Лабораторная работа №3	M3136	2022
ISA	Тарасевич Артем Сергеевич	

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

Инструментарий и требования к работе: OpenJDK Java 19.0.1, версия компилятора javac - 17.0.5

Описание системы кодирования RISC-V.

RISC-V - архитектура набора команд (далее ISA - Instruction Set Architecture), где в идеологии лежит концепция RISC (reduced instruction set computer) - простота декодирования команд, что влечет за собой повышенную скорость выполнения.

RISC-V поддерживает принцип модульностей. Любое устройство должно реализовать стандартный набор команд, и, при желании, может реализовывать расширения (как стандартные, так и пользовательские)

Любое устройство, реализующее RISC-V, должно реализовать стандартный набор команд - RV32I (если процессор 64-битный, то и RV64I, что является надмножеством RV32I), который состоит из 40 команд.

В распоряжении команд RISC-V имеется 32 обычных регистра и 1 служебный “pc”, отвечающий за адрес текущей команды). Все регистры фиксированного размера (чаще всего, 32 бит). Так как RISC-V - General Purpose Register (не стековская и не аккумуляторная), то обращаться к этим регистрам можно независимо от других.

Кроме того, RISC-V - GPD Load-Store архитектура:

- Данные должны явно перемещаться между регистрами и памятью
- ALU (арифметико-вычислительное устройство) должно использовать только регистры
- Чаще всего команды принимают в себя три аргумента

Что касается регистров, регистр x0 всегда состоит только из битов, равных нулю (hardwired). Остальные регистры имеют разную природу использования (как бы они физически могут быть устроены одинаково, но

каждому регистру присвоен свой смысл использования). Например, по конвенции, регистр x1 хранит адрес вызова, а x2 - указатель на стеке.

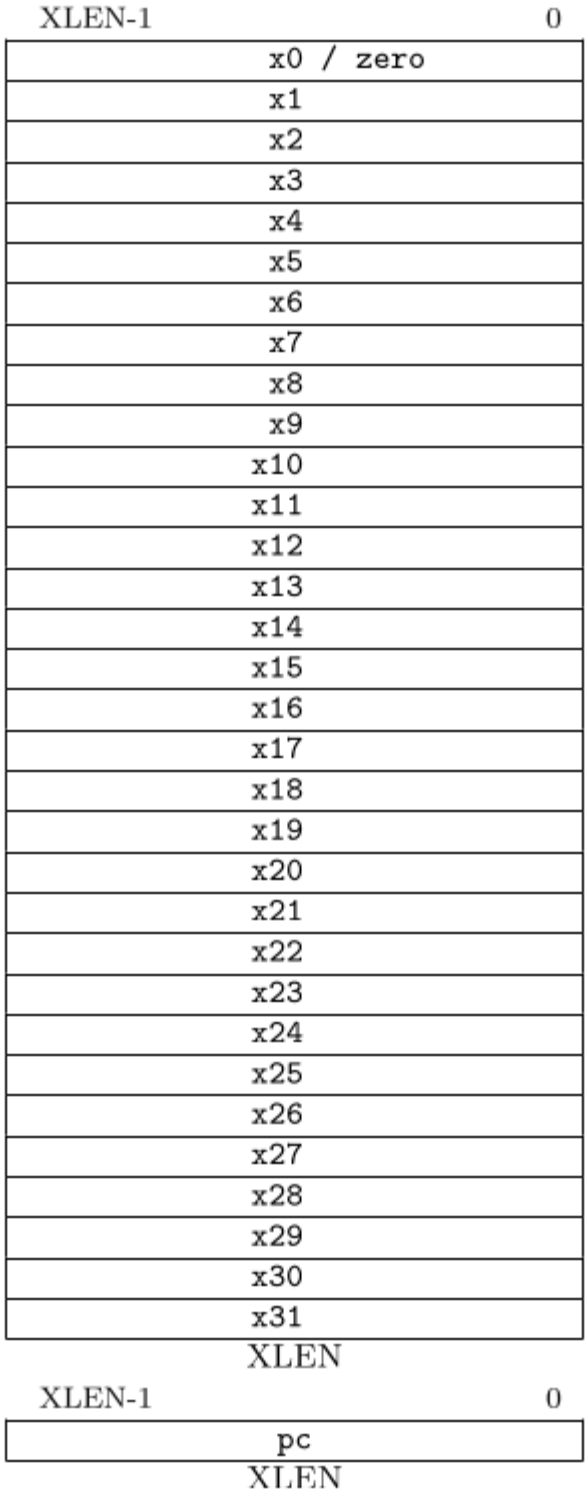

Табл 1. Схема устройства регистров в системе RISC-V

Регистр	ABI название	Описание
x0	zero	Hard-wired ноль
x1	ra	Адрес возврата
x2	sp	Указатель на стеке
x3	gp	Глобальный указатель
x4	tp	Указатель потока
x5	t0	Временный link регистр
x6-x7	t1-2	Временные регистры
x8	s0	Saved регистр
x9	s1	Saved регистр
x10-11	a0-1	Аргументы функций\возвращаемые значения
x12-17	a2-7	Аргументы функций
x18-27	s2-11	Saved регистры
x28-31	t3-6	Временные регистры

Табл 2. Роль регистров

Помимо регистров, RISC-V использует сущность immediate (далее, во всем тексте будет использоваться слово имм) - данные, зашитые в команду. Иммы могут интерпретироваться, в зависимости от команды, как за смещение адреса, так и буквально за константу, или как-нибудь еще.

Так как некоторые команды используют разные данные: есть те, что работают с тремя регистрами (add или mull, например), а есть те, что используют два регистра и имм (addi), или те, что используют один регистр (jal), то RISC-V делит команды на 6 типов: R, I, S, B, U, J. Рассмотрим их поподробнее:

