Лабораторная работа №3	Б10	2022
ISA	Пешков Артём Вадимович	

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

Инструментарий и требования к работе: работа выполнялась на языке C++, компилятор microsoft visual studio, version (_MSC_VER) = 1934

Описание

Необходимо написать программу-транслятор (дизассемблер), с помощью которой можно преобразовывать машинный код в текст программы на языке ассемблера.

Должен поддерживаться следующий набор команд: RISC-V RV32I, RV32M. Подробнее (volume 1): https://riscv.org/technical/specifications/

Кодирование: little endian.

Обрабатывать нужно только секции .text, .symtable.

Для каждой строки кода указывается её адрес в hex формате.

Обозначение меток нужно найти в Symbol Table (.symtab). Если же название метки там не найдено, то используется следующее обозначение: L%i, например, L2, L34. Нумерация начинается с 0.

Вариант

Без варианта.

Описание системы кодирования RISC-V

RISC-V как ISA

RISC-V это ISA типа reg-reg, так как большинство команд работают только с регистрам, кроме load и store инструкций, которые загружают данные в регистры из памяти и выгружают их из регистров в память. Также RISC-V является модульной ISA, состоящей из нескольких базовых ISA и нескольких ISA-надстроек (в нашей лабораторной мы работаем с базовой ISA RV32I и расширением RV32M). Данная ISA описывает интерфейс исполнителя, который на практике может быть ОС или настоящим железом.

Регистры

В RISC-V есть 31 регистр общего назначения (x1-x31), которые хранят в себе целочисленные значения, регистр x0 - который хранит в себе 0 и дополнительный регистр program counter (pc), который хранит в себе адрес текущей инструкции. Для RV32 ширина регистра 32 бита (это поле называется XLEN). Более удобные и понятные названия и значения регистров можно посмотреть на рисунке 1.

Register	ABI Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6–7	t1–2	Temporaries
x8	s0/fp	Saved register/frame pointer
x9	s1	Saved register
x10–11	a0–1	Function arguments/return values
x12–17	a2–7	Function arguments
x18–27	s2–11	Saved registers
x28–31	t3–6	Temporaries

Рисунок №1 - названия регистров RISC-V и их назначения

Осталось поговорить о командах. В рамках данное лабораторной работы нам нужны только команды RV32I и RV32M, поговорим про них.

Команды

Команды делятся на несколько типов (см римунок 2):

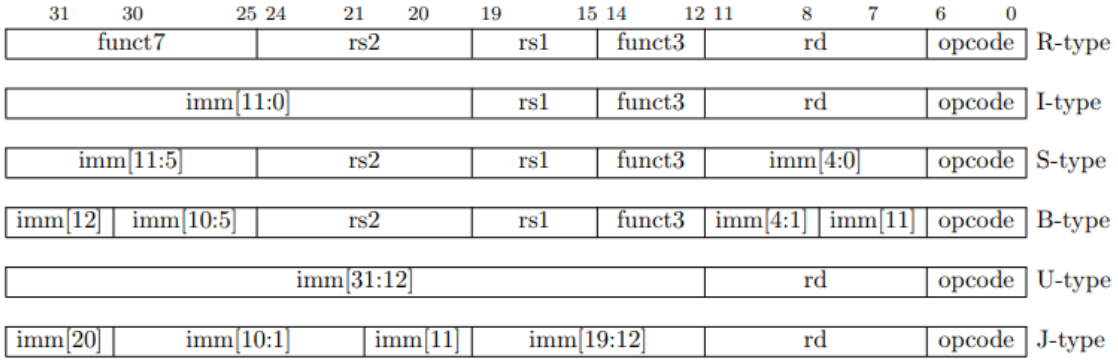

Рисунок № 2 - типы команд

Команда LUI (load upper immediate) помещает константу в регистр rd. Старшие 20 бит константы определены, младшие 12 равны 0. (U-type)

Команда AUPIC (add upper immediate to pc) берёт 20-битное смещение, дописывает в младшие биты 12 нулей, прибавляет к адресу AUPIC инструкции и записывает в регистр rd. (U-type)

Команда JAL (jump and link) берёт смещение и прибавляет его к адресу jump инструкции, чтобы сформировать адрес перехода, а в rd сохраняется адрес следующей за JAL инструкции. (J-type)

Команда JALR аналогична JAL, просто берёт значение адреса к которому прибавить из rs1, чтобы можно было перемещаться в более отдалённые части. (I-type)

Команды BRANCH (beq (eq = equal), bne (ne = not equal), blt (lt = less than), bge, bltu, bgeu) берут смещение, прибавляя его к адресу BRANCH инструкции и получают целевой адрес. Команда также принимает 2 регистра и в зависимости от результата сравнения (желаемый результат сравнения зашит в названии команды, также оно бывает знаковым или беззнаковым) совершает переход или нет. (B-type)

Команды LOAD/STORE взаимодействуют непосредственно с памятью и делать это могут только они. Load и Store инструкции берут значение по адресу, который хранится в rs1, прибавляют к нему offset и получают целевой адрес в памяти, Load записывает результат в rd, Store записывает данные rs2 в память.

