САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 4

**«ISA. АССЕМБЛЕР, ДИЗАССЕМБЛЕР»**

Выполнил: Ивченков Дмитрий Артемович

Номер ИСУ: 334906

студ. гр. M3134

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**

**Система кодирования команд RISC-V**

RISC-V – это открытая система команд (*ISA*) и процессорная архитектура на основе концепции RISC. Идея RISC заключается в том, что аппаратная реализация редко используемых команд отсутствует, они нуждаются в программной. Такие команды будут исполняться медленнее, с другой стороны, чаще используемые команды получат ускорение.

Чтобы сделать RISC-V широко применимой и удобной, создателями были определены и достигнуты некоторые цели:

1. Разделение на базовую ISA и дополнительные расширения. Базовый набор инструкций относительно компактен, однако он достаточно полон, а расширения повышают производительность вычислительных нагрузок и предоставляют дополнительные возможности.
2. Повышение энергоэффективности и производительности, удешевление разработки. Реализация лишь минимального необходимого набора команд и открытость системы позволяет разрабатывать полноценно работающие системы, что даёт возможность экономить аппаратные ресурсы и снижать затраты на разработку.
3. Поддержка 32-битных и 64-битных адресных пространств при одинаковой кодировке инструкций для работы на различных устройствах. Также существует вариант с 128-битными адресными пространствами.
4. Поддержка как современных стандартов, так и пользовательских расширений ISA.

В архитектуре RISC-V имеется обязательное для реализации небольшое подмножество команд (набор инструкций I – Integer) и несколько стандартных расширений. Существуют три базовых варианта ISA – RV32I, RV64I, RV32E, реализации которых тесно связаны. Первые два различаются в первую очередь шириной регистров и размером адресного пространства. RV32E – это вариация RV32I с меньшим количеством регистров, предназначенная для встраиваемых систем, где важен каждый транзистор. В базовый набор входят инструкции условной и безусловной передачи управления и ветвления, минимальный набор арифметических и битовых операций на регистрах, операций с памятью и небольшое количество служебных инструкций. В работе рассматривается RV32I – базовая 32-битная целочисленная ISA, включающая в себя 47 инструкций.

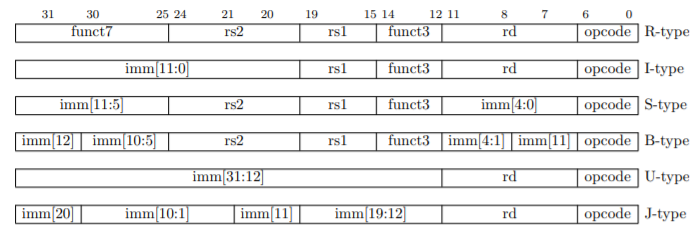
RISC-V является архитектурой типа регистр-регистр (*Reg-Reg*), в которой арифметические инструкции работают только с регистрами и только инструкции load и store передают данные из памяти и в память. В RV32I есть 31 целочисленный регистр общего назначения с именами *x1-x31* размерами 32 бита каждый, специальный регистр *x0* (zero), всегда равный нулю, и единственный дополнительный регистр – программный счётчик (PC), содержащий байтовый адрес текущей инструкции. В ABI (двоичном интерфейсе приложений) регистры имеют названия, представленные в таблице 1.

Таблица № 1 – Регистры в ABI

|  |  |
| --- | --- |
| Регистр | Название в ABI |
| x0 | zero |
| x1 | ra |
| x2 | sp |
| x3 | gp |
| x4 | tp |
| x5 | t0 |
| x6-7 | t1-2 |
| x8 | s0 |
| x9 | s1 |
| x10-11 | a0-1 |
| x12-17 | a2-7 |
| x18-27 | s2-11 |
| x28-31 | t3-6 |

**RV32I**

В базовом ISA есть четыре основных формата инструкций (R, I, S, U), которые определяют, как и на какие смысловые части делится инструкция. Инструкции имеют длину 32 бита и должны храниться в памяти с естественным выравниванием, т.е. их адрес должен быть кратен их размеру, в порядке байтов little-endian, т.е. первый байт в памяти соответствует наименее значащим битам адреса. Команды используют до двух регистровых операндов (source registers), называемых rs1 и rs2, и выдают до одного регистрового результата (destination register), называемого rd. Эти регистры-спецификаторы, если они есть, всегда занимают одинаковую позицию в инструкции. Инструкции могут работать с константными значениями (immediate operand) помечаемыми как imm[x], их значения могут быть расположены в разных частях инструкции. Каждое такое подполе индексируется битовой позицией в самом значении, а не в общей инструкции. Существует ещё два варианта форматов инструкций (B и J), основанных на обработке константных значений. Устройство инструкций вышеперечисленных форматов изображено на рисунке 1.

Рисунок № 1 – Форматы инструкций RV32I

RV32I содержит 21 вычислительную инструкцию, включая арифметические, логические инструкции и инструкции сравнения. Они представлены в таблице 2. Данные команды работают с целочисленными регистрами, некоторые из них принимают константные значения. Вычислительные инструкции работают как со знаковыми, так и беззнаковыми целыми числами. В знаковых целых числах используется дополнение до двух. Все константные операнды расширяются до знакового представления, даже если представляют собой беззнаковые величины.

Таблица № 2 – Вычислительные инструкции RV32I

|  |  |  |
| --- | --- | --- |
| Инструкция | Формат | Действие |
| add rd, rs1, rs2 | R | Сложение регистров |
| sub rd, rs1, rs2 | R | Вычитание регистров |
| sll rd, rs1, rs2 | R | Логический битовый сдвиг регистра влево |
| srl rd, rs1, rs2 | R | Логический битовый сдвиг регистра вправо |
| sra rd, rs1, rs2 | R | Арифметический битовый сдвиг регистра вправо |
| and rd, rs1, rs2 | R | Побитовое И с регистрами |
| or rd, rs1, rs2 | R | Побитовое ИЛИ с регистрами |
| xor rd, rs1, rs2 | R | Побитовое исключающее ИЛИ с регистрами |
| slt rd, rs1, rs2 | R | Установление, если меньше регистра (дополнение до 2) |
| sltu rd, rs1, rs2 | R | Установление, если меньше регистра (без знака) |
| addi rd, rs1, imm[11:0] | I | Прибавление константы |
| slli rd, rs1, shamt[4:0] | I | Логический битовый сдвиг регистра влево на константу |
| srli rd, rs1, shamt[4:0] | I | Логический битовый сдвиг регистра вправо на константу |
| srai rd, rs1, shamt[4:0] | I | Арифметический битовый сдвиг регистра вправо на константу |
| andi rd, rs1, imm[11:0] | I | Побитовое И с константой |
| ori rd, rs1, imm[11:0] | I | Побитовое ИЛИ с константой |
| xori rd, rs1, imm[11:0] | I | Побитовое исключающее ИЛИ с константой |
| slti rd, rs1, imm[11:0] | I | Установление, если меньше константы (дополнение до 2) |
| sltiu rd, rs1, imm[11:0] | I | Установление, если меньше константы (без знака) |
| lui rd, imm[31:12] | U | Установление старших 20 бит регистра равными 20-битному константному значению, установление младших 12 бит равными 0. |
| auipc rd, imm[31:12] | U | Добавление 20-битного константного значения к PC и запись в регистр |

Команды операций над регистрами типа register-register (ADD, SLT, SLTU, AND, OR, XOR, SLL, SRL, SUB, SRA) используют формат R. Команды операций над регистром и константой типа register-immediate (ADDI, SLTI/SLTIU, ANDI, ORI, XORI) имеют формат I. Инструкции битовых сдвигов SLLI, SRLI, SRAI записываются в отдельном варианте формата I. Инструкции LUI и AUIPC используют формат U.

RV32I предоставляет пять инструкций для загрузки (load) значения из памяти в регистр и три для сохранения (store) значения регистра в память. Эти инструкции используют байтовые адреса мест в памяти. Они формируют адрес путём добавления значения rs1 к 12-битному знаковому константному значению. Инструкции доступа к памяти указаны в таблице 3.

Таблица № 3 – Инструкции доступа к памяти RV32I

|  |  |  |
| --- | --- | --- |
| Инструкция | Формат | Действие |
| lb rd, imm[11:0](rs1) | I | Загрузка байта со знаком |
| lbu rd, imm[11:0](rs1) | I | Загрузка байта без знака |
| lh rd, imm[11:0](rs1) | I | Загрузка половины слова со знаком |
| lhu rd, imm[11:0](rs1) | I | Загрузка половины слова без знака |
| lw rd, imm[11:0](rs1) | I | Загрузка слова |
| sb rs2, imm[11:0](rs1) | S | Сохранение байта |
| sh rs2, imm[11:0](rs1) | S | Сохранение половины слова |
| sw rs2, imm[11:0](rs1) | S | Сохранение слова |
| fence pred, succ | I | Упорядочение между обращениями к памяти |
| fence.i | I | Синхронизация потоков инструкций с обращениями к памяти |

Все инструкции загрузки имеют формат I. Команда LW копирует 32-битное слово из памяти в регистр rd. LH и LB загружают 16-разрядные (половина слова) и 8-разрядные (байт) величины соответственно, помещая результат в наименее значащие биты rd и заполняя старшие биты копиями знакового бита. LHU и LBU аналогичны, но заполняют старшие биты нулями. Инструкции сохранения являются инструкциями S-типа. SW копирует 32-битное значение регистра rs2 в память по заданному адресу. Команды SH и SB копируют младшие 16 и 8 битов rs2 в ячейки памяти размером в половину слова и байт соответственно. Структура инструкций загрузки (LOAD) и сохранения (STORE), работающих с регистрами и памятью.

Поток выполнения в RISC-V воспринимает все свои собственные обращения к памяти (загрузки и сохранения) как произошедшие в программном порядке, но в многопоточной среде отсутствует гарантия порядка, в котором один тред воспринимает обращения к памяти другого треда. Такая модель используется в RISC-V и называется ослабленной моделью памяти. Ввиду сложности и высокой стоимости передачи данных между потоками, сильная модель памяти, когда любое изменение в памяти, совершённое в одном потоке, должно быть увидено в другом, ограничивает производительность. В ослабленной модели дорогостоящая операция синхронизации данных может не выполняться, если в данный момент она не нужна. RV32I предоставляет инструкцию FENCE, которая обеспечивает упорядочение между обращениями к памяти. Инструкция имеет два аргумента: набор предшественников (predecessor set) и набор последователей (successor set). Любая комбинация входа (I) и выхода (O) устройства, чтения из памяти (R) и записи в память (W) может быть упорядочена в соответствии с любой другой комбинацией такого же плана. Другими словами, никакое устройство не может наблюдать любые операции из последующего после FENCE набора до любых операций в наборе, предшествующем FENCE. Инструкция FENCE.I используется для синхронизации потоков инструкций с обращениями к памяти. Сохранение в памяти команд гарантированно отражается только после выполнения FENCE.I.

RV32I предоставляет два типа команд передачи управления, представленных в таблице 4: безусловные прыжки и условные ветвления. Инструкция перехода и ссылки JAL (jump and link) имеет формат J и устанавливает программный счётчик pc в указанное место. Он также записывает адрес следующей за собой инструкции в регистр rd. Инструкция косвенного перехода JALR (jump and link register) является инструкцией формата I. Адрес назначения получается добавлением 12-битного знакового константного значения к регистру rs1 и установки младшего бита результата в ноль. Адрес следующей инструкции пишется в регистр rd.