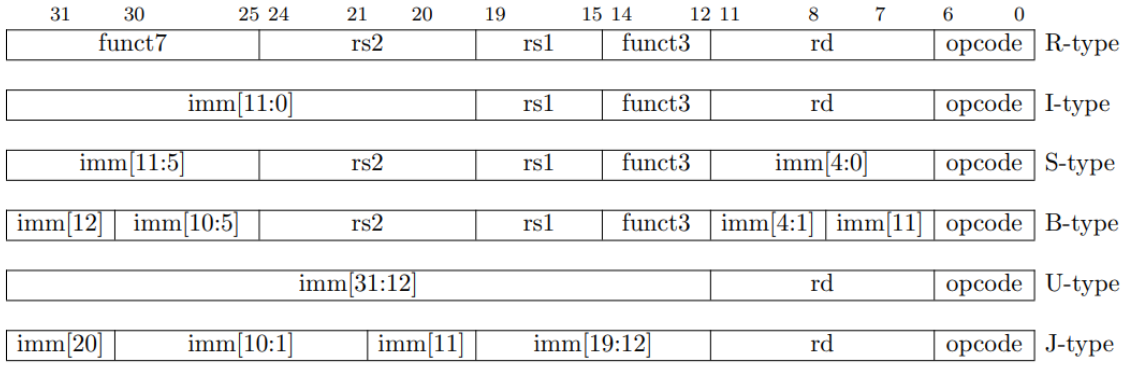

Рис. 3. Типы команд

У всех из них есть opcode - младшие семь бит, отвечающие за идентификацию команды. Именно opcode позволяет однозначно понять, какому типу принадлежит команда (необязательно саму команду) и какие данные как нужно интерпретировать.

- R-Type команды определяются как Register-Register, то есть все взаимодействие происходит только среди регистров
- I-Type команды определяются как Register-Immediate, то есть взаимодействие происходит среди среди регистров и имма (который, как было сказано ранее, имеет свой смысл в отдельно взятой команде).
- S-Type команды определяются как команды, которые сохраняют данные из памяти в регистр.
- B-Type команды определяются как условные, которые, в зависимости от значений rs1, rs2 делают различные прыжки (goto)
- U-Type - отдельная формат для команд Upper-Immediate, которые работают со старшими битами имма
- J-Type - отдельный формат для прыжков (не условных), где имм кодирует смещение с текущего адреса, умноженного на два. На самом деле, под данный тип попадает только одна команда - JAL.

Теперь разберем непосредственно наборы команд, которые требуются по заданию: RV32I (базовый набор) и RV32M:

RV32I

- *Целочисленные вычислительные инструкции.* Команды данного типа либо R-Type, либо I-Type (чаще всего, у каждой команды R-Type есть аналог I-Type и наоборот). В обоих случаях, результат операции записывается в регистр rd. Никакие из данных команд не вызывают

исключений. Если в команде последняя буква I, то это I-Type команда, иначе R-Type.
    - Команды ADD, ADDI, SUB, SUBI выполняют сложение или вычитание из двух поданных операндов.
    - Команды SLT, SLTU, SLTI, SLTIU кладут в rd единицу, если операнд rs1 меньше второго, иначе ставит ноль
    - Команды AND, XOR, OR, ANDI, XORI, ORI выполняют побитовое “И”, “искл. ИЛИ”, “ИЛИ”
    - Команды SLLI, SRLI, SRAI (не имеют R-Type аналога) выполняют логический влево, логический вправо, арифметический вправо сдвиги соответственно. Важно, что сдвиг выполняется на значение, указанное в пяти младших битах имма.
    - Команда LUI (U-Type) строит из имма константу (заполняя при этом младшие 12 бит нулями) и записывает результат в rd
    - Команда AUIPC (U-Type) также строит константу из имма (аналогично LUI), добавляет эту константу к значению регистру pc. Затем, кладет новое значение pc в rd.
- *Прыжки.* Бывают два типа прыжков: условные и безусловные.
    - Безусловные прыжки.
        - Команда JAL (J-Type) прыгает от текущего адреса на смещение, указанное в имме, умноженное на два. Помимо этого, записывает новый адрес в регистр pc.
        - Команд JALR (I-Type) делает прыжок от значения, записанного в регистре rs1, на длину знакового имма. Полученный адрес записывается в регистр pc.
    - Условные прыжки (B-Type).
        - Команды BEQ, BNQ делают прыжок на значение имма, если rs1 = rs2 или rs1 != rs2 соответственно.
        - Команды BLT, BLTU, BGE, BGEU делают прыжок на значение имма, если rs1 < rs2 или rs1 >= rs2 соответственно (U версии считают имм беззнаковым, другие - знаковым).
- *Загрузка (load) и выгрузка (store).* RV32I - load-store архитектура, где load и store инструкции имеют право обращаться к памяти. RV32I обеспечивает 32-битным пространством адресов. Замечу, что загрузка в регистр x0, согласно конвенции, бросает исключение.
    - Команды LB, LH, LW, LBU, LHU, LWU (I-type) читают из памяти байт, два байта, 4 байта соответственно и пишут в

регистр rd. LWU, LHU, LBU есть безннаковые версии данных команд. Сам знак определяется старшим считанным битом.
    - Команды SB, SH, SW (S-type) записывают младшие 8, 16, 32 бита из rs2 в память.
- *Memory Ordering Instructions.* Состоит из единственной команды FENCE, которая отвечает за управление устройствами ввода\вывода и доступ к памяти.
- *Environment Call and Breakpoints.* Состоит из двух инструкций:
    - Команда ECALL создает запрос к среде выполнения
    - Команда EBREAK возвращает контроль к среде отладки (debugging)

RV32I Base Instruction Set

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

Табл. 4 RV32I команды

RV32M

Данное дополнение используется для целочисленного умножения и деления. Все команды, перечисленные здесь, I-type.

- *Операции умножения.*
  - MUL производит умножение двух чисел rs1 и rs2 и кладет младшие 32 бита в регистр rd
  - MULH, MULHS, MULHSU производят умножение, но возвращают старшие 32 бита (которые не уместились при MUL). MULH представляет числа как знаковые, MULHU – как беззнаковые, а MULHSU представляет rs1 как знаковый, а rs2 как беззнаковый.
- *Операции деления.*
  - DIV, DIVU производят деление числа rs1 на rs2, округляя до нуля в случае дробного результата (знаковое и беззнаковое соответственно). В случае деления на ноль получается число, все биты которого - единички. Если же происходит деление, вызывающее переполнение (например, число, состоящее из всех единичек, поделили на -1), получится rs1.
  - REM, REMU находят остаток от деления числа rs1 на rs2 (знаковое и беззнаковое деление соответственно). В случае деления на ноль результат - само число rs1. Если же происходит деление, вызывающее переполнение, получится ноль.

Выше были перечислены наборы, дизассемблер которых будет реализован. Среди других расширений стоит отметить следующие:

- RV32E – другой базовый набор, использующий 16 регистров вместо 32.
- RV64I, RV128I - расширения, позволяющий работать с 64 и 128 битными система
- RV32A – расширение для работы с атомарными инструкциями
- ‘F’, ‘D’, ‘Q’ расширения призваны работать с числами с плавающей точкой (float, double и quad точностью соответственно)
- ‘C’ расширение использует укороченную версию команд - RVC

Оставшиеся стандартные расширения до сих пор не реализованы в должном объёме, чтобы их рассмотреть.

