, 31, 31) > 0) {
            res = 0b011_111_111_111_100_000_000_000_000_000_000L;
        }
        res |= (bitSubstr(mask, 19, 12) << 12);
        res |= (bitSubstr(mask, 20, 20) << 11);
        res |= (bitSubstr(mask, 30, 25) << 5);
        res |= (bitSubstr(mask, 24, 21) << 1);
        if (bitSubstr(res, 31, 31) > 0) {
            res -= (1L << 32);
        }
        return Long.toString(res);
    }

    private String parseRV32M(long mask) {
        long funct3 = bitSubstr(mask,14, 12);
        long rd = bitSubstr(mask, 11, 7);
        long rs1 = bitSubstr(mask, 19, 15);
        long rs2 = bitSubstr(mask, 24, 20);
        String rdReg = TO_REG(rd);
        String rs1Reg = TO_REG(rs1);
        String rs2Reg = TO_REG(rs2);
        switch ((int) funct3) {
            case 0:
                    return normalView(new String[]{"mul", rdReg, rs1Reg,
rs2Reg});
            case 1:
                   return normalView(new String[]{"mulh", rdReg, rs1Reg,
rs2Reg});
            case 2:
                  return normalView(new String[]{"mulhsu", rdReg, rs1Reg,
rs2Reg});
            case 3:
                    return normalView(new String[]{"mulhu", rdReg, rs1Reg,
rs2Reg});
```

```
            case 4:
                        return normalView(new String[]{"div", rdReg, rs1Reg,
rs2Reg});
            case 5:
                       return normalView(new String[]{"divu", rdReg, rs1Reg,
rs2Reg});
            case 6:
                        return normalView(new String[]{"rem", rdReg, rs1Reg,
rs2Reg});
            case 7:
                       return normalView(new String[]{"remu", rdReg, rs1Reg,
rs2Reg});
            default:
                return unknown_command;
        }
    }

    private long unshuffle(long mask, int[] order, boolean signed) {
        long result = 0;
        for (int i = 0; i < order.length; i++) {
            long hit = Math.min(1, mask & (1L << i));
            if (i == order.length - 1 && signed) {
                result -= (hit << order[order.length - i - 1]);
            } else {
                result += (hit << order[order.length - i - 1]);
            }
        }
        return result;
    }

    // Immediate bits' orders for RVC
     private static final int[] immOrder1 = new int[]{5, 4, 9, 8, 7, 6, 2,
3};
    private static final int[] immOrder2 = new int[]{5, 4, 3, 2, 6};
     private static final int[] immOrder3 = new int[]{11, 4, 9, 8, 10, 6, 7,
3, 2, 1, 5};
```

```
   private static final int[] immOrder4 = new int[]{9, 4, 6, 8, 7, 5};

    private static final int[] immOrder5 = new int[]{17, 16, 15, 14, 13,
12};

    private static final int[] immOrder6 = new int[]{8, 4, 3, 7, 6, 2, 1,
5};

   private static final int[] immOrder7 = new int[]{5, 4, 3, 2, 7, 6};


   private String parseRVC(long mask) {

       String offset;

       switch ((int) bitSubstr(mask, 1, 0)) {

           case 0b00:

               switch ((int) bitSubstr(mask, 15, 13)) {

                   case 0b000:

                       if (bitSubstr(mask, 12, 2) == 0) {

                                                          return  normalView(new
String[]{"illegal_instruction"});

                       }

                                  return normalView(new String[]{"c.addi4spn",
TO_CREG(bitSubstr(mask, 4, 2)),

                                                                 "sp",
Long.toString(unshuffle(bitSubstr(mask, 12, 5), immOrder1, false))});

                   case 0b010:

                                    return loadStoreView(new String[]{"c.lw",
TO_CREG(bitSubstr(mask, 4, 2)),

                                     Long.toString(unshuffle(bitSubstr(mask, 12,
10) * 4 + bitSubstr(mask, 6, 5), immOrder2, false)),

                                    TO_CREG(bitSubstr(mask, 9, 7))});

                   case 0b110:

                                    return loadStoreView(new String[]{"c.sw",
TO_CREG(bitSubstr(mask, 4, 2)),

                                     Long.toString(unshuffle(bitSubstr(mask, 12,
10) * 4 + bitSubstr(mask, 6, 5), immOrder2, false)),

                                    TO_CREG(bitSubstr(mask, 9, 7))});

                   default:

                       return unknown_command;

               }

           case 0b01:

               switch ((int) bitSubstr(mask, 15, 13)) {
```

```
                    case 0b000:
                        if (bitSubstr(mask, 12, 2) == 0) {
                            return normalView(new String[]{"c.nop"});
                        } else if (bitSubstr(mask, 11, 7) != 0) {
                                return normalView(new String[]{"c.addi",
TO_REG(bitSubstr(mask, 11, 7)),
                                        Long.toString(bitSubstr(mask, 12, 12) *
32 + bitSubstr(mask, 6, 2))});
                        }
                        return unknown_command;
                    case 0b001:
                                            offset = Long.toHexString(pc +
unshuffle(bitSubstr(mask, 12, 2), immOrder3, true));
                                return normalViewAddr(new String[]{"c.jal",
offset});
                    case 0b010:
                                        return normalView(new String[]{"c.li",
TO_REG(bitSubstr(mask, 11, 7)),
                                        Long.toString(bitSubstr(mask, 12, 12) *
(-32) + bitSubstr(mask, 6, 2))});
                    case 0b011:
                        if (bitSubstr(mask, 11, 7) == 2) {
                                return normalView(new String[]{"c.addi16sp",
TO_REG(bitSubstr(mask, 11, 7)),
                                                                        "sp",
Long.toString(unshuffle(bitSubstr(mask, 12, 12) * 32 + bitSubstr(mask, 6,
2), immOrder4, true))});
                        } else if (bitSubstr(mask, 11, 7) != 0) {
                                return normalView(new String[]{"c.lui",
TO_REG(bitSubstr(mask, 11, 7)),
                                        Long.toString(unshuffle(bitSubstr(mask,
12, 12) * 32 + bitSubstr(mask, 6, 2), immOrder5, true))});
                        }
                        return unknown_command;
                    case 0b100:
                        switch ((int) bitSubstr(mask, 11, 10)) {
                            case 0:
                                return normalView(new String[]{"c.srli",
TO_CREG(bitSubstr(mask, 9, 7)),
```

```
                                        Long.toString(bitSubstr(mask, 12,
12) * 32 + bitSubstr(mask, 6, 2))});
                            case 1:
                                    return normalView(new String[]{"c.srai",
TO_CREG(bitSubstr(mask, 9, 7)),
                                        Long.toString(bitSubstr(mask, 12,
12) * 32 + bitSubstr(mask, 6, 2))});
                            case 2:
                                    return normalView(new String[]{"c.andi",
TO_CREG(bitSubstr(mask, 9, 7)),
                                        Long.toString(bitSubstr(mask, 12,
12) * (-32) + bitSubstr(mask, 6, 2))});
                            default:
                                switch ((int) bitSubstr(mask, 6, 5)) {
                                    case 0:
                                            return normalView(new
String[]{"c.sub", TO_CREG(bitSubstr(mask, 9, 7)),
                                                TO_CREG(bitSubstr(mask, 4,
2))});
                                    case 1:
                                            return normalView(new
String[]{"c.xor", TO_CREG(bitSubstr(mask, 9, 7)),
                                                TO_CREG(bitSubstr(mask, 4,
2))});
                                    case 2:
                                            return normalView(new
String[]{"c.or", TO_CREG(bitSubstr(mask, 9, 7)),
                                                TO_CREG(bitSubstr(mask, 4,
2))});
                                    case 3:
                                            return normalView(new
String[]{"c.and", TO_CREG(bitSubstr(mask, 9, 7)),
                                                TO_CREG(bitSubstr(mask, 4,
2))});
                                }
                        }
                    case 0b101:
                                        offset = Long.toHexString(pc +
unshuffle(bitSubstr(mask, 12, 2), immOrder3, true));
```