Все инструкции ветвления имеют формат B. Они сравнивают два регистра. BEQ и BNE переходят в ветвь, если rs1 и rs2 равны и не равны соответственно. BLT и BLTU производят ответвление, если rs1 меньше rs2, используя знаковое и беззнаковое сравнения соответственно. BGE и BGEU переходят в ветвь, если rs1 больше или равно rs2, используя знаковое и беззнаковое сравнение соответственно.

Таблица № 4 – Инструкции передачи управления RV32I

|  |  |  |
| --- | --- | --- |
| Инструкция | Формат | Действие |
| beq rs1, rs2, imm[12:1] | B | Переход, если значения равны. |
| bne rs1, rs2, imm[12:1] | B | Переход, если значения не равны |
| blt rs1, rs2, imm[12:1] | B | Переход, если меньше (дополнение до 2) |
| bltu rs1, rs2, imm[12:1] | B | Переход, если меньше (без знака) |
| bge rs1, rs2, imm[12:1] | B | Переход, если больше или равно (дополнение до 2) |
| bgeu rs1, rs2, imm[12:1] | B | Переход, если больше или равно (без знака) |
| jal rd, imm[20:1] | J | Переход по значению |
| jalr rd, rs1, imm[11:0] | I | Переход по значению регистра, сложенному с константой |

Завершают RV32I восемь системных инструкций, перечисленных в таблице 5. Инструкция ECALL (ранее SCALL) используется для вызова вспомогательной среды, обычно являющейся операционной системой. Инструкция EBREAK (ранее SBREAK) используется для вызова отладчика.

Оставшиеся шесть команд реализованы для чтения регистров управления и состояния (CSR), которые предоставляют общие возможности для управления системой.

Таблица № 5 – Системные инструкции RV32I

|  |  |  |
| --- | --- | --- |
| Инструкция | Формат | Действие |
| ecall | I | Вызов системы |
| ebreak | I | Вызов отладчика |
| csrrw rd, csr, rs1 | I | Копирование и перезапись |
| csrrc rd, csr, rs1 | I | Копирование и очищение |
| csrrs rd, csr, rs1 | I | Копирование и установка |
| csrrwi rd, csr, imm[4:0] | I | csrrw с константой |
| csrrci rd, csr, imm[4:0] | I | csrrc с константой |
| csrrsi rd, csr, imm[4:0] | I | csrrs с константой |

В большинстве систем CSR доступны только для привилегированного программного обеспечения, но RV32I имеет ряд 64-битных счётчиков на уровне пользователя, доступных только для чтения. Таблица 6 содержит их описание.

Таблица № 6 – Регистры управления и состояния

|  |  |
| --- | --- |
| Название | Значение |
| cycle | Количество прошедших тактов |
| time | Значение, пропорциональное реальному прошедшему времени |
| instret | Количество выполненных инструкций |
| cycleh | Старшие 32 бита cycle |
| timeh | Старшие 32 бита time |
| instreth | Старшие 32 бита instret |

**RV32M**

Во многих вычислениях часто используются умножение и деление. Даже при небольшом количестве таких операций они могут составлять значительную часть времени выполнения, если их реализовывать в виде подпрограмм.

Следовательно, в большинстве случаев желательна их ускоренная аппаратная реализация. Для этого в RISC-V предусмотрено расширение M, добавляющее инструкции, представленные в таблице 7.

Таблица № 7 – Инструкции расширения RV32M

|  |  |  |
| --- | --- | --- |
| Инструкция | Формат | Действие |
| mul rd, rs1, rs2 | R | Получение младших битов умножения |
| mulh rd, rs1, rs2 | R | Получение старших битов умножения со знаком |
| mulhu rd, rs1, rs2 | R | Получение старших битов умножения без знака |
| mulhsu rd, rs1, rs2 | R | Получение старших битов умножения со смешанными знаками |
| div rd, rs1, rs2 | R | Целочисленное деление со знаком |
| divu rd, rs1, rs2 | R | Целочисленное деление без знака |
| rem rd, rs1, rs2 | R | Взятие остатка от деления со знаком |
| remu rd, rs1, rs2 | R | Взятие остатка от деления без знака |

Появляются четыре инструкции, вычисляющие 32×32-битные произведения. Инструкция MUL возвращает младшие 32 бита произведения. Команды MULH, MULHU, MULHSU возвращают старшие 32 бита произведения, рассматривая множители как знаковые, беззнаковые числа и числа со смешанными знаками соответственно.

Ещё есть четыре инструкции, выполняющие деление 32 бит на 32 бит: DIV и DIVU вычисляют результат знакового и беззнакового деления, а REM и REMU находят знаковый и беззнаковый остаток от деления. Деление на ноль не вызывает исключения. Если языкам программирования требуется подобное поведение, они могут выполнять ветвление после операции деления.

**RVC**

Для уменьшения размера кода существует расширение RVC, добавляющее более короткие инструкции для общих операций. Данное расширение нацелено на использование производительности и энергетических преимуществ кодирования переменной длины. Создание RVC мотивировано четырьмя наблюдениями о составе инструкций многих программ:

1. Небольшое количество кодов составляет большинство инструкций в программах.
2. Многие инструкции имеют мало уникальных операндов.
3. Доступы к регистрам демонстрируют существенную локальность ссылок.
4. Константные значения обычно небольшие.

RVC использует простую схему сжатия, предлагающую 16-битные версии обычных 32-битных инструкций в случаях, если

1. Константное или адресное смещение небольшое;
2. Один из регистров является zero (x0), ABI link register (x1) или ABI stack pointer (x2);
3. Регистр назначения и первый исходный регистр совпадают;
4. Используются 8 самых используемых регистров.

Все инструкции расширения имеют 5-битный код операции и не менее одного операнда. Константные значения длиной 5 или 6 бит. Смещения условных ветвлений длиной 6 бит, смещения безусловных переходов длиной 10 бит. Инструкции, которые ссылаются на регистр, делают это с помощью его полного 5-битного или сокращённого 3-битного спецификатора. ИRVC и им эквивалентные стандартные инструкции представлены в таблице 8.

Таблица № 8 – Инструкции RVC

|  |  |
| --- | --- |
| Инструкция RVC | Стандартная эквивалентая инструкция |
| c.lwsp | Lw rd, offset[7:2] (x2) |
| c.swsp | sw rs2, offset[7:2](x2) |
| c.lw | lw rd ′, offset[6:2](rs1 ′) |
| c.sw | sw rs2 ′, offset[6:2](rs1 ′) |
| c.j | jal x0, offset[11:1] |
| c.jal | jal x1, offset[11:1] |
| c.jr | jalr x0, 0(rs1) |
| c.jalr | jalr x1, 0(rs1) |
| c.beqz | beq rs1 ′, x0, offset[8:1] |
| c.bnez | bne rs1 ′, x0, offset[8:1] |
| c.li | addi rd, x0, imm[5:0]. |
| c.lui | lui rd, nzimm[17:12] |
| c.addi | addi rd, rd, nzimm[5:0] |
| c.addi16sp | addi x2, x2, nzimm[9:4]. |
| c.addi4spn | addi rd ′, x2, nzuimm[9:2]. |
| c.slli | slli rd, rd, shamt[5:0] |
| c.srli | srli rd ′, rd ′, shamt[5:0] |
| c.srai | srai rd ′, rd ′, shamt[5:0] |
| c.andi | andi rd ′, rd ′, imm[5:0]. |
| c.mv | add rd, x0, rs2 |
| c.add | add rd, rd, rs2 |
| c.and | and rd ′, rd ′, rs2 ′ |
| c.or | or rd ′, rd ′, rs2 ′ |
| c.xor | xor rd ′, rd ′, rs2 ′ |
| c.sub | sub rd ′, rd ′, rs2 ′ |
| c.ebreak | ebreak |

**Структура elf-файла**

ELF (Executable and Linkable Format) – формат исполнимых и компонуемых двоичных файлов, использующийся во многих UNIX-подобных и прочих системах. Файл ELF может быть различных типов:

1. Перемещаемый (Relocatable) файл, хранящий код и данные, которые могут быть связаны с другими объектными файлами для создания исполняемого или разделяемого объектного файла.
2. Исполняемый (Executable) файл, содержащий программу, пригодную для исполнения, и позволяющий создать образ процесса.
3. Разделяемый объектный (Shared object) файл, содержащий код и данные, подходящие для его обработки с другими перемещаемыми и разделяемыми объектными файлами для создания другого объектного файла или для объединения с исполняемым файлом для создания образа процесса.

ELF файл состоит из трёх частей: заголовка файла (ELF Header), таблицы заголовков программы (Program Header) и таблицы заголовков секций (Section Header).

Заголовок файла находится в начале файла и является «дорожной картой», описывающей организацию файла. В 32-битном файле заголовок имеет размер 52 байта. В начале заголовка находятся 16 идентификационных байт e\_ident, представляющих основные характеристики файла. Первые четыре байта Magic всегда равны 7f, 45, 4c, 46 соответственно. Они определяют, является ли файл корректным ELF файлом. Следующий байт Class содержит информацию о разрядности файла (32-битный или 64-битный). Байт Data отвечает за метод кодирования данных (Little или Big Endian). Байт версии Version всегда устанавливается равным единице. Следующий байт OS ABI определяет двоичный интерфейс приложения системы. По умолчанию он ставится в ноль. Ещё один байт ABI Version почти никогда не используется. Оставшиеся 7 идентификационных байт являются набивкой (padding bytes) и зарезервированы для будущего использования. Значение e\_type хранит тип файла. Архитектура аппаратной платформы файла определяется значением e\_machine. Всегда равное единице значение e\_version содержит версию формата. Значение e\_entry хранит виртуальный адрес точки входа, которому система передаёт управление при запуске процесса. Размер заголовка файла хранится в e\_ehsize. Размер одного заголовка программы и размер одного заголовка секции определяются значениями e\_phentsize и e\_shentsize соответственно. Все заголовки программы имеют одинаковый размер (32 для 32-битных файлов), как и заголовки секций (40 для 32-битных файлов). Число заголовков программы и число заголовков секций содержатся соответственно в значениях e\_phnum и e\_shnum. Значение e\_shstrndx определяет индекс записи в таблице заголовков секций, описывающей таблицу названий секций .shstrtab, содержащую подряд записанные строки названий секций файла.

Таблица заголовков программы содержит заголовки, описывающие отдельные сегменты программы и информацию о них, необходимую системы для подготовки программы к исполнению. Таблица заголовков программы не является объектом исследования данной работы.

Таблица заголовков секций содержит атрибуты секций ELF файла. Она необходима только компоновщику, исполняемые файлы в ней не нуждаются. Предоставленная в таблице информация используется для размещения данных при сборке файла. Каждый заголовок секции (Section Header) обладает полезной информацией. Значение sh\_name является смещением строки с названием данной секции относительно начала таблицы названий секций .shstrtab. Значение sh\_type определяет тип заголовка, дающий понимание какую информацию содержит секция. Существует некоторое число различных типов заголовков. Секция типа SHT\_PROGBITS содержит определённую программой информацию. Такой тип имеет секция .text, содержащая код программы. Единственная на файл секция с типом заголовка SHT\_SYMTAB содержит таблицу символов, секция с типом заголовка SHT\_STRTAB содержит одну из таблиц строк файла. Таблица символов .symtab содержит информацию, необходимую для поиска и перемещения символических определений и ссылок программы. Таблица строк .strtab состоит из последовательностей символов (строк), заканчивающихся нулевыми байтами. Эти строки используются для представления символов и названий секций. Значение sh\_flags хранит различные атрибуты секции. Если секция должна быть загружена в память, то поле sh\_addr указывает адрес, начиная с которого будет загружена эта секция. Значение sh\_offset определяет смещение секции от начала файла в байтах. Размер секции содержится в sh\_size и может быть нулевым. Значение sh\_link является индексом ассоциированной секции и может иметь различную интерпретацию в зависимости от типа заголовка. Значение sh\_info содержит дополнительную информацию о секции. Поля sh\_addralign и sh\_entsize хранят необходимое выравнивание и размер в байтах каждой записи секции соответственно.