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

Табл. 5 RV32M команды

Описание структуры файла ELF

ELF (Executable and Linkable Format) - распространенный формат файлов для исполняемых файлов, объектного кода, общих библиотек и прочего. По дизайну, ELF формат расширяемый и кросс-платформенный, поддерживает различный порядок данных (endianness).

Любой ELF файл содержит заголовок. На 32-битных системах он состоит из 52 байт, а на 64-битных - 64.

Смещение		**Размер (байт)**		**Поле**	**Цель**
32 бит	**64 бит**	**32 бит**	**64 бит**		
0x00		4		e_ident[EI_MAG0] до e_ident[EI_MAG3]	Магическая константа 7f 45 4c 46, отвечающая за то, что перед нами ELF файл
0x04		1		e_ident[EI_CLASS]	Битность системы
0x05		1		e_ident[EI_DATA]	Порядок байт (endianness)
0x06		1		e_ident[EI_VERSION]	Версия ELF (на данный момент 1)
0x07		1		e_ident[EI_OSABI]	Определяет целевую систему (должно быть 0 для System V)
0x08		1		e_ident[EI_ABIVERSION]	Версия ABI
0x10		2		e_type	Тип объектного файла
0x12		2		e_machine	ISA

Смещение		**Размер (байт)**		**Поле**	**Цель**
32 бит	**64 бит**	**32 бит**	**64 бит**		
0x14		4		e_version	Версия ELF
0x18		4	8	e_entry	Точка старта программы
0x1C	0x20	4	8	e_phoff	Указывает на начало таблицы заголовков программ
0x20	0x28	4	8	e_shoff	Указывает на начало таблицы заголовков секций
0x24	0x30	4		e_flags	Флаги
0x28	0x34	2		e_ehsize	Размер ELF заголовка
0x2A	0x36	2		e_phentsize	Размер заголовка в таблице заголовков программ
0x2C	0x38	2		e_phnum	Количество заголовков в таблице заголовков программ
0x2E	0x3A	2		e_shentsize	Размер секции в таблице заголовков секций
0x30	0x3C	2		e_shnum	Количество секций в таблице заголовков секций
0x32	0x3E	2		e_shstrndx	Индекс секции в таблице заголовков секций, которая хранит названия для таблицы заголовков секций

Табл. 6 Устройство заголовка ELF файла

Чтобы проверить, что перед нами файл, удовлетворяющий условию, необходимо сверить, что следующие константы были равны:

- EI_MAG = 0x7f 0x45 0x4c 0x46 – проверка на то, что перед нами ELF
- EI_CLASS = 1 – 32-битная система
- EI_DATA = 1 – Little-Endian
- EI_VERSION = 1
- EI_OSABI = 0 – System V
- E_TYPE = 2 – перед нами Executable файл

- E_MACHINE = 0xF3 – RISC-V
- E_VERSION = 1

После того, как мы убедились на валидность файла, нужно достать таблицу заголовков секции. Нам потребуется E_SHOFF, E_SHENTSIZE, E_SHNUM для парсинга секций, а также E_SHSTRNDX для парсинга названия секций (эта секция называется ‘.shstrtab’).

Рассмотрим поподробнее таблицу заголовков секций:

Смещение	Размер (байт)	Поле	Цель
0x00	4	sh_name	Смещение строки в секции .shstrtab, которая отвечает за название секции
0x04	4	sh_type	Тип секции
0x08	4	sh_flags	Флаги секций
0x0C	4	sh_addr	Виртуальный адрес секции в памяти
0x10	4	sh_offset	Смещение секции в текущем файле
0x14	4	sh_size	Размер секции
0x18	4	sh_link	Содержит дополнительную информацию о секции (смысл разнится от секции к секции)
0x1C	4	sh_info	
0x20	4	sh_addralign	Содержит выравнивание секции
0x24	4	sh_entsize	Размер отдельного вхождения в секции (в байт)

Табл 7. Таблица заголовков секций

В секции .shstrtab строки хранятся в виде подряд записанных строк, где конец строки - символ-терминатор ‘\0’.

Рассмотрим секцию .symtab.

Смещение	Размер (байт)	Поле	Цель
0x00	4	st_name	Смещение строки в .strtab (секция), которая отвечает за название очередного символа
0x04	4	st_value	Значение символа
0x08	4	st_size	Размер символа
0x12	1	st_info	Тип и бинд символа
0x13	1	st_other	Видимость символа
0x14	2	st_shndx	Индекс секции, которому принадлежит символ

Табл 8. Symbol Table

Закономерный вопрос, который может возникнуть: откуда взять .strtab? Она хранится в заголовке секции в поле sh_link - индекс .strtab в таблице заголовков секций.

Секция .text же состоит из подряд записанных команд RISC-V. Количество и размер секции также можно достать из заголовка секции.

Описание работы написанного кода

Для взаимодействия с файлом использовался вспомогательный класс BinaryFile, позволяющий доставать отдельно взятые байты по адресу и\или приводить их к типу int.

Программа вначале полностью считывает файл, после чего проверяет заголовек ELF на соответствие условию – этим занимается класс ELFHeader. Помимо этого, ELFHeader умеет доставать информацию о таблице заголовков секций (это getSectionHeaderOffset, getSectionHeaderEntrySize и getSectionHeaderEntriesCount), и индекс секции, содержащей названия секций: getShStrNdx.

Класс ELFSectionHeader занимается поиском нужной секции либо по индексу, либо по названию (если программа не смогла найти, то бросает ошибку). Класс, который ELFSectionHeader возвращает -

ELFSectionHeaderEntry умеет доставать все необходимые данные из секции по адресу.

Класс ELFSymbolTable занимается парсингом таблицы симболов. Его функция getMap(BinaryFile) пробегается по таблице, записывая в контейнер map симболы, являющиеся функциями (эти симболы будут использованы в качестве меток при парсинге секции .text).

Функция output(BinaryFile, PrintWriter) пишет в поток out распаршенную таблицу symtab. Происходит это следующим образом: высчитывается адрес симбола, и затем этот адрес пропускается через разные функции, которые достают о нем информацию (getValue, getSize, getType и т.п.). Каждая из этих функций устроена одинаково: они достают необходимый байт\байты, пропускают их через switch-case конструкцию и возвращают строку.