Название всех Load инструкций начинается с l, вторая и третья(если есть) буквы означают количество бит, с которым мы хотим поработать (b - 8 bits, h - 16 bits, w - 32 bits, если в конце есть буква H, происходит заполнение полученных данных нулями до 32 бит). Load инструкции имеют I-type.

Store инструкции работают по тем же правилам, первая буква - s, вторая отвечает за количество считываемых данных. Эти инструкции относятся к S-type.

Integer Register-Immediate Instructions нужны чтобы совершать операции суммы (addi), сравнения (знакового и беззнакового) (stli, stliu), логические операции (xori, ori, andi) между регистром и 12-битной константой, они помещают результат в регистр rd. Также к этим инструкциям относятся команды сдвига (slli, srli, srai) которые осуществляют сдвиг значения из rs1 влево или вправо на shamt (5-ти битная константа). SRLI и SLLI это логический сдвиг, а SRAI это арифметический сдвиг. (I-type)

Integer Register-Register Operations это такие команды как: add и sub (+ и -), slt and sltu (знаковое и беззнаковое сравненение), and , or, xor, sll, srl и sra (левый правый логический и арифметический сдвиги) операции над двумя регистрами(rs1 и rs2), результат которых записывается в rd. (R-type)

Остались только команды RV32M: mul (произведение 2-х XLEN битных чисел, возвращающее младшие XLEN бит результата), mulh mulhu и mulhsu (произведение 2-х XLEN битных чисел, возвращающее старшие XLEN бит результата, причём в зависимости от последних букв произведение может быть знаковым, беззнаковым или считать знаковым только rs1), div и divu (целочисленное деление 2-х XLEN битных чисел, знаковое и беззнаковое), rem и remu (остаток от деление 2-х XLEN битных чисел, знаковое и беззнаковое).

Описание структуры файла ELF

Я буду описывать разделы файла в том порядке, в котором они были нужны мне для парсинга.

Заголовок файла (ELF header)

Заголовок находится в самом начале файла и занимает 52 байта, если у нас 32-битный файл.

Первые 16 байт заголовка - это раздел e_ident, для проверки корректности полученного файла из него нам нужны:
байты с 0 по 3 (EI_MAG0 - EI_MAG3) (магическая константа, которая подтверждает, что мы работаем именно с elf файлом `0x7f 0x45 0x4c 0x46`)
4 байт (EI_CLASS), отвечающий за битность файла (нужно, чтобы он равнялся 1)
5 байт (EI_DATA), отвечающий за метод кодирования (в нашем случае little endian, он тоже должен быть равен 1)
6 байт (EI_VERSION), подтверждающий что версия elf заголовка актуальная, этот байт всегда должен быть равен 1
18-19 байты (e_machine), хранящие архитектуру аппаратной платформы для которой создан файл, в нашем случае эти 2 байта должны равняться 0xF3, так как у нас архитектура RISC-V
20-23 байты (e_version) хранящие номер версии формата, корректным считается только когда это значение равно 1.

Всё что касается проверок корректности мы посмотрели, дальше будем доставать информацию необходимую для дальнейшего парсинга файла. В оставшихся байтах заголовка хранится много различных данных, но нам пригодится только следующее:
4 байта e_shoff по адресу 0x20 (смещение таблицы заголовков секций от начала файла в байтах)
2 байта e_shentsize по адресу 0x2E (размер одного заголовка секции)
2 байта e_shnum по адресу 0x30 (число заголовков секций)
2 байта e_shstrndx по адресу 0x32 (индекс записи в таблице заголовков

секций, описывающей таблицу названий секций, таблица названий секций - секция .shstrtab). Далее будем разбираться с таблицей заголовков секций.

Таблица заголовков секций.

Заголовок каждой секции имеет размер 40 байт, пойдём по порядку.

sh_name (0x00) - смещение в таблице .shstrtab, с помощью которого можно определить заголовок какой именно секции мы смотрим.

sh_addr (0x0C) - если секция загружается в оперативную помаять, то по этому полю стоит ее виртуальный адрес там

sh_offset (0x10) - смещение секции от начала файла в байтах. Когда мы хотим прочитать данные интересующей нас секции мы смещаемся именно с помощью этого значения.

sh_size (0x14) - размер секции в elf файле, это значение используется, чтобы знать сколько байт считывать от начала секции.