**Практическая часть**

Проект написан на Java. Для хранения и работы с элементами ELF файла написан абстрактный класс Element. Класс ElfFile содержит файл и необходимые для дизассемблера его части. Класс Disassembler предназначен для самого дизассемблирования. В методе main класса hw4 создаётся Disassembler и вызывается метод disassemble(String filename). Имена входного и выходного файлов заданы как аргументы командной строки.

При создании ElfFile прочитываются его ELF заголовок, заголовок .shstrtab, заголовок .text, заголовок .strtab и таблица символов .symtab. Из ELF заголовка определяются нужные смещения и размеры секций для их считывания и последующей работы с ними. Для нахождения заголовка секции .text в таблице .shstrtab ищется строка .text и по её смещению относительно начала таблицы в разделе секций находится нужный заголовок. Аналогичный алгоритм проводится при чтении заголовка таблицы строк .strtab. Таблица символов .symtab имеется единственная на весь файл, поэтому обнаруживается при поиске заголовка секции с типом, равным 2. Символы таблицы читаются из таблицы строк .strtab.

При создании Disassembler ему задаётся ELF файл и необходимые для работы элементы и атрибуты. Заполняется ассоциативный массив меток, читаемых из SymbolTable файла, для их указания на адресах. Далее заполняется ассоциативный массив меток переходов для их проставления на командах и адресах перехода. Чтобы понять, какие метки и где ставить, предварительно пробегается секция .text с целью обнаружить команды условных и безусловных переходов и прикрепить к ним метки. Метки имеют адреса памяти, а не адреса внутри файла, поэтому изначальный адрес устанавливается в значение адреса из заголовка .text. Метод disassembler(String filename) записывает в файл сначала секцию .text, потом через перевод строки таблицу символов .symtab.

Секция .text дизассемблируется последовательным считыванием инструкций, пока количество пройденных байт не достигнет размера секции, указанного в заголовке .text. Изначально считается 16 бит (2 байта). Если такая инструкция RVC является корректной, то алгоритм повторяется. Если же строка не является инструкцией (unknown\_command), то считываются следующие 2 байта и проверяется корректность 32-битной команды RV32I или RV32M. Если же и она не корректна, то на рассмотрение остаётся вариант, когда первая 16-битная инструкция некорректна, а за ней следует новая 16-битная инструкция. Инструкции считываются и дизассемблируются в файл в соответствии с документацией RISC-V. Некоторые избыточные выражения оставлены для точности и понятности определения вида инструкции. Таким образом, считывается вся секция .text, содержащая код программы.

Строки таблицы символов .symtab записываются как элементы списка symbolTable файла ElfFile в цикле. Для каждого элемента выводятся соответствующие значения и названия.

**Экспериментальная часть**

Программа протестирована на нескольких elf файлах и при тестировании, насколько можно судить, дизассемблировала инструкции и таблицу символов корректно и в правильном формате. Ниже приводится результат работы программы на одном из тестов.

.text

00010074 register\_fini: addi a5, zero, 0

00010078 beq a5, zero, 8, LOC\_10088

0001007c lui a0, 16

00010080 addi a0, a0, 1164

00010084 LOC\_10084: jal zero, 506, LOC\_10084

00010088 LOC\_10088: jalr zero, ra

0001008c \_start: auipc gp, 2

00010090 addi gp, gp, 3412

00010094 addi a0, gp, 3124

00010098 addi a2, gp, 3152

0001009c sub a2, a2, a0

000100a0 addi a1, zero, 0

000100a4 LOC\_100a4: jal ra, 236, LOC\_100a4

000100a8 auipc a0, 0

000100ac addi a0, a0, 976

000100b0 beq a0, zero, 8, LOC\_100c0

000100b4 auipc a0, 0

000100b8 addi a0, a0, 984

000100bc LOC\_100bc: jal ra, 478, LOC\_100bc

000100c0 LOC\_100c0: jal ra, 144, LOC\_100c0

000100c4 lw a0, 0(sp)

000100c8 addi a1, sp, 4

000100cc addi a2, zero, 0

000100d0 LOC\_100d0: jal ra, 58, LOC\_100d0

000100d4 LOC\_100d4: jal zero, 110, LOC\_100d4

000100d8 \_\_do\_global\_dtors\_aux: lbu a4, 3124(gp)

000100dc bne a4, zero, 34, LOC\_100e0

000100e0 LOC\_100e0: addi sp, sp, 4080

000100e4 sw s0, 8(sp)

000100e8 addi s0, a5, 0

000100ec sw ra, 12(sp)

000100f0 addi a5, zero, 0

000100f4 beq a5, zero, 10, LOC\_10108

000100f8 lui a0, 17

000100fc addi a0, a0, 1484

00010100 auipc ra, 0

00010104 jalr ra, zero

00010108 LOC\_10108: addi a5, zero, 1

0001010c lw ra, 12(sp)

00010110 sb a5, 3124(gp)

00010114 lw s0, 8(sp)

00010118 addi sp, sp, 16

0001011c jalr zero, ra

00010120 jalr zero, ra

00010124 frame\_dummy: addi a5, zero, 0

00010128 beq a5, zero, 12, LOC\_10140

0001012c lui a0, 17

00010130 addi a1, gp, 3128

00010134 addi a0, a0, 1484

00010138 auipc t1, 0

0001013c jalr zero, zero

00010140 LOC\_10140: jalr zero, ra

00010144 main: addi sp, sp, 4064

00010148 sw s0, 28(sp)

0001014c addi s0, sp, 32

00010150 addi a5, zero, 2

00010154 sw a5, 4068(s0)

00010158 addi a5, zero, 3

0001015c sw a5, 4064(s0)

00010160 sw zero, 4076(s0)

00010164 sw zero, 4072(s0)

00010168 LOC\_10168: jal zero, 16, LOC\_10168

0001016c lw a4, 4076(s0)

00010170 lw a5, 4072(s0)

00010174 add a5, a4, a5

00010178 sw a5, 4076(s0)

0001017c lw a5, 4072(s0)

00010180 addi a5, a5, 1

00010184 sw a5, 4072(s0)

00010188 lw a4, 4068(s0)

0001018c lw a5, 4064(s0)

00010190 mul a5, a4, a5

00010194 lw a4, 4072(s0)

00010198 blt a4, a5, 4074, LOC\_109ac

0001019c addi a5, zero, 0

000101a0 addi a0, a5, 0

000101a4 lw s0, 28(sp)

000101a8 addi sp, sp, 32

000101ac jalr zero, ra

000101b0 exit: addi sp, sp, 4080

000101b4 addi a1, zero, 0

000101b8 sw s0, 8(sp)

000101bc sw ra, 12(sp)

000101c0 addi s0, a0, 0

000101c4 LOC\_101c4: jal ra, 202, LOC\_101c4

000101c8 lw a0, 3112(gp)

000101cc lw a5, 60(a0)

000101d0 beq a5, zero, 4, LOC\_101d8

000101d4 jalr ra, a5

000101d8 LOC\_101d8: addi a0, s0, 0

000101dc LOC\_101dc: jal ra, 466, LOC\_101dc

000101e0 \_\_libc\_init\_array: addi sp, sp, 4080

000101e4 sw s0, 8(sp)

000101e8 sw s2, 0(sp)

000101ec lui s0, 17

000101f0 lui s2, 17

000101f4 addi a5, s0, 1488

000101f8 addi s2, s2, 1488

000101fc sub s2, s2, a5

00010200 sw ra, 12(sp)

00010204 sw s1, 4(sp)

00010208 srai s2, s2, 2

0001020c LOC\_1020c: beq s2, zero, 16, LOC\_1020c

00010210 addi s0, s0, 1488

00010214 addi s1, zero, 0

00010218 lw a5, 0(s0)

0001021c addi s1, s1, 1

00010220 addi s0, s0, 4

00010224 jalr ra, a5

00010228 bne s2, s1, 4088, LOC\_10a38

0001022c lui s0, 17

00010230 lui s2, 17

00010234 addi a5, s0, 1488

00010238 addi s2, s2, 1496

0001023c sub s2, s2, a5

00010240 srai s2, s2, 2

00010244 LOC\_10244: beq s2, zero, 16, LOC\_10244

00010248 addi s0, s0, 1488

0001024c addi s1, zero, 0

00010250 lw a5, 0(s0)

00010254 addi s1, s1, 1

00010258 addi s0, s0, 4

0001025c jalr ra, a5

00010260 bne s2, s1, 4088, LOC\_10a70

00010264 lw ra, 12(sp)

00010268 lw s0, 8(sp)

0001026c lw s1, 4(sp)

00010270 lw s2, 0(sp)

00010274 addi sp, sp, 16

00010278 jalr zero, ra

0001027c memset: addi t1, zero, 15

00010280 addi a4, a0, 0

00010284 bgeu t1, a2, 30, LOC\_102a0

00010288 andi a5, a4, 15

0001028c LOC\_1028c: bne a5, zero, 80, LOC\_1028c

00010290 bne a1, zero, 66, LOC\_10294

00010294 LOC\_10294: andi a3, a2, 4080

00010298 andi a2, a2, 15

0001029c add a3, a3, a4

000102a0 LOC\_102a0: sw a1, 0(a4)

000102a4 sw a1, 4(a4)

000102a8 sw a1, 8(a4)

000102ac sw a1, 12(a4)

000102b0 addi a4, a4, 16

000102b4 bltu a4, a3, 4086, LOC\_10ac0

000102b8 bne a2, zero, 4, LOC\_102c0

000102bc jalr zero, ra

000102c0 LOC\_102c0: sub a3, t1, a2

000102c4 slli a3, a3, 2

000102c8 auipc t0, 0

000102cc add a3, a3, t0

000102d0 jalr zero, a3

000102d4 sb a1, 14(a4)

000102d8 sb a1, 13(a4)

000102dc sb a1, 12(a4)

000102e0 sb a1, 11(a4)

000102e4 sb a1, 10(a4)

000102e8 sb a1, 9(a4)

000102ec sb a1, 8(a4)

000102f0 sb a1, 7(a4)

000102f4 sb a1, 6(a4)

000102f8 sb a1, 5(a4)

000102fc sb a1, 4(a4)

00010300 sb a1, 3(a4)

00010304 sb a1, 2(a4)

00010308 sb a1, 1(a4)

0001030c sb a1, 0(a4)

00010310 jalr zero, ra

00010314 andi a1, a1, 255

00010318 slli a3, a1, 8

0001031c or a1, a1, a3

00010320 slli a3, a1, 16

00010324 or a1, a1, a3

00010328 jal zero, 1048502, LOC\_1f328

0001032c slli a3, a5, 2

00010330 auipc t0, 0

00010334 add a3, a3, t0

00010338 addi t0, ra, 0

0001033c jalr ra, a3

00010340 addi ra, t0, 0

00010344 addi a5, a5, 4080

00010348 sub a4, a4, a5

0001034c add a2, a2, a5

00010350 bgeu t1, a2, 4024, LOC\_10b60

00010354 jal zero, 1048478, LOC\_1f354

00010358 \_\_call\_exitprocs: addi sp, sp, 4048

0001035c sw s4, 24(sp)

00010360 lw s4, 3112(gp)

00010364 sw s2, 32(sp)

00010368 sw ra, 44(sp)

0001036c lw s2, 328(s4)

00010370 sw s0, 40(sp)

00010374 sw s1, 36(sp)