Касательно вывода, если текущий симбол - секция, то её название программа ищет не в strtab, а в .shstr (берется индекс секции, которой симбол принадлежит, т.е. индекс этой секции в таблице заголовков секций, и затем выводится название этой секции).

Класс ELFText содержит только одну функцию - вывод “.text”. Он ищет секцию под названием “.text”, достает метки из ELFSymbolTable (getMap). Прежде, чем нормально выводить команды, класс пробегается по всем командам, и если какие-то из них ссылаются на адреса, у которых нет метки, то он добавляет нужный адрес в map.

После того, как все метки были добавлены, класс бежит по командам, проверяя, есть ли на текущий адрес метка (если есть, то выводит, что это метка).

Парсингом непосредственно команд занимается класс RISCVCOMMAND, который устроен, грубо говоря, как один большой switch-case.

Программа запускается через консоль следующим образом:

java rv3 <input_file> <output_file>

где <input_file> - название входного файла, <output_file> - выходного.

Результат работы написанной программы на приложенном к заданию файле

```
.text
00010074   <main>:
   10074:    ff010113        addi      sp, sp, -16
   10078:    00112623          sw      ra, 12(sp)
   1007c:    030000ef         jal      ra, 100ac <mmul>
   10080:    00c12083          lw      ra, 12(sp)
   10084:    00000513        addi      a0, zero, 0
   10088:    01010113        addi      sp, sp, 16
   1008c:    00008067        jalr      zero, 0(ra)
   10090:    00000013        addi      zero, zero, 0
   10094:    00100137         lui      sp, 256
   10098:    fddff0ef         jal      ra, 10074 <main>
   1009c:    00050593        addi      a1, a0, 0
   100a0:    00a00893        addi      a7, zero, 10
   100a4:    0ff0000f       fence
   100a8:    00000073       ecall
000100ac   <mmul>:
   100ac:    00011f37         lui      t5, 17
   100b0:    124f0513        addi      a0, t5, 292
   100b4:    65450513        addi      a0, a0, 1620
   100b8:    124f0f13        addi      t5, t5, 292
   100bc:    e4018293        addi      t0, gp, -448
   100c0:    fd018f93        addi      t6, gp, -48
   100c4:    02800e93        addi      t4, zero, 40
000100c8   <L2>:
   100c8:    fec50e13        addi      t3, a0, -20
   100cc:    000f0313        addi      t1, t5, 0
   100d0:    000f8893        addi      a7, t6, 0
   100d4:    00000813        addi      a6, zero, 0
000100d8   <L1>:
   100d8:    00088693        addi      a3, a7, 0
   100dc:    000e0793        addi      a5, t3, 0
   100e0:    00000613        addi      a2, zero, 0
000100e4   <L0>:
   100e4:    00078703          lb      a4, 0(a5)
```

```
   100e8:    00069583          lh      a1, 0(a3)
   100ec:    00178793        addi      a5, a5, 1
   100f0:    02868693        addi      a3, a3, 40
   100f4:    02b70733         mul      a4, a4, a1
   100f8:    00e60633         add      a2, a2, a4
   100fc:    fea794e3         bne      a5, a0, 100e4 <L0>
   10100:    00c32023          sw      a2, 0(t1)
   10104:    00280813        addi      a6, a6, 2
   10108:    00430313        addi      t1, t1, 4
   1010c:    00288893        addi      a7, a7, 2
   10110:    fdd814e3         bne      a6, t4, 100d8 <L1>
   10114:    050f0f13        addi      t5, t5, 80
   10118:    01478513        addi      a0, a5, 20
   1011c:    fa5f16e3         bne      t5, t0, 100c8 <L2>
   10120:    00008067        jalr      zero, 0(ra)

.symtab
Symbol Value                Size Type  Bind    Vis             Index Name
[   0] 0x0                     0 NOTYPE   LOCAL    DEFAULT   UNDEF
[   1] 0x10074                 0 SECTION  LOCAL    DEFAULT       1 .text
[   2] 0x11124                 0 SECTION  LOCAL    DEFAULT       2 .bss
[   3] 0x0                     0 SECTION  LOCAL    DEFAULT       3 .comment
[   4] 0x0                     0 SECTION  LOCAL    DEFAULT       4 .riscv.attributes
[   5] 0x0                     0 FILE     LOCAL    DEFAULT     ABS test.c
[   6] 0x11924                 0 NOTYPE   GLOBAL   DEFAULT     ABS __global_pointer$
[   7] 0x118F4               800 OBJECT   GLOBAL   DEFAULT       2 b
[   8] 0x11124                 0 NOTYPE   GLOBAL   DEFAULT       1 __SDATA_BEGIN__
[   9] 0x100AC               120 FUNC     GLOBAL   DEFAULT       1 mmul
[  10] 0x0                     0 NOTYPE   GLOBAL   DEFAULT   UNDEF _start
[  11] 0x11124              1600 OBJECT   GLOBAL   DEFAULT       2 c
[  12] 0x11C14                 0 NOTYPE   GLOBAL   DEFAULT       2 __BSS_END__
[  13] 0x11124                 0 NOTYPE   GLOBAL   DEFAULT       2 __bss_start
[  14] 0x10074                28 FUNC     GLOBAL   DEFAULT       1 main
[  15] 0x11124                 0 NOTYPE   GLOBAL   DEFAULT       1 __DATA_BEGIN__
[  16] 0x11124                 0 NOTYPE   GLOBAL   DEFAULT       1 _edata
```

```
[  17] 0x11C14               0 NOTYPE   GLOBAL   DEFAULT       2 _end
[  18] 0x11764             400 OBJECT   GLOBAL   DEFAULT       2 a
```

Список источников

1. RISC-V: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
2. Wikipedia. Executable and Linkable Format: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
3. Спецификация ELF: https://refspecs.linuxbase.org/elf/gabi4+/

Листинг кода

base/myUtil.java

```
package base;


public class MyUtil {
    private MyUtil() {}

    public static int BytesToInt(byte[] data) {
        int result = 0;
        for (int i = data.length - 1; i >= 0; i--) {
            result = (result << 8) + ((256 + data[i]) % 256);
        }
        return result;
    }

    public static byte[] IntToBytes(int number, int size) {
        byte[] res = new byte[size];
        for (int i = 0; i < number; i++) {
            res[i] = (byte)(number % 256);
            number /= 256;
        }
        return res;
    }
}
```