Остальные данные нам не нужны для разбора файла, дальше посмотрим непосредственно на сами секции.

.strtab

Секция, в которой хранитсяа таблица имён символов из секции .symtab. В самой секции .symtab хранятся индекс в этой таблице вместо имени, поэтому без неё .symtab вывести не получится.

.symtab

Эта секция содержит в себе описание всех символов и функций программы, а также файлов в которых они были описаны. Она состоит из блоков длины 16 байт:

st_name (4 байта) - индекс в таблице .strtab.

st_value (4 байта) - хранит значение соответсвующего символа, в зависимости от контекста может быть абсолютным значением или адресом

st_size (4 байт) - для некоторых символов существует размер, как-то к ним относящийся. Если размер равен 0, то он либо действительно 0, либо размер неизвестен.

st_info (1 байт) - хранит в себе сразу 2 значения: старшие 4 бита отвечают за bind, младшие 4 бита за type.

st_other (1 байт) - задаёт видимость символа

st_shndx (2 байта) - каждая запись в таблице строк определена по отношению к какой-то секции. В этом поле либо хранится индекс в таблице заголовков секций, либо специальный индекс, имеющий особое значение.

Числам полученным из таблицы мы сопоставляем текстовые значения, основываясь на документации. Например: если младшие биты st_info равны 0010, это означает что type = 2, символ имеет тип STT_FUNC, то есть он связан с функцией или другим запускаемым кодом. Для нас это особенно важный случай, так как все символы с типом STT_FUNC нам нужно запомнить для проставления меток, ведь это вполне понятный для нас адрес.

.text

В этой секции лежат инструкции, которые нам необходимо дизассемблировать. В этой секции в отличие от остальных нам пригодится sh_addr из заголовка, чтобы выводить адреса наших инструкций и рассчитывать адреса для некоторых команд.

Описание работы написанного кода

Чтение данных

Проверки

Программа получает на вход elf файл и название файла, в который необходимо вывести результат, после чего начинает считывать данные elf файла. В начале считываются данные заголовка, необходимые для проверки корректности (подробно расписано при описании структуры elf файла) и если с файлом что-то не так, пользователю выведется сообщение об ошибке.

Дочитываем заголовок и считываем заголовки секций

После того, как мы проверили что с файлом всё хорошо, мы считываем e_shoff, e_shentsize, e_shnum и e_shstrndx. С помощью смещения на e_shoff + e_shentsize * e_shstrndx (начало таблицы заголовков + смещение к записи о секции .shstrtab) мы находим и считываем заголовок секции .shstrtab (далее в этой части отчёта я буду называть заголовок этой секции shstrtab_header).

Немного про считывание заголовков секций: в описании структуры ELF файла я описал только те значения заголовков, которые непосредственно буду использовать, при этом чтобы лишний раз не двигать указатель по файлу, я считываю заголовок полностью в специальную структуру.

Далее запускается цикл на e_shnum итераций в котором мы на каждом шаге ставим указатель на начало очередного заголовка секции, считываем заголовок, после чего смещаемся на начало секции .shstrtab с помощью sh_offset из shstrtab_header и оттуда ещё раз смещаемся на sh_name только что считанного заголовка, откуда читаем символы до первого \0 и узнать название секции, заголовок которой мы сейчас считали.

В рамках этой лабораторной работы нам интересны 3 секции: .text, .strtab, .symtab, если считанное название соответствует одному из

перечисленных, достанем нужную информацию из секции, иначе будем считывать следующий заголовок.

Считываем данные нужных секций

Когда мы хотим считать данные секции, нам нужны переменные sh_offset и sh_size заголовка. sh_offset место в файле откуда надо читать данные, sh_size - размер данной секции, то есть то, сколько надо считать, чтобы прочесть всю секцию.

Если мы считали заголовок секции .text, значит программа сместится на sh_offset от начала файла и считает sh_size/4 раз по 4 байта, записывая их в вектор 4-х байтовых беззнаковых интов (так как мы работаем только с 32-х битными командами). Этот вектор нам пригодится позже, он называется commands.

Если мы считали заголовок секции .symtab, после смещения на sh_offset программа будет считывать sh_size/16 раз по 16 байт, каждый такой блок помещается в специальную структуру и всё это хранится в векторе symbs.