00010378 sw s3, 28(sp)

0001037c sw s5, 20(sp)

00010380 sw s6, 16(sp)

00010384 sw s7, 12(sp)

00010388 sw s8, 8(sp)

0001038c LOC\_1038c: beq s2, zero, 32, LOC\_1038c

00010390 addi s6, a0, 0

00010394 addi s7, a1, 0

00010398 addi s5, zero, 1

0001039c addi s3, zero, 4095

000103a0 lw s1, 4(s2)

000103a4 addi s0, s1, 4095

000103a8 blt s0, zero, 18, LOC\_103ac

000103ac LOC\_103ac: slli s1, s1, 2

000103b0 add s1, s2, s1

000103b4 beq s7, zero, 36, LOC\_103bc

000103b8 lw a5, 260(s1)

000103bc LOC\_103bc: beq a5, s7, 32, LOC\_103bc

000103c0 addi s0, s0, 4095

000103c4 addi s1, s1, 4092

000103c8 bne s0, s3, 4086, LOC\_10bd4

000103cc lw ra, 44(sp)

000103d0 lw s0, 40(sp)

000103d4 lw s1, 36(sp)

000103d8 lw s2, 32(sp)

000103dc lw s3, 28(sp)

000103e0 lw s4, 24(sp)

000103e4 lw s5, 20(sp)

000103e8 lw s6, 16(sp)

000103ec lw s7, 12(sp)

000103f0 lw s8, 8(sp)

000103f4 addi sp, sp, 48

000103f8 jalr zero, ra

000103fc lw a5, 4(s2)

00010400 lw a3, 4(s1)

00010404 addi a5, a5, 4095

00010408 beq a5, s0, 46, LOC\_10424

0001040c sw zero, 4(s1)

00010410 beq a3, zero, 4056, LOC\_10c20

00010414 lw a5, 392(s2)

00010418 sll a4, s5, s0

0001041c lw s8, 4(s2)

00010420 and a5, a4, a5

00010424 LOC\_10424: bne a5, zero, 18, LOC\_10428

00010428 LOC\_10428: jalr ra, a3

0001042c lw a4, 4(s2)

00010430 lw a5, 328(s4)

00010434 bne a4, s8, 4, LOC\_1043c

00010438 beq a5, s2, 4036, LOC\_10c40

0001043c LOC\_1043c: beq a5, zero, 4040, LOC\_10c4c

00010440 addi s2, a5, 0

00010444 jal zero, 1048494, LOC\_1f444

00010448 lw a5, 396(s2)

0001044c lw a1, 132(s1)

00010450 and a4, a4, a5

00010454 bne a4, zero, 12, LOC\_1046c

00010458 addi a0, s6, 0

0001045c jalr ra, a3

00010460 jal zero, 1048550, LOC\_1f460

00010464 sw s0, 4(s2)

00010468 jal zero, 1048532, LOC\_1f468

0001046c LOC\_1046c: addi a0, a1, 0

00010470 jalr ra, a3

00010474 jal zero, 1048540, LOC\_1f474

00010478 atexit: addi a1, a0, 0

0001047c addi a3, zero, 0

00010480 addi a2, zero, 0

00010484 addi a0, zero, 0

00010488 LOC\_10488: jal zero, 48, LOC\_10488

0001048c \_\_libc\_fini\_array: addi sp, sp, 4080

00010490 sw s0, 8(sp)

00010494 lui a5, 17

00010498 lui s0, 17

0001049c addi s0, s0, 1496

000104a0 addi a5, a5, 1500

000104a4 sub a5, a5, s0

000104a8 sw s1, 4(sp)

000104ac sw ra, 12(sp)

000104b0 srai s1, a5, 2

000104b4 LOC\_104b4: beq s1, zero, 16, LOC\_104b4

000104b8 addi a5, a5, 4092

000104bc add s0, a5, s0

000104c0 lw a5, 0(s0)

000104c4 addi s1, s1, 4095

000104c8 addi s0, s0, 4092

000104cc jalr ra, a5

000104d0 bne s1, zero, 4088, LOC\_10ce0

000104d4 lw ra, 12(sp)

000104d8 lw s0, 8(sp)

000104dc lw s1, 4(sp)

000104e0 addi sp, sp, 16

000104e4 jalr zero, ra

000104e8 \_\_register\_exitproc: lw a4, 3112(gp)

000104ec lw a5, 328(a4)

000104f0 beq a5, zero, 44, LOC\_10508

000104f4 lw a4, 4(a5)

000104f8 addi a6, zero, 31

000104fc blt a6, a4, 62, LOC\_10518

00010500 slli a6, a4, 2

00010504 beq a0, zero, 22, LOC\_10510

00010508 LOC\_10508: add t1, a5, a6

0001050c sw a2, 136(t1)

00010510 LOC\_10510: lw a7, 392(a5)

00010514 addi a2, zero, 1

00010518 LOC\_10518: sll a2, a2, a4

0001051c or a7, a7, a2

00010520 sw a7, 392(a5)

00010524 sw a3, 264(t1)

00010528 addi a3, zero, 2

0001052c beq a0, a3, 20, LOC\_10534

00010530 addi a4, a4, 1

00010534 LOC\_10534: sw a4, 4(a5)

00010538 add a5, a5, a6

0001053c sw a1, 8(a5)

00010540 addi a0, zero, 0

00010544 jalr zero, ra

00010548 addi a5, a4, 332

0001054c sw a5, 328(a4)

00010550 jal zero, 1048530, LOC\_1f550

00010554 lw a3, 396(a5)

00010558 addi a4, a4, 1

0001055c sw a4, 4(a5)

00010560 or a2, a3, a2

00010564 sw a2, 396(a5)

00010568 add a5, a5, a6

0001056c sw a1, 8(a5)

00010570 addi a0, zero, 0

00010574 jalr zero, ra

00010578 addi a0, zero, 4095

0001057c jalr zero, ra

00010580 \_exit: addi a1, zero, 0

00010584 addi a2, zero, 0

00010588 addi a3, zero, 0

0001058c addi a4, zero, 0

00010590 addi a5, zero, 0

00010594 addi a7, zero, 93

00010598 ecall

0001059c blt a0, zero, 4, LOC\_105a4

000105a0 LOC\_105a0: jal zero, 0, LOC\_105a0

000105a4 LOC\_105a4: addi sp, sp, 4080

000105a8 sw s0, 8(sp)

000105ac addi s0, a0, 0

000105b0 sw ra, 12(sp)

000105b4 sub s0, zero, s0

000105b8 LOC\_105b8: jal ra, 6, LOC\_105b8

000105bc sw s0, 0(a0)

000105c0 LOC\_105c0: jal zero, 0, LOC\_105c0

000105c4 \_\_errno: lw a0, 3120(gp)

000105c8 jalr zero, ra

.symtab

Symbol Value Size Type Bind Vis Index Name

[ 0] 0x0 0 NOTYPE LOCAL DEFAULT UNDEF

[ 1] 0x10074 0 SECTION LOCAL DEFAULT 1

[ 2] 0x115CC 0 SECTION LOCAL DEFAULT 2

[ 3] 0x115D0 0 SECTION LOCAL DEFAULT 3

[ 4] 0x115D8 0 SECTION LOCAL DEFAULT 4

[ 5] 0x115E0 0 SECTION LOCAL DEFAULT 5

[ 6] 0x11A08 0 SECTION LOCAL DEFAULT 6

[ 7] 0x11A14 0 SECTION LOCAL DEFAULT 7

[ 8] 0x0 0 SECTION LOCAL DEFAULT 8

[ 9] 0x0 0 SECTION LOCAL DEFAULT 9

[ 10] 0x0 0 FILE LOCAL DEFAULT ABS \_\_call\_atexit.c

[ 11] 0x10074 24 FUNC LOCAL DEFAULT 1 register\_fini

[ 12] 0x0 0 FILE LOCAL DEFAULT ABS crtstuff.c

[ 13] 0x115CC 0 OBJECT LOCAL DEFAULT 2

[ 14] 0x100D8 0 FUNC LOCAL DEFAULT 1 \_\_do\_global\_dtors\_aux

[ 15] 0x11A14 1 OBJECT LOCAL DEFAULT 7 completed.1

[ 16] 0x115D8 0 OBJECT LOCAL DEFAULT 4 \_\_do\_global\_dtors\_aux\_fini\_array\_entry

[ 17] 0x10124 0 FUNC LOCAL DEFAULT 1 frame\_dummy

[ 18] 0x11A18 24 OBJECT LOCAL DEFAULT 7 object.0

[ 19] 0x115D4 0 OBJECT LOCAL DEFAULT 3 \_\_frame\_dummy\_init\_array\_entry

[ 20] 0x0 0 FILE LOCAL DEFAULT ABS test.c

[ 21] 0x0 0 FILE LOCAL DEFAULT ABS exit.c

[ 22] 0x0 0 FILE LOCAL DEFAULT ABS impure.c

[ 23] 0x115E0 1064 OBJECT LOCAL DEFAULT 5 impure\_data

[ 24] 0x0 0 FILE LOCAL DEFAULT ABS init.c

[ 25] 0x0 0 FILE LOCAL DEFAULT ABS atexit.c

[ 26] 0x0 0 FILE LOCAL DEFAULT ABS fini.c

[ 27] 0x0 0 FILE LOCAL DEFAULT ABS \_\_atexit.c

[ 28] 0x0 0 FILE LOCAL DEFAULT ABS sys\_exit.c

[ 29] 0x0 0 FILE LOCAL DEFAULT ABS errno.c

[ 30] 0x0 0 FILE LOCAL DEFAULT ABS crtstuff.c

[ 31] 0x115CC 0 OBJECT LOCAL DEFAULT 2 \_\_FRAME\_END\_\_

[ 32] 0x0 0 FILE LOCAL DEFAULT ABS

[ 33] 0x115DC 0 NOTYPE LOCAL DEFAULT 4 \_\_fini\_array\_end

[ 34] 0x115D8 0 NOTYPE LOCAL DEFAULT 4 \_\_fini\_array\_start

[ 35] 0x115D8 0 NOTYPE LOCAL DEFAULT 3 \_\_init\_array\_end

[ 36] 0x115D0 0 NOTYPE LOCAL DEFAULT 3 \_\_preinit\_array\_end

[ 37] 0x115D0 0 NOTYPE LOCAL DEFAULT 3 \_\_init\_array\_start

[ 38] 0x115D0 0 NOTYPE LOCAL DEFAULT 3 \_\_preinit\_array\_start

[ 39] 0x11DE0 0 NOTYPE GLOBAL DEFAULT ABS \_\_global\_pointer$

[ 40] 0x105C4 8 FUNC GLOBAL DEFAULT 1 \_\_errno

[ 41] 0x11A08 0 NOTYPE GLOBAL DEFAULT 6 \_\_SDATA\_BEGIN\_\_

[ 42] 0x11A0C 0 OBJECT GLOBAL HIDDEN 6 \_\_dso\_handle

[ 43] 0x11A08 4 OBJECT GLOBAL DEFAULT 6 \_global\_impure\_ptr

[ 44] 0x101E0 156 FUNC GLOBAL DEFAULT 1 \_\_libc\_init\_array

[ 45] 0x1048C 92 FUNC GLOBAL DEFAULT 1 \_\_libc\_fini\_array

[ 46] 0x10358 288 FUNC GLOBAL DEFAULT 1 \_\_call\_exitprocs

[ 47] 0x1008C 76 FUNC GLOBAL DEFAULT 1 \_start

[ 48] 0x104E8 152 FUNC GLOBAL DEFAULT 1 \_\_register\_exitproc

[ 49] 0x11A30 0 NOTYPE GLOBAL DEFAULT 7 \_\_BSS\_END\_\_

[ 50] 0x11A14 0 NOTYPE GLOBAL DEFAULT 7 \_\_bss\_start

[ 51] 0x1027C 220 FUNC GLOBAL DEFAULT 1 memset

[ 52] 0x10144 108 FUNC GLOBAL DEFAULT 1 main

[ 53] 0x10478 20 FUNC GLOBAL DEFAULT 1 atexit

[ 54] 0x11A10 4 OBJECT GLOBAL DEFAULT 6 \_impure\_ptr

[ 55] 0x115E0 0 NOTYPE GLOBAL DEFAULT 5 \_\_DATA\_BEGIN\_\_

[ 56] 0x11A14 0 NOTYPE GLOBAL DEFAULT 6 \_edata

[ 57] 0x11A30 0 NOTYPE GLOBAL DEFAULT 7 \_end

[ 58] 0x101B0 48 FUNC GLOBAL DEFAULT 1 exit

[ 59] 0x10580 68 FUNC GLOBAL DEFAULT 1 \_exit

**Листинг**

Компилятор javac, OpenJDK 16.0.2

**hw4.java**

import java.io.\*;  
  
public class hw4 {  
 public static void main(String[] args) {  
 try {  
 Disassembler d = new Disassembler(new ElfFile(args[0]));  
 d.disassemble(args[1]);  
 } catch (IOException e) {  
 System.*err*.println(e.getMessage());  
 }  
 }  
}