RISCVDisassembler/BinaryFile.java

```
package RISCVDisassembler;

import java.util.Arrays;

public class BinaryFile {
    private byte[] data;

    public BinaryFile(byte[] data) {
        this.data = data;
    }

    public byte getByte(int idx) {
        return data[idx];
    }

    public byte[] getByteArray(int idx, int len) {
        return Arrays.copyOfRange(data, idx, idx + len);
    }

    public int getValue(int idx, int len) {
        assert len <= 4: "len must be <= 4 cause int is 32bit";
        return base.MyUtil.BytesToInt(getByteArray(idx, len));
    }

}
```

RISCVDiassembler/ELFHeader.java

```
package RISCVDisassembler;

import java.util.Arrays;

public class ELFHeader {
    public static boolean checkForTask(BinaryFile file) {
        return Arrays.equals(
                    file.getByteArray(0, 4),
```

```
                        new byte[]{0x7f, 0x45, 0x4c, 0x46})  && // checking for
magic
                file.getValue(4, 1) == 1            && // 32-bit system
                file.getValue(5, 1) == 1            && // little endianness
                file.getValue(7, 1) == 0 && // System-V
                file.getValue(6, 1) == 1 && // ELF Version
                file.getValue(0x10, 2) == 2 && // executable
                file.getValue(0x14, 4) == 1 && // ELF Version
                Arrays.equals(
                        file.getByteArray(0x12, 2),    // RISC-V
                        new byte[]{(byte) 0xf3, 0x00});
        }

        public static int getShStrNdx(BinaryFile file) {
            return file.getValue(0x32, 2);
        }

        public static int getSectionHeaderOffset(BinaryFile file) {
            return file.getValue(0x20, 4);
        }

        public static int getSectionHeaderEntrySize(BinaryFile file) {
            return file.getValue(0x2E, 2);
        }

        public static int getSectionHeaderEntriesCount(BinaryFile file) {
            return file.getValue(0x30, 2);
        }
    }
```

RISCVDissasembler/ELFSectionHeader.java

```
    package RISCVDisassembler;

    public class ELFSectionHeader {
         public static ELFSectionHeaderEntry getSection(BinaryFile file, String
name) {
              for (int i = 0; i < ELFHeader.getSectionHeaderEntriesCount(file);
i++) {
```

```
                int address = ELFHeader.getSectionHeaderOffset(file) +
                                i * ELFHeader.getSectionHeaderEntrySize(file);
                                    ELFSectionHeaderEntry curSection = new
ELFSectionHeaderEntry(address);
                if (curSection.getName(file).equals(name)) {
                    return curSection;
                }
            }

            throw new RuntimeException("Section " + name + " wasn't found");
        }

          public static ELFSectionHeaderEntry getSection(BinaryFile file, int
ndx) {
            if (ELFHeader.getSectionHeaderEntriesCount(file) < ndx) {
                 throw new RuntimeException(String.format("Given section's index
(%i) is bound out: %i", ndx,
                    ELFHeader.getSectionHeaderEntriesCount(file)));
            } else {
                                                                return     new
ELFSectionHeaderEntry(ELFHeader.getSectionHeaderOffset(file) +
                                                                        ndx  *
ELFHeader.getSectionHeaderEntrySize(file));
            }
        }
    }

    class ELFSectionHeaderEntry {
        int address;

        public ELFSectionHeaderEntry(int address) {
            this.address = address;
        }

        public int getOffset(BinaryFile file) {
            return file.getValue(address + 0x10, 4);
        }
```

```
        public String getName(BinaryFile file) {
                                        ELFSectionHeaderEntry    ShStr    =    new
ELFSectionHeaderEntry(ELFHeader.getSectionHeaderOffset(file)
                                    +  ELFHeader.getSectionHeaderEntrySize(file)  *
ELFHeader.getShStrNdx(file));
            int start = ShStr.getOffset(file) +      // space of names
                 file.getValue(address, 4);      // sh_name - offset in space of
names
            int end = start;
            while ((char) file.getByte(end) != '\0') { ++end; }
            return new String(file.getByteArray(start, end), 0, end - start);
        }

        public int getLink(BinaryFile file) {
            return file.getValue(address + 0x18, 4);
        }

        public int getInfo(BinaryFile file) {
            return file.getValue(address + 0x1C, 4);
        }

        public int getEntrySize(BinaryFile file) {
            return file.getValue(address + 0x24, 4);
        }

        public int getSize(BinaryFile file) {
            return file.getValue(address + 0x14, 4);
        }

        public int getAddress(BinaryFile file) {
            return file.getValue(address + 0xc, 4);
        }
    }
```