Если мы считали заголовок секции .strtab, после смещения на sh_offset программа будет считывать sh_size байт в строчку, чтобы потом поместить эту строчку и заголовок секции в специальный класс SectionDataStrtab.

В классе SectionDataStrtab хранится заголовок секции strtab, строчка с данными этой секции и реализован метод get_name. Этот метод понадобится нам, когда мы захотим достать названия символов для вывода symtab. Как мы уже знаем symtab содержит в себе 16-ти байтовые блоки, хранящие информацию о символе, в частности 4 байта st_name - смещение в таблице strtab по которому можно узнать название символа. Метод get_name смещается в строке, куда мы сохранили данные секции strtab на st_name символов и считывает их с этого момента до первого \0, после чего возвращая прочитанное имя.

Парсинг и вывод

Парсинг

Изначально мы объединяем наши symtab и strtab и с помощью функции get_name для каждого символа из вектора symbs ищем его имя.

Теперь в векторе symbs для каждого символа хранится вся необходимая информация для вывода. При конструировании структуры хранящей символ сразу вычисляются строковые интерпретации bind, type и тд

Потом мы проходимся по таблице символов и проставляем метку для адреса, если тип символа равен STT_FUNC.

Далее разберёмся с парсингом инструкций. Этим занимается recognize_and_print_instruct, которая принимает в себя 32 бита инструкции, её адрес и map с метками. От кода каждой инструкции сразу отделяем некоторые блоки, с которыми часто приходится работать. Чтобы однозначно разобрать инструкцию нам достаточно знать последние 7 бит (opcode), 3 бита с 12 по 14 (иногда это часть константы, но для некоторых команд это именно биты отличающие её от других) и в некоторых командах могут отличаться первые 7 бит. Как только мы распознали инструкцию, считаем для неё константу, если это необходимо, после чего основываясь на спецификации RISC-V и ТЗ формируем строку с нужным форматом. Для некоторых команд (BRANCH и JAL) нужно поставить метку рядом с адресом, с которым они работают. Метки проставляются, как описано в условии.

Вывод

Теперь просто по циклу проходимся по всем командам, вычисляя их адрес с помощью формулы sh_addr + номер обрабатываемой команды * 4, выводим метку для этого адреса, если необходимо, а если нет, то запускаем функцию, recognize_and_print_instruct и выводим в файл то, что она вернула.

Далее выводим шапку таблицы symbol table и выводим все элементы вектора symbs, как это указано в формате, после чего можем закрывать выходной поток.

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, 0x100ac <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, 0x10074 <main>
   1009c:    00050593        addi      a1, a0, 0
   100a0:    00a00893        addi      a7, zero, 10
   100a4:    0ff0000f        unknown_instruction
   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
   100c8:    fec50e13        addi      t3, a0, -20
   100cc:    000f0313        addi      t1, t5, 0
   100d0:    000f8893        addi      a7, t6, 0
   100d4:    00000813        addi      a6, zero, 0
   100d8:    00088693        addi      a3, a7, 0
   100dc:    000e0793        addi      a5, t3, 0
   100e0:    00000613        addi      a2, zero, 0
   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, 0x100e4 <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, 0x100d8 <L1>
   10114:    050f0f13        addi      t5, t5, 80
   10118:    01478513        addi      a0, a5, 20
   1011c:    fa5f16e3         bne      t5, t0, 0x100c8 <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
[   2] 0x11124               0 SECTION  LOCAL    DEFAULT       2
[   3] 0x0                   0 SECTION  LOCAL    DEFAULT       3
[   4] 0x0                   0 SECTION  LOCAL    DEFAULT       4
[   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

https://riscv.org/technical/specifications/

https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md

https://docs.oracle.com/cd/E19683-01/816-1386/6m7qcoblj/index.html

https://mark.theis.site/riscv/

https://en.wikichip.org/wiki/risc-v/registers

Листинг кода

main.cpp

```
#include "elfParser.h"
#include "except.h"

#include <fstream>

int main(int argc, char** argv) {
   try {
          ElfParser elf;

          std::ofstream output;
          output.open(argv[2], std::ios::out);

          if (!output)
                throw IO_Exception("File failed to open");

          elf.parse(argv[1], output);

          output.close();
   }
   catch (IO_Exception except) {
          std::cerr << except.what();
   }
}
```