**Disassembler.java**

import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class Disassembler {  
 private final ElfFile file;  
 private int address;  
 private int byteNumber;  
 private BufferedInputStream reader;  
 private final Map<Integer, String> labels;  
 private final Map<Integer, String> jumps;  
 private static final String[] *REGISTERS* = new String[] {  
 "zero", "ra", "sp", "gp", "tp", "t0", "t1", "t2",  
 "s0", "s1", "a0", "a1", "a2", "a3", "a4", "a5",  
 "a6", "a7", "s2", "s3", "s4", "s5", "s6", "s7",  
 "s8", "s9", "s10", "s11", "t3", "t4", "t5", "t6"  
 };  
  
 public Disassembler(ElfFile file) throws IOException {  
 this.file = file;  
 this.address = file.textHeader.sh\_addr;  
 this.byteNumber = 0;  
 this.labels = new HashMap<>();  
 markLabels();  
 *// Mark jumps* this.reader = new BufferedInputStream(new FileInputStream(file.source));  
 this.reader.skip(file.textHeader.sh\_offset);  
 this.jumps = new HashMap<>();  
 markJumps();  
 reader.close();  
 *// Making ready to disassemble* this.address = file.textHeader.sh\_addr;  
 this.byteNumber = 0;  
 this.reader = new BufferedInputStream(new FileInputStream(file.source));  
 this.reader.skip(file.textHeader.sh\_offset);  
 }  
  
 public void disassemble(String filename) {  
 try (BufferedWriter writer =  
 new BufferedWriter(  
 new OutputStreamWriter(  
 new FileOutputStream(  
 filename  
 ),  
 StandardCharsets.*UTF\_8* )  
 )  
 ) {  
 writer.write(".text");  
 writer.write(System.*lineSeparator*());  
 writer.write(writeTextSection());  
 writer.write(System.*lineSeparator*());  
 writer.write(".symtab");  
 writer.write(System.*lineSeparator*());  
 writer.write(writeSymbolTable());  
 reader.close();  
 file.closeAll();  
 } catch (IOException e) {  
 System.*err*.println(e.getMessage());  
 }  
 }  
  
 private String writeTextSection() throws IOException {  
 StringBuilder stringBuilder = new StringBuilder();  
 boolean previousReceived = true;  
 int previousCode = 0;  
 while (byteNumber < file.textHeader.sh\_size) {  
 String label = getLabel(address);  
 int currentCode = read16bit();  
 if (previousReceived) {  
 *// Checking current 16bit code* String instruction = get16bitInstruction(currentCode);  
 if (!instruction.equals("unknown\_command")) {  
 if (isJumpInstruction(currentCode)) {  
 label = "";  
 }  
 stringBuilder.append(writeInstruction(label, instruction));  
 stringBuilder.append(System.*lineSeparator*());  
 address += 2;  
 byteNumber += 2;  
 } else {  
 previousCode = currentCode;  
 previousReceived = false;  
 }  
 } else {  
 *// Checking all 32bit code* int code = previousCode + currentCode \* 0x10000;  
 String instruction = get32bitInstruction(code);  
 if (!instruction.equals("unknown\_command")) {  
 if (isJumpInstruction(code)) {  
 label = "";  
 }  
 stringBuilder.append(writeInstruction(label, instruction));  
 stringBuilder.append(System.*lineSeparator*());  
 address += 4;  
 byteNumber += 4;  
 previousReceived = true;  
 } else {  
 *// Writing previous 16bit code* stringBuilder.append(writeInstruction(label, "unknown\_command"));  
 stringBuilder.append(System.*lineSeparator*());  
 address += 2;  
 byteNumber += 2;  
 label = getLabel(address);  
 *// Checking current 16bit code* instruction = get16bitInstruction(currentCode);  
 if (!instruction.equals("unknown\_command")) {  
 if (isJumpInstruction(currentCode)) {  
 label = "";  
 }  
 stringBuilder.append(writeInstruction(label, instruction));  
 } else {  
 stringBuilder.append(writeInstruction(label, "unknown\_command"));  
 }  
 stringBuilder.append(System.*lineSeparator*());  
 previousReceived = true;  
 address += 2;  
 byteNumber += 2;  
 }  
 }  
 }  
 return stringBuilder.toString();  
 }  
  
 private String writeInstruction(String label, String instruction) {  
 if (label.length() != 0) {  
 return String.*format*("%08x %10s: %s", address, label, instruction);  
 } else {  
 return String.*format*("%08x %10s %s", address, label, instruction);  
 }  
 }  
  
 private String writeSymbolTable() {  
 StringBuilder stringBuilder = new StringBuilder();  
 final List<SymbolTableEntry> table = file.symbolTable.table;  
 stringBuilder.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 < file.symbolTable.numberOfEntries; i++) {  
 String type = switch(table.get(i).st\_type) {  
 case 0 -> "NOTYPE";  
 case 1 -> "OBJECT";  
 case 2 -> "FUNC";  
 case 3 -> "SECTION";  
 case 4 -> "FILE";  
 case 13 -> "LOPROC";  
 case 15 -> "HIPROC";  
 default -> null;  
 };  
 String bind = switch(table.get(i).st\_bind) {  
 case 0 -> "LOCAL";  
 case 1 -> "GLOBAL";  
 case 2 -> "WEAK";  
 case 13 -> "LOPROC";  
 case 15 -> "HIPROC";  
 default -> null;  
 };  
 String vis = switch(table.get(i).st\_vis) {  
 case 0 -> "DEFAULT";  
 case 1 -> "INTERNAL";  
 case 2 -> "HIDDEN";  
 case 3 -> "PROTECTED";  
 case 4 -> "EXPORTED";  
 case 5 -> "SINGLETON";  
 case 6 -> "ELIMINATE";  
 default -> null;  
 };  
 String index = switch(table.get(i).st\_shndx) {  
 case 0 -> "UNDEF";  
 case 0xff00 -> "LORESERVE";  
 case 0xff1f -> "HIPROC";  
 case 0xfff1 -> "ABS";  
 case 0xfff2 -> "COMMON";  
 case 0xffff -> "HIRESERVE";  
 default -> String.*valueOf*(table.get(i).st\_shndx);  
 };  
 stringBuilder.append(String.*format*(  
 "[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s\n",  
 i, table.get(i).st\_value, table.get(i).st\_size, type, bind, vis, index, table.get(i).name  
 )  
 );  
 }  
 return stringBuilder.toString();  
 }  
  
 private void markLabels() {  
 for (int i = 0; i < file.symbolTable.table.size(); i++) {  
 *// Finding FUNC entries* if (file.symbolTable.table.get(i).st\_type == 2) {  
 labels.put(file.symbolTable.table.get(i).st\_value, file.symbolTable.table.get(i).name);  
 }  
 }  
 }  
  
 private String getLabel(int addr) {  
 if (labels.containsKey(addr)) {  
 if (labels.get(addr).length() != 0) {  
 return labels.get(addr);  
 } else {  
 return String.*format*("LOC\_%05x", addr);  
 }  
 } else {  
 return "";  
 }  
 }  
  
 private void markJumps() throws IOException {  
 boolean previousReceived = true;  
 int previousCode = 0;  
 int labelNumber = 0;  
 while (byteNumber < file.textHeader.sh\_size) {  
 int currentCode = read16bit();  
 if (previousReceived) {  
 *// Checking current 16bit code* if (isJumpInstruction(currentCode)) {  
 int jumpAddress = address + getJumpOffset(currentCode);  
 if (!labels.containsKey(jumpAddress)) {  
 labels.put(jumpAddress, String.*format*("LOC\_%05x", labelNumber));  
 }  
 jumps.put(address, labels.get(jumpAddress));  
 labelNumber++;  
 address += 2;  
 byteNumber += 2;  
 } else {  
 previousCode = currentCode;  
 previousReceived = false;  
 }  
 } else {  
 *// Checking all 32bit code* int code = previousCode + currentCode \* 0x10000;  
 if (isJumpInstruction(code)) {  
 int jumpAddress = address + getJumpOffset(code);  
 if (!labels.containsKey(jumpAddress)) {  
 labels.put(jumpAddress, String.*format*("LOC\_%05x", labelNumber));  
 }  
 jumps.put(address, labels.get(jumpAddress));  
 labelNumber++;  
 address += 4;  
 byteNumber += 4;  
 previousReceived = true;  
 } else {  
 address += 2;  
 byteNumber += 2;  
 *// Checking current 16bit code* if (isJumpInstruction(currentCode)) {  
 int jumpAddress = address + getJumpOffset(currentCode);  
 if (!labels.containsKey(jumpAddress)) {  
 labels.put(jumpAddress, String.*format*("LOC\_%05x", labelNumber));  
 }  
 jumps.put(address, labels.get(jumpAddress));  
 labelNumber++;  
 }  
 previousReceived = true;  
 address += 2;  
 byteNumber += 2;  
 }  
 }  
 }  
 }  
  
 private boolean isJumpInstruction(int instruction) {  
 int opcode = instruction << 25 >>> 25;  
 if (opcode == 0b1101111) {  
 *// JAL* return true;  
 } else if (opcode == 0b1100011) {  
 *// BEQ BNE BLT BGE BLTU BGEU* return true;  
 }  
 int op = instruction << 14 >>> 14;  
 int func3 = instruction >>> 13;  
 if (op == 0b01) {  
 if (func3 == 0b001 || func3 == 0b101) {  
 *// C.JAL C.J* return true;  
 } else if (func3 == 0b110 || func3 == 0b111) {  
 *// C.BEQZ C.BNEZ* return true;  
 }  
 }  
 return false;  
 }  
  