RISCVDissasembler/ELFSymbolTable.java

```
package RISCVDisassembler;
```

```
import java.io.PrintWriter;

import java.util.HashMap;

import java.util.Map;


public class ELFSymbolTable {

    public static void output(BinaryFile file, PrintWriter out) {

                                ELFSectionHeaderEntry   symbolTableAsEntry   =
ELFSectionHeader.getSection(file, ".symtab");

                                     ELFSectionHeaderEntry    stringTable    =
ELFSectionHeader.getSection(file,          ELFSymbolTable.getStringTableNdx(file,
symbolTableAsEntry));




        out.format(".symtab\n");

        out.format("Symbol Value               Size Type  Bind         Vis
      Index Name\n");


                  for  (int  i  =  0;  i  <  symbolTableAsEntry.getSize(file)  /
symbolTableAsEntry.getEntrySize(file); i++) {

                     int  address  =  symbolTableAsEntry.getOffset(file)  +  i  *
symbolTableAsEntry.getEntrySize(file);

              out.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s\n",

                  i,

                  ELFSymbolTableEntry.getValue(file, address),

                  ELFSymbolTableEntry.getSize(file, address),

                  ELFSymbolTableEntry.getType(file, address),

                  ELFSymbolTableEntry.getBind(file, address),

                  ELFSymbolTableEntry.getVisibility(file, address),

                          ELFSymbolTableEntry.getSectionObjectBelongsTo(file,
address),

                  ELFSymbolTableEntry.getName(file, address, stringTable)

              );

        }

      }


                public    static    int    getStringTableNdx(BinaryFile    file,
ELFSectionHeaderEntry symbolTableAsEntry) {

        return symbolTableAsEntry.getLink(file);

      }
```

```
          public static Map<Integer, String> getMap(BinaryFile file) {

                                    ELFSectionHeaderEntry    symbolTableAsEntry    =
ELFSectionHeader.getSection(file, ".symtab");

                                          ELFSectionHeaderEntry    stringTable    =
ELFSectionHeader.getSection(file,           ELFSymbolTable.getStringTableNdx(file,
symbolTableAsEntry));


              Map<Integer, String> result = new HashMap<>();

                     for  (int  i  =  0;  i  <  symbolTableAsEntry.getSize(file)  /
symbolTableAsEntry.getEntrySize(file); i++) {

                        int  address  =  symbolTableAsEntry.getOffset(file)  +  i  *
symbolTableAsEntry.getEntrySize(file);

                     if (ELFSymbolTableEntry.getTypeType(file, address) == 2) //
Function

                  result.put(

                              ELFSymbolTableEntry.getValue(file, address),

                                       ELFSymbolTableEntry.getName(file, address,
stringTable)

                              );

              }


              return result;

          }

      }


      class ELFSymbolTableEntry {

                public  static  String  getName(BinaryFile  file,  int  address,
ELFSectionHeaderEntry stringTable) {

                 if (getTypeType(file, address) == 0x03) {            // if it's
section => give its own name

                   return getSection(file, address).getName(file);

              } else {

                  int start = stringTable.getOffset(file) +  // space of names

                        file.getValue(address, 4);         // sh_name - offset in
space of names

                  int end = start;

                  while ((char) file.getByte(end) != '\0') { ++end; }

                       return  new  String(file.getByteArray(start,  end),  0,  end  -
start);
```

```
    }
}

public static int getValue(BinaryFile file, int address) {
    return file.getValue(address + 4, 4);
}

public static int getSize(BinaryFile file, int address) {
    return file.getValue(address + 8, 4);
}

public static int getBindType(BinaryFile file, int address) {
    return file.getValue(address + 12, 1) >> 4;
}

public static String getBind(BinaryFile file, int address) {
    return switch (getBindType(file, address)) {
        case 0 -> "LOCAL";
        case 1 -> "GLOBAL";
        case 2 -> "WEAK";
        case 10 -> "LOOS";
        case 12 -> "HIOS";
        case 13 -> "LOPROC";
        case 15 -> "HIPROC";
        default -> "ERROR";
    };
}

public static int getTypeType(BinaryFile file, int address) {
    return file.getValue(address + 12, 1) & (0xf);
}

public static String getType(BinaryFile file, int address) {
    return switch (getTypeType(file, address)) {
        case 0 -> "NOTYPE";
```

```
                case 1 -> "OBJECT";
                case 2 -> "FUNC";
                case 3 -> "SECTION";
                case 4 -> "FILE";
                        default -> String.format("%x ERROR", getTypeType(file,
address));
            };
        }

        public static int getVisibilityType(BinaryFile file, int address) {
            return file.getValue(address + 13, 1);
        }

        public static String getVisibility(BinaryFile file, int address) {
            return switch(getVisibilityType(file, address)) {
                case 0 -> "DEFAULT";
                case 1 -> "INTERNAL";
                case 2 -> "HIDDEN";
                case 3 -> "PROTECTED";
                default -> "ERROR";
            };
        }

          public static ELFSectionHeaderEntry getSection(BinaryFile file, int
address) {
              return ELFSectionHeader.getSection(file, file.getValue(address +
14, 2));
        }

          public static String getSectionObjectBelongsTo(BinaryFile file, int
address) {
            int shndx = file.getValue(address + 14, 2);
            return switch (shndx) {
                case 0 -> String.valueOf("UNDEF");
                case 0xfff1 -> String.valueOf("ABS");
                default -> String.valueOf(shndx);
            };
```

```
            }

        }
```

RISCVDissasembly/ELFText.java

```
package RISCVDisassembler;

import java.io.PrintWriter;
import java.util.Map;

public class ELFText {
    public static void output(BinaryFile file, PrintWriter out) {
                ELFSectionHeaderEntry  dotText  =  ELFSectionHeader.getSection(file,
".text");
        Map<Integer, String> symtab = ELFSymbolTable.getMap(file);
        int startToRead = dotText.getOffset(file),
            endToRead   = startToRead + dotText.getSize(file),
            countOfCMD  = (endToRead - startToRead) / 4;

        for (int i = 0; i < countOfCMD; i++) {
            int curCMD = dotText.getOffset(file) + i * 4;
            int curAddress = dotText.getAddress(file) + i * 4;
                                    new   RISCVCommand(file.getValue(curCMD,   4),
curAddress).updateLabels(symtab);
        }

        out.println(".text");

        for (int i = 0; i < countOfCMD; i++) {
            int curCMD = dotText.getOffset(file) + i * 4;
            int curAddress = dotText.getAddress(file) + i * 4;
            if (symtab.containsKey(curAddress)) {
                out.format("%08x   <%s>:\n", curAddress, symtab.get(curAddress));
            }
                   RISCVCommand  cmd  =  new  RISCVCommand(file.getValue(curCMD,  4),
curAddress);
            out.print(cmd.disAssembly(symtab));
        }
```

```
    }
}
```