Utility.h

```
#pragma once
#include "elfFileReader.h"

#include <iostream>
#include <algorithm>
#include <map>
#include <string>
#include <vector>

class StHelpers {
private:
    static std::string abstract_getter(int index, const std::map<int,
std::string> &gen) {
        typename std::map<int, std::string>::const_iterator it =
gen.find(index);
        if (it == gen.end())
            return  "UNKNOWN";
        else
            return it->second;
    }
    static std::map<int, std::string> types;
    static std::map<int, std::string> indexes;
    static std::map<int, std::string> visibilities;
    static std::map<int, std::string> binds;

public:

    static std::string get_type(int index) {
        return abstract_getter(index, types);
    }
    static std::string get_index(int index) {
        std::string ind = abstract_getter(index, indexes);
        return (ind != "UNKNOWN") ? ind : std::to_string(index);
    }
    static std::string get_visibility(int index) {
```

```
        return abstract_getter(index, visibilities);
    }
    static std::string get_bind(int index) {
        return abstract_getter(index, binds);
    }
};

struct FieldOfHeaderSectionTable {
    uint32_t sh_name = 0;
    uint32_t sh_type = 0;
    uint32_t sh_flags = 0;
    uint32_t sh_addr = 0;
    uint32_t sh_offset = 0;
    uint32_t sh_size = 0;
    uint32_t sh_link = 0;
    uint32_t sh_info = 0;
    uint32_t sh_addralign = 0;
    uint32_t sh_entsize = 0;

    FieldOfHeaderSectionTable() = default;
    FieldOfHeaderSectionTable(ElfFileReader& elf_data) {
        sh_name = elf_data.get_next_n_bytes(4);
        sh_type = elf_data.get_next_n_bytes(4);
        sh_flags = elf_data.get_next_n_bytes(4);
        sh_addr = elf_data.get_next_n_bytes(4);
        sh_offset = elf_data.get_next_n_bytes(4);
        sh_size = elf_data.get_next_n_bytes(4);
        sh_link = elf_data.get_next_n_bytes(4);
        sh_info = elf_data.get_next_n_bytes(4);
        sh_addralign = elf_data.get_next_n_bytes(4);
        sh_entsize = elf_data.get_next_n_bytes(4);
    };
};

struct SymbOfSymbolTable {
```

```
    uint32_t st_name;
    uint32_t st_value;
    uint32_t st_size;
    uint8_t st_info;
    uint8_t st_other;
    uint16_t st_shndx;
    uint8_t st_type;
    uint8_t st_bind;
    std::string st_type_txt;
    std::string st_vis_txt;
    std::string st_bind_txt;
    std::string st_index_txt;
    std::string name = "";

    SymbOfSymbolTable() = default;
    SymbOfSymbolTable(ElfFileReader& elf_data) {
        st_name = elf_data.get_next_n_bytes(4);
        st_value = elf_data.get_next_n_bytes(4);
        st_size = elf_data.get_next_n_bytes(4);
        st_info = elf_data.get_next_byte();
        st_other = elf_data.get_next_byte();
        st_shndx = elf_data.get_next_n_bytes(2);
        st_type = st_info & 0xf;
        st_bind = st_info >> 4;
        st_vis_txt = StHelpers::get_visibility(st_other);
        st_index_txt = StHelpers::get_index(st_shndx);
        st_type_txt = StHelpers::get_type(st_type);
        st_bind_txt = StHelpers::get_bind(st_bind);

    };
};

class Printer {
private:
    static std::string head;
```

```
    public:
        static void print_symtab(const std::vector<SymbOfSymbolTable>& syms) {
            std::cout << head << std::endl;
            for (int i = 0; i < syms.size(); i++) {
                std::string buffer(100 + syms[i].name.size(), '\0');
                snprintf(&(buffer[0]), 100 + syms[i].name.size(), "[%4i]
0x%-15X %5i %-8s %-8s %-8s %6s %s",
                    i,
                    syms[i].st_value,
                    syms[i].st_size,
                    syms[i].st_type_txt.c_str(),
                    syms[i].st_bind_txt.c_str(),
                    syms[i].st_vis_txt.c_str(),
                    syms[i].st_index_txt.c_str(),
                    syms[i].name.c_str());
                int j = buffer.size() - 1;
                while (buffer[j--] == '\0')
                    buffer.pop_back();
                std::cout << buffer << std::endl;
            }
        };
        static std::string format_lable() {
            return "";
        }

        static std::string format_0_arg(std::string inst_name) {
            return "   " + inst_name;
        }

        static std::string format_2_arg(std::string inst_name, std::string
dest, std::string source) {
            int size = 50 + dest.size() + source.size();
            std::string buffer(size, '\0');
            snprintf(&(buffer[0]), size, "%7s\t%s, %s", inst_name.c_str(),
dest.c_str(), source.c_str());
            int i = buffer.size() - 1;
```

```
            while (buffer[i--] == '\0')
                buffer.pop_back();
            return buffer;
        }

        static std::string format_3_arg(std::string inst_name, std::string
dest, std::string source1, std::string source2) {
            int size = 100 + dest.size() + source1.size() + source2.size();
            std::string buffer(size, '\0');
            snprintf(&(buffer[0]), size, "%7s\t%s, %s, %s", inst_name.c_str(),
dest.c_str(), source1.c_str(), source2.c_str());
            int i = buffer.size() - 1;
            while (buffer[i--] == '\0')
                buffer.pop_back();
            return buffer;
        }

        static std::string format_load_store(std::string inst_name, std::string
dest, std::string offset, std::string source) {
            int size = 100 + dest.size() + offset.size() + source.size();
            std::string buffer(size, '\0');
            snprintf(&(buffer[0]), size, "%7s\t%s, %s(%s)", inst_name.c_str(),
dest.c_str(), offset.c_str(), source.c_str());
            int i = buffer.size() - 1;
            while (buffer[i--] == '\0')
                buffer.pop_back();
            return buffer;
        }

        static std::string get_hex(int32_t dec_num) {
            int size = std::to_string(dec_num).size() + 10;
            std::string buffer(size, '\0');
            snprintf(&(buffer[0]), size, "%x", dec_num);
            int i = buffer.size() - 1;
            while (buffer[i--] == '\0')
                buffer.pop_back();
            return buffer;
```

```
    }

};
```