 private int getJumpOffset(int instruction) {  
 int opcode = instruction << 25 >>> 25;  
 if (opcode == 0b1101111) {  
 *// JAL* return (instruction << 1 >>> 22) \* 0b10 +  
 (instruction << 11 >>> 31) \* 0b100000000000 +  
 (instruction << 12 >>> 24) \* 0b1000000000000 +  
 (instruction >>> 31) \* 0b100000000000000000000;  
 } else if (opcode == 0b1100011) {  
 *// BEQ BNE BLT BGE BLTU BGEU* return (instruction << 20 >>> 28) \* 0b10 +  
 (instruction << 1 >>> 26) \* 0b100000 +  
 (instruction << 24 >>> 31) \* 0b100000000000 +  
 (instruction >>> 31) \* 0b1000000000000;  
 }  
 int op = instruction << 14 >>> 14;  
 int func3 = instruction >>> 13;  
 if (op == 0b01) {  
 if (func3 == 0b001 || func3 == 0b101) {  
 *// C.JAL C.J* return (instruction << 10 >>> 13) \* 0b10 +  
 (instruction << 4 >>> 15) \* 0b10000 +  
 (instruction << 13 >>> 15) \* 0b100000 +  
 (instruction << 8 >>> 15) \* 0b1000000 +  
 (instruction << 9 >>> 15) \* 0b10000000 +  
 (instruction << 5 >>> 14) \* 0b100000000 +  
 (instruction << 7 >>> 15) \* 0b10000000000 +  
 (instruction << 3 >>> 15) \* 0b100000000000;  
 } else if (func3 == 0b110 || func3 == 0b111) {  
 *// C.BEQZ C.BNEZ* return (instruction << 11 >>> 14) \* 0b10 +  
 (instruction << 4 >>> 14) \* 0b1000 +  
 (instruction << 13 >>> 15) \* 0b100000 +  
 (instruction << 9 >>> 14) \* 0b1000000 +  
 (instruction << 3 >>> 15) \* 0b100000000;  
 }  
 }  
 return 0;  
 }  
  
 private String get16bitInstruction(int instruction) {  
 int op = instruction << 14 >>> 14;  
 int func3 = instruction >>> 13;  
 if (op == 0b00) {  
 int rdc = instruction << 11 >>> 13;  
 int rs2c = rdc;  
 int rs1c = instruction << 6 >>> 13;  
 long uimm = ((long) instruction << 9 >>> 15) +  
 ((long) instruction << 3 >>> 13) \* 0b10 +  
 ((long) instruction << 10 >>> 15) \* 0b10000;  
 if (func3 == 0b000) {  
 *// C.ADDI4SPN* long nzuimm = ((long) instruction << 9 >>> 15) +  
 ((long) instruction << 10 >>> 15) \* 0b10 +  
 ((long) instruction << 3 >>> 14) \* 0b100 +  
 ((long) instruction << 5 >>> 12) \* 0b10000;  
 if (nzuimm != 0) {  
 return String.*format*("%s %s, %s, %s", "c.addi4spn", getC\_Register(rdc), "sp", nzuimm);  
 }  
 } else if (func3 == 0b010) {  
 *// C.LW* return String.*format*("%s %s, %s, %s", "c.lw", getC\_Register(rdc), getC\_Register(rs1c), uimm);  
 } else if (func3 == 0b110) {  
 *// C.SW* return String.*format*("%s %s, %s, %s", "c.sw", getC\_Register(rs2c), getC\_Register(rs1c), uimm);  
 }  
 } else if (op == 0b01) {  
 if (func3 == 0b000) {  
 *// C.ADDI* int nzimm = (instruction << 9 >>> 11) +  
 (instruction << 3 >>> 15) \* 0b100000;  
 int rd = instruction << 4 >>> 11;  
 if (rd != 0 && nzimm != 0) {  
 return String.*format*("%s %s, %s", "c.addi", getRegister(rd), nzimm);  
 }  
 } else if (func3 == 0b001) {  
 *// C.JAL* int imm = (instruction << 10 >>> 13) +  
 (instruction << 4 >>> 15) \* 0b1000 +  
 (instruction << 13 >>> 15) \* 0b10000 +  
 (instruction << 8 >>> 15) \* 0b100000 +  
 (instruction << 9 >>> 15) \* 0b1000000 +  
 (instruction << 5 >>> 14) \* 0b10000000 +  
 (instruction << 7 >>> 15) \* 0b1000000000 +  
 (instruction << 3 >>> 15) \* 0b10000000000;  
 return String.*format*("%s %s, %s", "c.jal", imm, jumps.get(address));  
 } else if (func3 == 0b010) {  
 *// C.LI* int imm = (instruction << 9 >>> 11) +  
 (instruction << 3 >>> 15) \* 0b100000;  
 int rd = instruction << 4 >>> 11;  
 if (rd != 0) {  
 return String.*format*("%s %s, %s", "c.li", getRegister(rd), imm);  
 }  
 } else if (func3 == 0b011) {  
 int rd = instruction << 4 >>> 11;  
 int nzimm;  
 if (rd == 2) {  
 *// C.ADDI16SP* nzimm = (instruction << 9 >>> 15) +  
 (instruction << 13 >>> 15) \* 0b10 +  
 (instruction << 10 >>> 15) \* 0b100 +  
 (instruction << 11 >>> 14) \* 0b1000 +  
 (instruction << 3 >>> 15) \* 0b100000;  
 if (nzimm != 0) {  
 return String.*format*("%s %s, %s", "c.addi16sp", "sp", nzimm);  
 }  
 } else if (rd != 0){  
 *// C.LUI* nzimm = (instruction << 9 >>> 11) +  
 (instruction << 3 >>> 15) \* 0b1000000;  
 if (nzimm != 0) {  
 return String.*format*("%s %s, %s", "c.lui", getRegister(rd), nzimm);  
 }  
 }  
 } else if (func3 == 0b100) {  
 *// C.SRLI C.SRAI C.ANDI C.SUB C.XOR C.OR C.AND* int func2 = instruction << 4 >>> 14;  
 int func6 = instruction >>> 10;  
 int rdc = instruction << 6 >>> 13;  
 long nzuimm = ((long) instruction << 9 >>> 11) +  
 ((long) instruction << 3 >>> 15) \* 0b100000;  
 int imm = ( instruction << 9 >>> 11) +  
 (instruction << 3 >>> 15) \* 0b100000;  
 if (func2 == 0b00) {  
 *// C.SRLI* if (nzuimm != 0) {  
 return String.*format*("%s %s, %s", "c.srli", getC\_Register(rdc), nzuimm);  
 }  
 } else if (func2 == 0b01) {  
 *// C.SRAI* if (nzuimm != 0) {  
 return String.*format*("%s %s, %s", "c.srai", getC\_Register(rdc), nzuimm);  
 }  
 } else if (func2 == 0b10) {  
 *// C.ANDI* return String.*format*("%s %s, %s", "c.andi", getC\_Register(rdc), imm);  
 } else if (func6 == 0b100011) {  
 int rs2c = instruction << 11 >>> 13;  
 func2 = instruction << 9 >>> 14;  
 if (func2 == 0b00) {  
 *// C.SUB* return String.*format*("%s %s, %s", "c.sub", getC\_Register(rdc), getC\_Register(rs2c));  
 } else if (func2 == 0b01) {  
 *// C.XOR* return String.*format*("%s %s, %s", "c.xor", getC\_Register(rdc), getC\_Register(rs2c));  
 } else if (func2 == 0b10) {  
 *// C.OR* return String.*format*("%s %s, %s", "c.or", getC\_Register(rdc), getC\_Register(rs2c));  
 } else if (func2 == 0b11) {  
 *// C.AND* return String.*format*("%s %s, %s", "c.and", getC\_Register(rdc), getC\_Register(rs2c));  
 }  
 }  
 } else if (func3 == 0b101) {  
 *// C.J* int imm = (instruction << 10 >>> 13) +  
 (instruction << 4 >>> 15) \* 0b1000 +  
 (instruction << 13 >>> 15) \* 0b10000 +  
 (instruction << 8 >>> 15) \* 0b100000 +  
 (instruction << 9 >>> 15) \* 0b1000000 +  
 (instruction << 5 >>> 14) \* 0b10000000 +  
 (instruction << 7 >>> 15) \* 0b1000000000 +  
 (instruction << 3 >>> 15) \* 0b10000000000;  
 return String.*format*("%s %s, %s", "c.j", imm, jumps.get(address));  
 } else if (func3 == 0b110) {  
 *// C.BEQZ* int rs1c = instruction << 6 >>> 13;  
 int imm = (instruction << 11 >>> 14) +  
 (instruction << 4 >>> 14) \* 0b100 +  
 (instruction << 13 >>> 15) \* 0b10000 +  
 (instruction << 9 >>> 14) \* 0b100000 +  
 (instruction << 3 >>> 15) \* 0b10000000;  
 return String.*format*("%s %s, %s, %s", "c.beqz", getC\_Register(rs1c), imm, jumps.get(address));  
 } else if (func3 == 0b111) {  
 *// C.BNEZ* int rs1c = instruction << 6 >>> 13;  
 int imm = (instruction << 11 >>> 14) +  
 (instruction << 4 >>> 14) \* 0b100 +  
 (instruction << 13 >>> 15) \* 0b10000 +  
 (instruction << 9 >>> 14) \* 0b100000 +  
 (instruction << 3 >>> 15) \* 0b10000000;  
 return String.*format*("%s %s, %s, %s", "c.bnez", getC\_Register(rs1c), imm, jumps.get(address));  
 }  
 } else if (op == 0b10) {  
 int rd = instruction << 4 >>> 11;  
 int rs1 = rd;  
 if (func3 == 0b000) {  
 *// C.SLLI* long nzuimm = ((long) instruction << 9 >>> 11) +  
 ((long) instruction << 3 >>> 15) \* 0b100000;  
 if (nzuimm != 0 && rd != 0) {  
 return String.*format*("%s %s, %s", "c.slli", getRegister(rd), nzuimm);  
 }  
 } else if (func3 == 0b010) {  
 *// C.LWSP* long uimm = ((long) instruction << 9 >>> 13) +  
 ((long) instruction << 3 >>> 15) \* 0b1000 +  
 ((long) instruction << 12 >>> 14) \* 0b10000;  
 if (rd != 0) {  
 return String.*format*("%s %s, %s", "c.lwsp", getRegister(rd), uimm);  
 }  
 } else if (func3 == 0b100) {  
 *// C.JR C.MV C.EBREAK C.JALR C.ADD* int rs2 = instruction << 9 >>> 11;  
 int func4 = instruction >>> 12;  
 if (func4 == 0b1000 && rd != 0) {  
 if (rs2 == 0) {  
 *// C.JR* return String.*format*("%s %s", "c.jr", getRegister(rs1));  
 } else {  
 *// C.MV* return String.*format*("%s %s, %s", "c.mv", getRegister(rd), getRegister(rs2));  
 }  
 } else if (func4 == 0b1001) {  
 if (rs2 == 0) {  
 if (rs1 == 0) {  
 *// C.EBREAK* return "c.ebreak";  
 } else {  
 *// C.JALR* return String.*format*("%s %s", "c.jalr", getRegister(rs1));  
 }  
 } else if (rd != 0) {  
 *// C.ADD* return String.*format*("%s %s, %s", "c.add", getRegister(rd), getRegister(rs2));  
 }  
 }  
 } else if (func3 == 0b110) {  
 *// C.SWSP* int rs2 = instruction << 9 >>> 11;  
 long uimm = ((long) instruction << 3 >>> 12) +  
 ((long) instruction << 7 >>> 14) \* 0b10000;  
 return String.*format*("%s %s, %s", "c.swsp", getRegister(rs2), uimm);  
 }  
 }  
 return "unknown\_command";  
 }  
  