RISCVDisassembler/RISCVCommand.java

```
package RISCVDisassembler;

import java.util.Map;

public class RISCVCommand {
    private static final String UNKNOWN = "UNKNOWN COMMAND";

    private int data;
    private int address;

    public RISCVCommand(byte[] data, int address) {
        assert data.length == 4 : "bad command size";
        this.data = base.MyUtil.BytesToInt(data);
        this.address = address;
    }

    public RISCVCommand(int data, int address) {
        this.data = data;
        this.address = address;
    }

    public String disAssembly(Map<Integer, String> symtab) {
        int opcode = getOpcode();
        return switch (opcode) {
            case 0b0110111 -> parseLUI();
            case 0b0010111 -> parseAUIPC();
            case 0b1101111 -> parseJAL(symtab);
            case 0b1100111 -> parseJALR();
            case 0b1100011 -> parseBEQorBNEorBLTorBGEorBLTUorBGEU(symtab);
            case 0b0000011 -> parseLBorLHorLWorLBUorLHU();
            case 0b0100011 -> parseSBorSHorSW();
                                                                case     0b0010011     ->
parseADDIorSLTIorSLTIUorXORIorORIorANDIorSLLIorSRLIorSRAI();
```

```
                                                                    case     0b0110011     ->
parseADDorSUBorSLLorSLTorSLTUorXORorSRLorSRAorORorANDorMULorMULHorMULHSUorMULHUorD
IVorDIVUorREMorREMU();
            case 0b0001111 -> parseFENCE();
            case 0b1110011 -> parseECALLorEBREAK();
                 default  ->  threeArgumentsInstruction(UNKNOWN,  UNKNOWN,  UNKNOWN,
UNKNOWN);
        };
    }



    static int idx = 0;


    public void updateLabels(Map<Integer, String> symtab) {
        int dest = switch (getOpcode()) {
            case 0b1101111 -> address + immJ(); // JAL
            case 0b1100011 -> address + immB(); // BEQ BNE BLT BGE BLTU BGEU
            default -> -1;
        };
        if (dest > -1 && !symtab.containsKey(dest)) {
            symtab.put(dest, String.format("L%d", idx));
            idx++;
        }
    }


                                                                private              String
parseADDorSUBorSLLorSLTorSLTUorXORorSRLorSRAorORorANDorMULorMULHorMULHSUorMULHUorD
IVorDIVUorREMorREMU() {
        String nameOfCommand = UNKNOWN;
        switch (getFunct3()) {
        case 0b000 -> {
            if (getFunct7() == 0) {
                nameOfCommand = "add";
            }
            if (getFunct7() == 0b0100000) {
                nameOfCommand = "sub";
            }
```

```
    if (getFunct7() == 0b0000001) {
        nameOfCommand = "mul";
    }
}
case 0b001 -> {
    if (getFunct7() == 0) {
        nameOfCommand = "sll";
    }
    if (getFunct7() == 0b0000001) {
        nameOfCommand = "mulh";
    }
}
case 0b010 -> {
    if (getFunct7() == 0) {
        nameOfCommand = "slt";
    }
    if (getFunct7() == 0b0000001) {
        nameOfCommand = "mulhsu";
    }
}
case 0b011 -> {
    if (getFunct7() == 0) {
        nameOfCommand = "sltu";
    }
    if (getFunct7() == 0b0000001) {
        nameOfCommand = "mulhu";
    }
}
case 0b100 -> {
    if (getFunct7() == 0) {
        nameOfCommand = "xor";
    }
    if (getFunct7() == 0b0000001) {
        nameOfCommand = "div";
    }
```

```
        }
        case 0b101 -> {
            if (getFunct7() == 0) {
                nameOfCommand = "srl";
            }
            if (getFunct7() == 0b0100000) {
                nameOfCommand = "sra";
            }
            if (getFunct7() == 0b0000001) {
                nameOfCommand = "divu";
            }
        }
        case 0b110 -> {
            if (getFunct7() == 0) {
                nameOfCommand = "or";
            }
            if (getFunct7() == 0b0000001) {
                nameOfCommand = "rem";
            }
        }
        case 0b111 -> {
            if (getFunct7() == 0) {
                nameOfCommand = "and";
            }
            if (getFunct7() == 0b0000001) {
                nameOfCommand = "remu";
            }
        }
        }

          return threeArgumentsInstruction(nameOfCommand, getRSRegister(getRD()),
getRSRegister(getRS1()),
                getRSRegister(getRS2()));
    }

    private String parseADDIorSLTIorSLTIUorXORIorORIorANDIorSLLIorSRLIorSRAI() {
```

```
String nameOfCommand = UNKNOWN; int immediate = 0;
switch (getFunct3()) {
    case 0b000 -> {
        nameOfCommand = "addi";
        immediate = immI();
    }
    case 0b001 -> {
        if (getFunct7() == 0) {
            nameOfCommand = "slli";
        }
        immediate = immI() & 0b11111;
    }
    case 0b010 -> {
        nameOfCommand = "slti";
        immediate = immI();
    }
    case 0b011 -> {
        nameOfCommand = "sltiu";
        immediate = immI();
    }
    case 0b100 -> {
        nameOfCommand = "xori";
        immediate = immI();
    }
    case 0b101 -> {
        if (getFunct7() == 0) {
            nameOfCommand = "srli";
        }
        if (getFunct7() == 0b0100000) {
            nameOfCommand = "srai";
        }
        immediate = immI() & 0b11111;
    }
    case 0b110 -> {
        nameOfCommand = "ori";
```

```
                immediate = immI();
            }
            case 0b111 -> {
                nameOfCommand = "andi";
                immediate = immI();
            }
        }

          return threeArgumentsInstruction(nameOfCommand, getRSRegister(getRD()),
getRSRegister(getRS1()), String.valueOf(immediate));
    }

    private String parseLBorLHorLWorLBUorLHU() {
        String nameOfCommand = UNKNOWN;
        switch (getFunct3()) {
            case 0b000 -> {
                nameOfCommand = "lb";
            }
            case 0b001 -> {
                nameOfCommand = "lh";
            }
            case 0b010 -> {
                nameOfCommand = "lw";
            }
            case 0b100 -> {
                nameOfCommand = "lbu";
            }
            case 0b101 -> {
                nameOfCommand = "lhu";
            }
        }

        return LOADorSTOREorJALRInstruction(nameOfCommand, getRSRegister(getRD()),
String.valueOf(immI()), getRSRegister(getRS1()));
    }
```

```
    private String parseLUI() {
                    return  twoArgumentsInstruction("lui",  getRSRegister(getRD()),
String.valueOf(immU()));
    }

    private String parseAUIPC() {
                  return  twoArgumentsInstruction("auipc",  getRSRegister(getRD()),
String.valueOf(immU()));
    }

    private String parseECALLorEBREAK() {
        String name = UNKNOWN;
         if (getRD() == 0 && getFunct3() == 0 && getRS1() == 0 && ((data >>> 20) &
0xfff) == 0) {
            name = "ecall";
        } else if (getRD() == 0 && getFunct3() == 0 && getRS1() == 0 && ((data >>>
20) & 0xfff) == 1) {
            name = "ebreak";
        }
        return noArgumentsInstruction(name);
    }

    private String parseFENCE() {
        String name = UNKNOWN;
        if (getFunct3() == 0) {
            name = "fence";
        }
        return fenceInstruction(name);
    }

    private String parseJALR() {
        String nameOfCommand = UNKNOWN;