Utility.cpp

```
#include "Utility.h"

#include <map>
#include <string>

std::map<int, std::string> StHelpers::indexes = {
    {0, "UNDEF"},
    {0xff00, "LOPROC"},
    {0xff1f, "HIPROC"},
    {0xff20, "LIVEPATCH"},
    {0xfff1, "ABS"},
    {0xfff2, "COMMON"},
    {0xffff, "HIRESERVE"}
};
std::map<int, std::string> StHelpers::types = {
    {0, "NOTYPE"},
    {1, "OBJECT"},
    {2, "FUNC"},
    {3, "SECTION"},
    {4, "FILE"},
    {5, "COMMON"},
    {6, "TLS"}
};
std::map<int, std::string> StHelpers::visibilities = {
    {0, "DEFAULT"},
    {1, "INTERNAL"},
    {2, "HIDDEN"},
    {3, "PROTECTED"},
    {4, "EXPORTED"},
```

```
        {5, "SINGLETON"},
        {6, "ELIMINATE"}
    };

    std::map<int, std::string> StHelpers::binds = {
        {0, "LOCAL"},
        {1, "GLOBAL"},
        {2, "WEAK"}
    };
```

except.h

```
    #pragma once
    #include <iostream>
    #include <string>

    struct IO_Exception : public std::logic_error {
        explicit IO_Exception(const std::string& reason) :
std::logic_error(reason) {}
    };

    struct ELF_Data_Exception : public std::logic_error {
        explicit ELF_Data_Exception(const std::string& reason) :
std::logic_error(reason) {}
    };
```

sectionTable.h

```
    #pragma once
    #include "Utility.h"
    #include <iostream>
    #include <string>

    class SectionDataStrtab {
        std::string section_data = "";
    public:
        SectionDataStrtab() = default;
```

```
            std::string get_name(uint32_t offset) const;
            SectionDataStrtab(std::string section_data, FieldOfHeaderSectionTable
header_data);
      };
```

sectionTable.cpp

```
#include "sectionTable.h"

std::string SectionDataStrtab::get_name(uint32_t offset) const {
    if (offset >= section_data.size())
        return "";
    std::string res = "";
    while (section_data[offset] != '\0')
        res += section_data[offset++];

    return res;
}

SectionDataStrtab::SectionDataStrtab(std::string                section_data,
FieldOfHeaderSectionTable header_data) : section_data(section_data) {}
```

elfFileReader.h

```
#pragma once
#include <string>
#include <fstream>
#include <iostream>
#include <vector>

class ElfFileReader {
private:
            std::ifstream my_file;
            void parseClassDataVerExcept(std::string, std::string, std::string,
uint8_t);
            uint32_t byte_vec_2_int(std::vector<uint8_t>);
public:
            ElfFileReader() = default;
            ElfFileReader(std::string input_file);
            void seek(int index_of_byte);
```

```
	int tell();
	uint8_t get_next_byte();
	uint32_t get_next_n_bytes(int num_of_bytes);
	void close_file();
};
```