 private String get32bitInstruction(int instruction) {  
 int opcode = instruction << 25 >>> 25;  
 if (opcode == 0b0110111) {  
 *// LUI* int rd = instruction << 20 >>> 27;  
 int imm = instruction >>> 12;  
 return String.*format*("%s %s, %s", "lui", getRegister(rd), imm);  
 } else if (opcode == 0b0010111) {  
 *// AUIPC* int rd = instruction << 20 >>> 27;  
 int imm = instruction >>> 12;  
 return String.*format*("%s %s, %s", "auipc", getRegister(rd), imm);  
 } else if (opcode == 0b1101111) {  
 *// JAL* int rd = instruction << 20 >>> 27;  
 int imm = (instruction << 1 >>> 22) +  
 (instruction << 11 >>> 31) \* 0b10000000000 +  
 (instruction << 12 >>> 24) \* 0b100000000000 +  
 (instruction >>> 31) \* 0b10000000000000000000;  
 return String.*format*("%s %s, %s, %s", "jal", getRegister(rd), imm, jumps.get(address));  
 } else if (opcode == 0b1100111) {  
 *// JALR* int rd = instruction << 20 >>> 27;  
 int rs1 = instruction << 12 >>> 27;  
 int imm = instruction >>> 20;  
 return String.*format*("%s %s, %s", "jalr", getRegister(rd), getRegister(rs1), imm);  
 } else if (opcode == 0b1100011) {  
 *// BEQ BNE BLT BGE BLTU BGEU* int rs1 = instruction << 12 >>> 27;  
 int rs2 = instruction << 7 >>> 27;  
 int imm = (instruction << 20 >>> 28) +  
 (instruction << 1 >>> 26) \* 0b10000 +  
 (instruction << 24 >>> 31) \* 0b10000000000 +  
 (instruction >>> 31) \* 0b100000000000;  
 int func3 = instruction << 17 >>> 29;  
 return switch(func3) {  
 case 0b000 -> String.*format*("%s %s, %s, %s, %s", "beq", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BEQ* case 0b001 -> String.*format*("%s %s, %s, %s, %s", "bne", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BNE* case 0b100 -> String.*format*("%s %s, %s, %s, %s", "blt", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BLT* case 0b101 -> String.*format*("%s %s, %s, %s, %s", "bge", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BGE* case 0b110 -> String.*format*("%s %s, %s, %s, %s", "bltu", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BLTU* case 0b111 -> String.*format*("%s %s, %s, %s, %s", "bgeu", getRegister(rs1), getRegister(rs2), imm, jumps.get(address)); *// BGEU* default -> "unknown\_command";  
 };  
 } else if (opcode == 0b0000011) {  
 *// LB LH LW LBU LHU* int rd = instruction << 20 >>> 27;  
 int rs1 = instruction << 12 >>> 27;  
 int imm = instruction >>> 20;  
 int func3 = instruction << 17 >>> 29;  
 return switch(func3) {  
 case 0b000 -> String.*format*("%s %s, %s(%s)", "lb", getRegister(rd), imm, getRegister(rs1)); *// LB* case 0b001 -> String.*format*("%s %s, %s(%s)", "lh", getRegister(rd), imm, getRegister(rs1)); *// LH* case 0b010 -> String.*format*("%s %s, %s(%s)", "lw", getRegister(rd), imm, getRegister(rs1)); *// LW* case 0b100 -> String.*format*("%s %s, %s(%s)", "lbu", getRegister(rd), imm, getRegister(rs1)); *// LBU* case 0b101 -> String.*format*("%s %s, %s(%s)", "lhu", getRegister(rd), imm, getRegister(rs1)); *// LHU* default -> "unknown\_command";  
 };  
 } else if (opcode == 0b0100011) {  
 *// SB SH SW* int rs1 = instruction << 12 >>> 27;  
 int rs2 = instruction << 7 >>> 27;  
 int imm = (instruction << 20 >>> 27) +  
 (instruction >>> 25) \* 0b100000;  
 int func3 = instruction << 17 >>> 29;  
 return switch(func3) {  
 case 0b000 -> String.*format*("%s %s, %s(%s)", "sb", getRegister(rs2), imm, getRegister(rs1)); *// SB* case 0b001 -> String.*format*("%s %s, %s(%s)", "sh", getRegister(rs2), imm, getRegister(rs1)); *// SH* case 0b010 -> String.*format*("%s %s, %s(%s)", "sw", getRegister(rs2), imm, getRegister(rs1)); *// SW* default -> "unknown\_command";  
 };  
 } else if (opcode == 0b0010011) {  
 *// ADDI SLTI SLTIU XORI ORI ANDI SLLI SRLI SRAI* int rd = instruction << 20 >>> 27;  
 int rs1 = instruction << 12 >>> 27;  
 int func3 = instruction << 17 >>> 29;  
 int imm = instruction >>> 20;  
 int shamt = instruction << 7 >>> 27;  
 int func7 = instruction >>> 25;  
 if (func3 == 0b000) {  
 *// ADDI* return String.*format*("%s %s, %s, %s", "addi", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b010) {  
 *// SLTI* return String.*format*("%s %s, %s, %s", "slti", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b011) {  
 *// SLTIU* return String.*format*("%s %s, %s, %s", "sltiu", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b100) {  
 *// XORI* return String.*format*("%s %s, %s, %s", "xori", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b110) {  
 *// ORI* return String.*format*("%s %s, %s, %s", "ori", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b111) {  
 *// ANDI* return String.*format*("%s %s, %s, %s", "andi", getRegister(rd), getRegister(rs1), imm);  
 } else if (func3 == 0b001 && func7 == 0b0000000) {  
 *// SLLI* return String.*format*("%s %s, %s, %s", "slli", getRegister(rd), getRegister(rs1), shamt);  
 } else if (func3 == 0b101) {  
 if (func7 == 0b0000000) {  
 *// SRLI* return String.*format*("%s %s, %s, %s", "srli", getRegister(rd), getRegister(rs1), shamt);  
 } else if (func7 == 0b0100000) {  
 *// SRAI* return String.*format*("%s %s, %s, %s", "srai", getRegister(rd), getRegister(rs1), shamt);  
 }  
 }  
 } else if (opcode == 0b0110011) {  
 *// ADD SUB SLL SLT SLTU XOR SRL SRA OR AND  
 // MUL MULH MULHSU MULHU DIV DIVU REM REMU* int rd = instruction << 20 >>> 27;  
 int rs1 = instruction << 12 >>> 27;  
 int rs2 = instruction << 7 >>> 27;  
 int func3 = instruction << 17 >>> 29;  
 int func7 = instruction >>> 25;  
 if (func3 == 0b000) {  
 if (func7 == 0b0000000) {  
 *// ADD* return String.*format*("%s %s, %s, %s", "add", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0100000) {  
 *// SUB* return String.*format*("%s %s, %s, %s", "sub", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// MUL* return String.*format*("%s %s, %s, %s", "mul", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b001) {  
 if (func7 == 0b0000000) {  
 *// SLL* return String.*format*("%s %s, %s, %s", "sll", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// MULH* return String.*format*("%s %s, %s, %s", "mulh", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b010) {  
 if (func7 == 0b0000000) {  
 *// SLT* return String.*format*("%s %s, %s, %s", "slt", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// MULHSU* return String.*format*("%s %s, %s, %s", "mulhsu", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b011) {  
 if (func7 == 0b0000000) {  
 *// SLTU* return String.*format*("%s %s, %s, %s", "sltu", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// MULHU* return String.*format*("%s %s, %s, %s", "mulhu", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b100) {  
 if (func7 == 0b0000000) {  
 *// XOR* return String.*format*("%s %s, %s, %s", "xor", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// DIV* return String.*format*("%s %s, %s, %s", "div", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b101) {  
 if (func7 == 0b0000000) {  
 *// SRL* return String.*format*("%s %s, %s, %s", "srl", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0100000) {  
 *// SRA* return String.*format*("%s %s, %s, %s", "sra", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// DIVU* return String.*format*("%s %s, %s, %s", "divu", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b110) {  
 if (func7 == 0b0000000) {  
 *// OR* return String.*format*("%s %s, %s, %s", "or", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// REM* return String.*format*("%s %s, %s, %s", "rem", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 } else if (func3 == 0b111) {  
 if (func7 == 0b0000000) {  
 *// AND* return String.*format*("%s %s, %s, %s", "and", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 } else if (func7 == 0b0000001) {  
 *// REMU* return String.*format*("%s %s, %s, %s", "remu", getRegister(rd), getRegister(rs1), getRegister(rs2));  
 }  
 }  
 } else if (opcode == 0b0001111) {  
 *// skip FENCE FENCE.I* } else if (opcode == 0b1110011) {  
 *// ECALL EBREAK CSRRW CSRRS CSRRC CSRRWI CSRRSI CSRRCI* int func3 = instruction << 17 >>> 29;  
 int rd = instruction << 20 >>> 27;  
 int rs1 = instruction << 12 >>> 27;  
 int func12 = instruction >>> 20;  
 long uimm = (long) instruction << 12 >>> 27;  
 int csr = func12;  
 if (func3 == 0b000) {  
 if (rd == 0b00000 && rs1 == 0b00000) {  
 if (func12 == 0b000000000000) {  
 *// ECALL* return "ecall";  
 } else if (func12 == 0b000000000001) {  
 *// EBREAK* return "ebreak";  
 }  
 }  
 } else {  
 return switch (func3) {  
 case 0b001 -> String.*format*("%s %s, %s, %s", "csrrw", getRegister(rd), csr, getRegister(rs1)); *// CSRRW* case 0b010 -> String.*format*("%s %s, %s, %s", "csrrs", getRegister(rd), csr, getRegister(rs1)); *// CSRRS* case 0b011 -> String.*format*("%s %s, %s, %s", "csrrc", getRegister(rd), csr, getRegister(rs1)); *// CSRRC* case 0b101 -> String.*format*("%s %s, %s, %s", "csrrwi", getRegister(rd), csr, uimm); *// CSRRWI* case 0b110 -> String.*format*("%s %s, %s, %s", "csrrsi", getRegister(rd), csr, uimm); *// CSRRSI* case 0b111 -> String.*format*("%s %s, %s, %s", "csrrci", getRegister(rd), csr, uimm); *// CSRRCI* default -> "unknown\_command";  
 };  
 }  
 }  
 return "unknown\_command";  
 }  
  
 private int read16bit() throws IOException {  
 return reader.read() + reader.read() \* 0x100;  
 }  
  
 private int read32bit() throws IOException {  
 return reader.read() + reader.read() \* 0x100 + reader.read() \* 0x10000 + reader.read() \* 0x1000000;  
 }  
  
 private String getRegister(int reg) {  
 if (reg < 0 || reg > *REGISTERS*.length) {  
 throw new AssertionError("Register " + reg + " does not exist");  
 }  
 return *REGISTERS*[reg];  
 }  
  
 private String getC\_Register(int regc) {  
 return getRegister(8 + regc);  
 }  
}