        if (getFunct3() == 0) {
            nameOfCommand = "jalr";
        }

        return LOADorSTOREorJALRInstruction(nameOfCommand, getRSRegister(getRD()),
```

```
                String.valueOf(immI()),
                getRSRegister(getRS1())
            );
    }

    private String parseSBorSHorSW() {
        String nameOfCommand = UNKNOWN;
        switch (getFunct3()) {
        case 0b000 -> {
            nameOfCommand = "sb";
        }
        case 0b001 -> {
            nameOfCommand = "sh";
        }
        case 0b010 -> {
            nameOfCommand = "sw";
        }
        }

                                return    LOADorSTOREorJALRInstruction(nameOfCommand,
getRSRegister(getRS2()),
            String.valueOf(immS()),
            getRSRegister(getRS1())
            );
    }

        private  String  parseBEQorBNEorBLTorBGEorBLTUorBGEU(Map<Integer,  String>
symtab) {
        String nameOfCommand = UNKNOWN;
        switch (getFunct3()) {
            case 0b000 -> {
                nameOfCommand = "beq";
            }
            case 0b001 -> {
                nameOfCommand = "bne";
            }
```

```
            case 0b100 -> {
                nameOfCommand = "blt";
            }
            case 0b101 -> {
                nameOfCommand = "bge";
            }
            case 0b110 -> {
                nameOfCommand = "bltu";
            }
            case 0b111 -> {
                nameOfCommand = "bgeu";
            }
        }

        String label = getLabel(immB(), symtab);



          return threeArgumentsInstruction(nameOfCommand, getRSRegister(getRS1()),
getRSRegister(getRS2()), label);
    }

    private String parseJAL(Map<Integer, String> symtab) {
        int offset = immJ();

        String label = getLabel(offset, symtab);

        return twoArgumentsInstruction("jal", getRSRegister(getRD()), label);
    }

    private String getLabel(int offset, Map<Integer, String> symtab) {
        int findingAddress = address + offset;
                            return   String.format("%05x   <%s>",   findingAddress,
symtab.get(findingAddress));
    }

      private String threeArgumentsInstruction(String name, String arg1, String
arg2, String arg3) {
```

```
        if (name.equals(UNKNOWN)) {
            return unknownCommand();
        } else {
               return String.format("   %05x:\t%08x\t%7s\t%s, %s, %s\n", address,
data, name, arg1, arg2, arg3);
        }
    }

    private String unknownCommand() {
        return String.format("   %05x:\t%08x\tunknown command\n", address, data);
    }

     private String twoArgumentsInstruction(String name, String arg1, String arg2)
{
        if (name.equals(UNKNOWN)) {
            return unknownCommand();
        } else {
              return String.format("   %05x:\t%08x\t%7s\t%s, %s\n", address, data,
name, arg1, arg2);
        }
    }

    private String fenceInstruction(String name) {
        if (name.equals(UNKNOWN)) {
            return unknownCommand();
        } else {
            return String.format("   %05x:\t%08x\t%7s\n", address, data, name);
        }
    }

     private String LOADorSTOREorJALRInstruction(String name, String arg1, String
arg2, String arg3) {
        if (name.equals(UNKNOWN)) {
            return unknownCommand();
        } else {
               return String.format("   %05x:\t%08x\t%7s\t%s, %s(%s)\n", address,
data, name, arg1, arg2, arg3);
```

```
    }
}

private String noArgumentsInstruction(String name) {
    if (name.equals(UNKNOWN)) {
        return unknownCommand();
    } else {
        return String.format("   %05x:\t%08x\t%7s\n", address, data, name);
    }
}

private int makeSigned(int number, int signedBit) {
    if ((number & (1 << signedBit)) != 0) {
        number = -((1 << signedBit + 1) - number);
    }
    return number;
}

private int getOpcode() {
    return data & 0b0111_1111;
}

private int getRD() {
    return (data >>> 7) & 0b11111;
}

private int getFunct3() {
    return (data >>> 12) & 0b111;
}

private int getFunct7() {
    return (data >>> 25) & 0b0111_1111;
}

private int getRS1() {
```

```
    return (data >>> 15) & 0b11111;
}

private int getRS2() {
    return (data >>> 20) & 0b11111;
}

private int immI() {
    return makeSigned(data >>> 20, 11);
}

private int immS() {
    return makeSigned(
    (((data >>> 25) & 0b1111111) << 5)
        |
    ((data >>> 7) & 0b11111), 11);
}

private int immB() {
    return makeSigned(
        (((data >>> 8) & 0b1111) << 1) |
        (((data >>> 25) & 0b111111) << 5) |
        (((data >>> 7) & 0b1) << 11) |
        (((data >>> 31) & 0b1) << 12),
        12);
}

private int immU() {
    return makeSigned(
        data >>> 12, 31-12);
}

private int immJ() {
    return makeSigned(
        (((data >>> 21) & (0b1111111111)) << 1) |
```

```
            (((data >>> 20) & (0b1)) << 11) |
            (((data >>> 12) & (0b11111111)) << 12) |
            (((data >>> 31) & (0b1)) << 20), 20);
    }

    private String getRSRegister(int number) {
        return switch (number) {
        case 0 -> "zero";
        case 1 -> "ra";
        case 2 -> "sp";
        case 3 -> "gp";
        case 4 -> "tp";
        case 5 -> "t0";
        case 6 -> "t1";
        case 7 -> "t2";
        case 8 -> "s0";
        case 9 -> "s1";
        case 10, 11, 12, 13, 14, 15, 16, 17 -> {
            yield "a" + number % 10;
        }
        case 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 -> {
            yield "s" + (number - 16);
        }
        case 28, 29, 30, 31 -> {
            yield "t" + (number - 25);
        }
        default -> "ERROR";
        };
    }
}
```

rv3.java

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.PrintWriter;


import RISCVDisassembler.BinaryFile;
import RISCVDisassembler.ELFHeader;
import RISCVDisassembler.ELFSymbolTable;
import RISCVDisassembler.ELFText;



public class rv3 {
    public static void main(String[] args) {
        if (args.length != 2) {
                System.out.println("Argument count must equal to 2. Your: " +
args.length);
            return;
        } else if (args[0] == args[1]) {
               System.out.println("Input and output files are identical. Please,
change output file's name.");
        }
        try (PrintWriter out = new PrintWriter(new FileOutputStream(args[1]))) {
            InputStream stream = new FileInputStream(args[0]);
            BinaryFile file = new BinaryFile(stream.readAllBytes());
            stream.close();
            if (!ELFHeader.checkForTask(file)) {
                    System.err.println("Sorry, input file must be ELF & 32bit &
Little-Endian & RISC-V");
                return;
            }
            ELFText.output(file, out);
            out.println();
            ELFSymbolTable.output(file, out);
        } catch (IOException e) {
            System.err.println("IO Exception: " + e.getMessage());
        } catch (RuntimeException e) {
            System.err.println("Runtime exception: " + e.getMessage());
        }
        return;
    }
```

```
}
```