elfFileReader.cpp

```
#include "elfFileReader.h"
#include "except.h"
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

void ElfFileReader::parseClassDataVerExcept(std::string fail, std::string
ok, std::string unknown, uint8_t to_check){
	if (to_check == 0)
		throw IO_Exception(fail);
	else if (to_check == 2)
		throw IO_Exception(ok);
	else if (to_check != 1)
		throw IO_Exception(unknown);
}

uint32_t ElfFileReader::byte_vec_2_int(std::vector<uint8_t> bytes)
{
	uint64_t res = 0;
	for (int i = 0; i < bytes.size(); i++)
		res += (bytes[i] << (i * 8));
	return res;
}

ElfFileReader::ElfFileReader(std::string input_file) {
	//если файл не найден, обра
	my_file.open(input_file, std::ios::binary | std::ios::in);
```

```
	if (!my_file)
		throw IO_Exception("File failed to open");


	if (get_next_n_bytes(4) != 0x464C457f)
		throw IO_Exception("Not elf file");
	int ei_class_byte = get_next_byte();
	int ei_data_byte = get_next_byte();
	int ei_version_byte = get_next_byte();
	seek(18);
	int e_machine_bytes = get_next_n_bytes(2);
	int e_version_bytes = get_next_n_bytes(4);
	parseClassDataVerExcept("elf class none","elf class 64", "unknown elf
class", ei_class_byte);
	parseClassDataVerExcept("elf data none", "elf data 2msb", "unknown
elf class", ei_data_byte);
	parseClassDataVerExcept("e version is invalid", "unknown e version",
"unknown e version", ei_version_byte);
	if (e_machine_bytes != 0xF3)
		throw IO_Exception("Not RISC-V architecture");
	if (e_version_bytes != 1)
		throw IO_Exception("e version none (invalid value)");
}



uint8_t ElfFileReader::get_next_byte() {
	uint8_t reader;
	my_file.read(reinterpret_cast<char*>(&reader), 1);
	return reader;
}


/*
Can't read more than 4 byte
*/


uint32_t ElfFileReader::get_next_n_bytes(int num_of_bytes) {
	std::vector<uint8_t> byteHolder(num_of_bytes);
```

```
        my_file.read(reinterpret_cast<char*>(&byteHolder[0]), num_of_bytes);
        return byte_vec_2_int(byteHolder);
    }

    void ElfFileReader::seek(int index_of_byte) {
        my_file.seekg(index_of_byte, std::ios::beg);
    }

    int ElfFileReader::tell() {
        return my_file.tellg();
    }

    void ElfFileReader::close_file(){
        my_file.close();
    }
```

elfParcer.h

```
#pragma once
#include <string>
#include <fstream>
#include <iostream>
#include <vector>
#include <map>
#include <cstdio>


class ElfParser {
private:
    static std::string reg_name(uint8_t reg) {
        if (reg == 0)
            return "zero";
        else if (reg == 1)
            return "ra";
        else if (reg == 2)
            return "sp";
        else if (reg == 3)
            return "gp";
        else if (reg == 4)
            return "tp";
        else if (reg >= 5 && reg <= 7)
            return "t" + std::to_string(reg - 5);
        else if (reg >= 8 && reg <= 9)
```

```
            return "s" + std::to_string(reg - 8);
        else if (reg >= 10 && reg <= 17)
            return "a" + std::to_string(reg - 10);
        else if (reg >= 18 && reg <= 27)
            return "s" + std::to_string(reg - 16);
        else if (reg >= 28 && reg <= 31)
            return "t" + std::to_string(reg - 25);
        return "UNKNOWN REG";
    }

public:
    int mark_cnt = 0;
    void parse(std::string input_file_name, std::ofstream& output);
     std::string recognize_and_print_instruct(uint32_t command_data, uint32_t
command_addr, std::map<uint32_t, std::string>& marks);

};
```