**ElfFile.java**

import java.io.\*;  
  
public class ElfFile {  
 public final File source;  
 public ElfHeader elfHeader;  
 public SectionHeader shstrtabHeader;  
 public SectionHeader textHeader;  
 public SymbolTable symbolTable;  
 public SectionHeader symtabHeader;  
 public SectionHeader strtabHeader;  
 public StringTable stringTable;  
  
 public ElfFile(String filename) throws IOException {  
 source = new File(filename);  
 readElfHeader();  
 readShstrtabHeader();  
 readTextHeader();  
 readStrtabHeader();  
 readSymbolTable();  
 }  
  
 private void readStrtabHeader() throws IOException {  
 *// Finding .strtab string in .shstrtab* StringTable shstrTable = new StringTable(source);  
 shstrTable.reader.skip(shstrtabHeader.sh\_offset);  
 shstrTable.read();  
 while (!shstrTable.string.equals(".strtab")) {  
 shstrTable.read();  
 }  
 *// Setting offset on the beginning of .strtab* shstrTable.offset -= 8;  
 *// Finding and reading .strtab Header* strtabHeader = new SectionHeader(source);  
 strtabHeader.reader.skip(elfHeader.e\_shoff);  
 while (strtabHeader.sh\_name != shstrTable.offset) {  
 strtabHeader.read();  
 }  
 }  
  
 private void readSymbolTable() throws IOException {  
 *// Finding and reading .symtab Header* symtabHeader = new SectionHeader(source);  
 symtabHeader.reader.skip(elfHeader.e\_shoff);  
 while (symtabHeader.sh\_type != 0x2) {  
 symtabHeader.read();  
 }  
 *// Reading .symtab* symbolTable = new SymbolTable(source, symtabHeader.sh\_size / symtabHeader.sh\_entsize);  
 symbolTable.reader.skip(symtabHeader.sh\_offset);  
 symbolTable.read();  
 *// Reading names of labels* for (int i = 0; i < symbolTable.table.size(); i++) {  
 stringTable = new StringTable(source);  
 stringTable.reader.skip(strtabHeader.sh\_offset + symbolTable.table.get(i).st\_name);  
 stringTable.read();  
 symbolTable.table.get(i).name = stringTable.string;  
 }  
 }  
  
 private void readTextHeader() throws IOException {  
 *// Finding .text string in .shstrtab* StringTable shstrTable = new StringTable(source);  
 shstrTable.reader.skip(shstrtabHeader.sh\_offset);  
 shstrTable.read();  
 while (!shstrTable.string.equals(".text")) {  
 shstrTable.read();  
 }  
 *// Setting offset on the beginning of .text* shstrTable.offset -= 6;  
 *// Finding and reading .text Header* textHeader = new SectionHeader(source);  
 textHeader.reader.skip(elfHeader.e\_shoff);  
 while (textHeader.sh\_name != shstrTable.offset) {  
 textHeader.read();  
 }  
 }  
  
 private void readShstrtabHeader() throws IOException {  
 *// Reading .shstrtab Header* shstrtabHeader = new SectionHeader(source);  
 shstrtabHeader.reader.skip(elfHeader.e\_shoff + (long) elfHeader.e\_shentsize \* elfHeader.e\_shstrndx);  
 shstrtabHeader.read();  
 }  
  
 private void readElfHeader() throws IOException {  
 *// Reading ELF Header* elfHeader = new ElfHeader(source);  
 elfHeader.read();  
 *// Checking file correctness* elfHeader.check();  
 }  
  
 public void closeAll() throws IOException {  
 elfHeader.reader.close();  
 shstrtabHeader.reader.close();  
 textHeader.reader.close();  
 symbolTable.reader.close();  
 symtabHeader.reader.close();  
 strtabHeader.reader.close();  
 stringTable.reader.close();  
 }  
}

**Element.java**

import java.io.\*;  
  
public abstract class Element {  
 public final File file;  
 public final BufferedInputStream reader;  
  
 public Element(File file) throws IOException {  
 this.file = file;  
 this.reader = new BufferedInputStream(new FileInputStream(file));  
 }  
  
 abstract void read() throws IOException;  
  
 abstract void dump();  
  
 protected int readByte() throws IOException {  
 return reader.read();  
 }  
  
 protected int readHalfWord() throws IOException {  
 return reader.read() + reader.read() \* 0x100;  
 }  
  
 protected int readWord() throws IOException {  
 return reader.read() + reader.read() \* 0x100 + reader.read() \* 0x10000 + reader.read() \* 0x1000000;  
 }  
}

**ElfHeader.java**

import java.io.File;  
import java.io.IOException;  
  
public class ElfHeader extends Element {  
 public int [] e\_ident;  
 public int e\_type;  
 public int e\_machine;  
 public int e\_version;  
 public int e\_entry;  
 public int e\_phoff;  
 public int e\_shoff;  
 public int e\_flags;  
 public int e\_ehsize;  
 public int e\_phentsize;  
 public int e\_phnum;  
 public int e\_shentsize;  
 public int e\_shnum;  
 public int e\_shstrndx;  
  
 public ElfHeader(File file) throws IOException {  
 super(file);  
 }  
  
 @Override  
 public void read() throws IOException {  
 e\_ident = new int[16];  
 for (int i = 0; i < 16; i++) {  
 e\_ident[i] = readByte();  
 }  
 e\_type = readHalfWord();  
 e\_machine = readHalfWord();  
 e\_version = readWord();  
 e\_entry = readWord();  
 e\_phoff = readWord();  
 e\_shoff = readWord();  
 e\_flags = readWord();  
 e\_ehsize = readHalfWord();  
 e\_phentsize = readHalfWord();  
 e\_phnum = readHalfWord();  
 e\_shentsize = readHalfWord();  
 e\_shnum = readHalfWord();  
 e\_shstrndx = readHalfWord();  
 }  
  
 public void check() throws IOException {  
 if (e\_ident[0] != 0x7f || e\_ident[1] != 0x45 || e\_ident[2] != 0x4c || e\_ident[3] != 0x46) {  
 throw new IOException("Illegal file");  
 }  
 if (e\_ident[4] != 0x01) {  
 throw new IOException("File is not 32 bit");  
 }  
 if (e\_ident[5] != 0x01) {  
 throw new IOException("File is not little-endian");  
 }  
 if (e\_machine != 0xF3) {  
 throw new IOException("File architecture is not RISC-V");  
 }  
 }  
  
 @Override  
 public void dump() {  
 System.*out*.printf("%-12s ", "e\_ident");  
 for (int i = 0; i < 16; i++) {  
 System.*out*.printf("%02x ", e\_ident[i]);  
 }  
 System.*out*.println();  
 System.*out*.printf("%-12s %02x %n", "e\_type", e\_type);  
 System.*out*.printf("%-12s %02x %n", "e\_machine", e\_machine);  
 System.*out*.printf("%-12s %04x %n", "e\_version", e\_version);  
 System.*out*.printf("%-12s %04x %n", "e\_entry", e\_entry);  
 System.*out*.printf("%-12s %04x %n", "e\_phoff", e\_phoff);  
 System.*out*.printf("%-12s %04x %n", "e\_shoff", e\_shoff);  
 System.*out*.printf("%-12s %04x %n", "e\_flags", e\_flags);  
 System.*out*.printf("%-12s %02x %n", "e\_ehsize", e\_ehsize);  
 System.*out*.printf("%-12s %02x %n", "e\_phentsize", e\_phentsize);  
 System.*out*.printf("%-12s %02x %n", "e\_phnum", e\_phnum);  
 System.*out*.printf("%-12s %02x %n", "e\_shentsize", e\_shentsize);  
 System.*out*.printf("%-12s %02x %n", "e\_shnum", e\_shnum);  
 System.*out*.printf("%-12s %02x %n", "e\_shstrndx", e\_shstrndx);  
 System.*out*.println();  
 }  
}

**SectionHeader.java**

import java.io.File;  
import java.io.IOException;  
  
public class SectionHeader extends Element {  
 public int sh\_name;  
 public int sh\_type;  
 public int sh\_flags;  
 public int sh\_addr;  
 public int sh\_offset;  
 public int sh\_size;  
 public int sh\_link;  
 public int sh\_info;  
 public int sh\_addralign;  
 public int sh\_entsize;  
  
 public SectionHeader(File file) throws IOException {  
 super(file);  
 }  
  
 @Override  
 public void read() throws IOException {  
 sh\_name = readWord();  
 sh\_type = readWord();  
 sh\_flags = readWord();  
 sh\_addr = readWord();  
 sh\_offset = readWord();  
 sh\_size = readWord();  
 sh\_link = readWord();  
 sh\_info = readWord();  
 sh\_addralign = readWord();  
 sh\_entsize = readWord();  
 }  
  
 @Override  
 public void dump() {  
 System.*out*.printf("%-12s %08x %n", "sh\_name", sh\_name);  
 System.*out*.printf("%-12s %08x %n", "sh\_type", sh\_type);  
 System.*out*.printf("%-12s %08x %n", "sh\_flags", sh\_flags);  
 System.*out*.printf("%-12s %08x %n", "sh\_addr", sh\_addr);  
 System.*out*.printf("%-12s %08x %n", "sh\_offset", sh\_offset);  
 System.*out*.printf("%-12s %08x %n", "sh\_size", sh\_size);  
 System.*out*.printf("%-12s %08x %n", "sh\_link", sh\_link);  
 System.*out*.printf("%-12s %08x %n", "sh\_info", sh\_info);  
 System.*out*.printf("%-12s %08x %n", "sh\_addralign", sh\_addralign);  
 System.*out*.printf("%-12s %08x %n", "sh\_entsize", sh\_entsize);  
 System.*out*.println();  
 }  
}

**StringTable.java**

import java.io.File;  
import java.io.IOException;  
  
public class StringTable extends Element {  
 public String string;  
 public int offset;  
  
 public StringTable(File file) throws IOException {  
 super(file);  
 }  
  
 @Override  
 void read() throws IOException {  
 StringBuilder sb = new StringBuilder();  
 int cur = readByte();  
 offset++;  
 while (cur != 0x0) {  
 sb.append((char) (cur));  
 cur = readByte();  
 offset++;  
 }  
 string = sb.toString();  
 }  
  
 @Override  
 void dump() {  
 System.*out*.println(string);  
 System.*out*.println();  
 }  
}

**SymbolTable.java**

import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class SymbolTable extends Element {  
 public List<SymbolTableEntry> table;  
 public final int numberOfEntries;  
  
 public SymbolTable(File file, int numberOfEntries) throws IOException {  
 super(file);  
 table = new ArrayList<>();  
 this.numberOfEntries = numberOfEntries;  
 }  
  
 @Override  
 void read() throws IOException {  
 for (int i = 0; i < numberOfEntries; i++) {  
 table.add(readEntry());  
 }  
 }  
  
 private SymbolTableEntry readEntry() throws IOException {  
 SymbolTableEntry entry = new SymbolTableEntry();  
 entry.st\_name = readWord();  
 entry.st\_value = readWord();  
 entry.st\_size = readWord();  
 entry.st\_info = readByte();  
 entry.st\_other = readByte();  
 entry.st\_shndx = readHalfWord();  
 entry.st\_type = entry.st\_info & 0xf;  
 entry.st\_vis = entry.st\_other & 0x3;  
 entry.st\_bind = entry.st\_info >>> 4;  
 return entry;  
 }  
  
 @Override  
 void dump() {  
 for (int i = 0; i < numberOfEntries; i++) {  
 System.*out*.printf("%s %d %n", "Entry ", i + 1);  
 System.*out*.printf("%-8s %08x %n", "st\_name", table.get(i).st\_name);  
 System.*out*.printf("%-8s %08x %n", "st\_value", table.get(i).st\_value);  
 System.*out*.printf("%-8s %08x %n", "st\_size", table.get(i).st\_size);  
 System.*out*.printf("%-8s %02x %n", "st\_info", table.get(i).st\_info);  
 System.*out*.printf("%-8s %02x %n", "st\_other", table.get(i).st\_other);  
 System.*out*.printf("%-8s %04x %n", "st\_shndx", table.get(i).st\_shndx);  
 System.*out*.printf("%-8s %04x %n", "st\_type", table.get(i).st\_type);  
 System.*out*.printf("%-8s %04x %n", "St\_vis", table.get(i).st\_vis);  
 System.*out*.printf("%-8s %04x %n", "st\_bind", table.get(i).st\_bind);  
 System.*out*.println();  
 }  
 }  
}

**SymbolTableEntry.java**

public class SymbolTableEntry {  
 public int st\_name;  
 public int st\_value;  
 public int st\_size;  
 public int st\_info;  
 public int st\_other;  
 public int st\_shndx;  
 public int st\_type;  
 public int st\_vis;  
 public int st\_bind;  
 public String name;  
}