```
                    return normalViewAddr(new String[]{"c.j", offset});
                case 0b110:
                                        offset = Long.toHexString(pc +
unshuffle(bitSubstr(mask, 12, 10) * 32 + bitSubstr(mask, 6, 2), immOrder6,
true));
                                return normalViewAddr(new String[]{"c.beqz",
TO_CREG(bitSubstr(mask, 9, 7)), offset});
                case 0b111:
                                        offset = Long.toHexString(pc +
unshuffle(bitSubstr(mask, 12, 10) * 32 + bitSubstr(mask, 6, 2), immOrder6,
true));
                                return normalViewAddr(new String[]{"c.bnez",
TO_CREG(bitSubstr(mask, 9, 7)), offset});
            }
        case 0b10:
            switch ((int) bitSubstr(mask, 15, 13)) {
                case 0b000:
                                    return normalView(new String[]{"c.slli",
TO_REG(bitSubstr(mask, 11, 7)),
                                Long.toString(bitSubstr(mask, 12, 12) * 32 +
bitSubstr(mask, 6, 2))});
                case 0b010:
                    offset = Long.toString(unshuffle(bitSubstr(mask, 12,
12) * 32 + bitSubstr(mask, 6, 2), immOrder7, false));
                                return loadStoreView(new String[]{"c.lwsp",
TO_REG(bitSubstr(mask, 11, 7)), offset, "sp"});
                case 0b100:
                    if (bitSubstr(mask, 12, 12) == 0) {
                        if (bitSubstr(mask, 11, 7) == 0) {
                            return unknown_command;
                        }
                        if (bitSubstr(mask, 6, 2) == 0) {
                                    return normalView(new String[]{"c.jr",
TO_REG(bitSubstr(mask, 11, 7))});
                        } else {
                                    return normalView(new String[]{"c.mv",
TO_REG(bitSubstr(mask, 11, 7)),
                                      TO_REG(bitSubstr(mask, 6, 2))});
                        }
```

```
                        } else {
                            if (bitSubstr(mask, 11, 7) == 0) {
                                if (bitSubstr(mask, 6, 2) == 0) {
                                                    return normalView(new
String[]{"c.ebreak"});
                                }
                                return unknown_command;
                            }
                            if (bitSubstr(mask, 6, 2) == 0) {
                                    return normalView(new String[]{"c.jalr",
TO_REG(bitSubstr(mask, 11, 7))});
                            } else {
                                     return normalView(new String[]{"c.add",
TO_REG(bitSubstr(mask, 11, 7)),
                                        TO_REG(bitSubstr(mask, 6, 2))});
                            }
                        }
                    case 0b110:
                        offset = Long.toString(unshuffle(bitSubstr(mask, 12,
7), immOrder7, false));
                                 return loadStoreView(new String[]{"c.swsp",
TO_REG(bitSubstr(mask, 6, 2)), offset, "sp"});
                }
            default:
                return unknown_command;
        }
    }
}
```

Main.java

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.text.ParseException;

public class Main {
```

```
    public static void main(String[] args) {
        if (args.length != 2) {
                System.err.println("Please, enter two arguments - input and
output file names");
            return;
        }
        File inputFile = new File(args[0]);
        byte[] fileContent;
        try {
            fileContent = Files.readAllBytes(inputFile.toPath());
        } catch (IOException e) {
            System.err.println("Sorry, an error occurred while reading input
file " + e.getMessage());
            return;
        }
        ByteSource source = new ByteSource(fileContent);
        ByteParser parser = new ElfParser(source);
                try (BufferedWriter writer = new BufferedWriter(new
FileWriter(args[1], StandardCharsets.UTF_8))) {
            writer.write(parser.parse());
        } catch (ParseException e) {
            System.err.println("The input file was probably incorrect :( \n"
+ e.getMessage());
        } catch (IOException e) {
            System.err.println("Sorry, an error occurred while output");
        }
    }
}
```

Source.java

```
import java.text.ParseException;

public interface Source {
    boolean hasNext();
    long getNext();
    void setPos(long newPos);
```

```
    void error(String message) throws ParseException;
}
```