elfParcer.cpp

```
#include "elfParser.h"
#include "elfFileReader.h"
#include "sectionTable.h"
#include "Utility.h"
#include "except.h"

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>


void ElfParser::parse(std::string input_file_name, std::ofstream& output) {
   ElfFileReader elf_data = ElfFileReader(input_file_name);
   elf_data.seek(32);

   //     Смещение таблицы заголовков секций от начала файла в байтах. Если у
файла нет таблицы заголовков секций, это поле содержит 0.
   uint32_t e_shoff = elf_data.get_next_n_bytes(4);

   elf_data.seek(46);
   // Размер одного заголовка секции. Все заголовки секций имеют одинаковый
размер (40 для 32-битных файлов и 64 для 64-битных).
   uint32_t e_shentsize = elf_data.get_next_n_bytes(2);

   if (e_shentsize != 40)
          throw ELF_Data_Exception("Unexpectedly e_shentsize not equal 40");

   // Число заголовков секций. Если у файла нет таблицы заголовков секций, это
поле содержит 0.
   uint32_t e_shnum = elf_data.get_next_n_bytes(2);
```

```
   if (e_shnum == 0)
          throw ELF_Data_Exception("No table section header found");

   // Индекс записи в таблице заголовков секций, описывающей таблицу названий
секций (обычно эта таблица называется .shstrtab и представляет собой отдельную
секцию). Если файл не содержит таблицы названий секций, это поле содержит 0.
   uint32_t e_shstrndx = elf_data.get_next_n_bytes(2);

   if (e_shstrndx == 0)
          throw ELF_Data_Exception("No .shstrtab found");

   //working with .shstrtab

   // начало блока .shstrtab в таблице заголовков секций
   elf_data.seek(e_shoff + e_shentsize * e_shstrndx);
   FieldOfHeaderSectionTable shstrtab(elf_data);
   FieldOfHeaderSectionTable text_section;
   SectionDataStrtab strtab_section;
   std::vector<SymbOfSymbolTable> symbs;
   std::vector<uint32_t> commands;
   std::map<uint32_t, std::string> marks;
   for (int k = 0; k < e_shnum; k++) {
          elf_data.seek(e_shoff + k * e_shentsize);
          FieldOfHeaderSectionTable section_header(elf_data);

          std::string section_name = "";
          char chr;
          elf_data.seek(shstrtab.sh_offset + section_header.sh_name);
          while ((chr = elf_data.get_next_byte()) > 0)
                 section_name += chr;

          if (section_name == ".strtab") {
                 std::string section_data = "";
                 elf_data.seek(section_header.sh_offset);
                 for (int i = 0; i < section_header.sh_size; i++)
                        section_data += (elf_data.get_next_byte());
                 strtab_section = SectionDataStrtab(section_data,
section_header);
          }
          else if (section_name == ".symtab") {
                 elf_data.seek(section_header.sh_offset);
                 for (int j = 0; j < section_header.sh_size/16; j ++)
                        symbs.push_back(SymbOfSymbolTable(elf_data));
          }
          else if (section_name == ".text") {
                 elf_data.seek(section_header.sh_offset);
                 for (int j = 0; j < section_header.sh_size / 4; j++)
                        commands.push_back(elf_data.get_next_n_bytes(4));
                 text_section = section_header;
          }
   }
   elf_data.close_file();

   for (int i = 0; i < symbs.size(); i++) {
          symbs[i].name = strtab_section.get_name(symbs[i].st_name);
          if (symbs[i].st_type_txt == "FUNC")
                 marks.insert({ symbs[i].st_value, ((symbs[i].name != "") ?
symbs[i].name : ("L" + std::to_string(mark_cnt++))) });
   }
```

```
   output << ".text" << std::endl;
   for (int i = 0; i < commands.size(); i++) {
          auto mark = marks.find(text_section.sh_addr + i * 4);
          uint32_t addr_val = text_section.sh_addr + i * 4;
          if (mark != marks.end()){
                //"%08x   <%s>:\n"
                int size = 15 + mark->second.size();
                std::string marker(size, '\0');
                snprintf(&(marker[0]), size, "%08x   <%s>:\n", addr_val, (mark
-> second).c_str());
                int j = marker.size() - 1;
                while (marker[j--] == '\0')
                      marker.pop_back();
                output << marker << std::endl;
          }
          //"   %05x:\t%08x\t"
          int size = 50;
          std::string addr(size, '\0');
          snprintf(&(addr[0]), size, "   %05x:\t%08x\t", addr_val,
commands[i]);
          int j = addr.size() - 1;
          while (addr[j--] == '\0')
                addr.pop_back();
          output << addr << recognize_and_print_instruct(commands[i], addr_val,
marks) << std::endl;
   }
   output << std::endl;
   output << ".symtab" << std::endl;
   output << "Symbol Value              Size Type     Bind     Vis       Index
Name" << std::endl;
   for (int i = 0; i < symbs.size(); i++) {
          std::string buffer(100 + symbs[i].name.size(), '\0');
          snprintf(&(buffer[0]), 100 + symbs[i].name.size(), "[%4i] 0x%-15X %5i
%-8s %-8s %-8s %6s %s",
          i,
          symbs[i].st_value,
          symbs[i].st_size,
          symbs[i].st_type_txt.c_str(),
          symbs[i].st_bind_txt.c_str(),
          symbs[i].st_vis_txt.c_str(),
          symbs[i].st_index_txt.c_str(),
          symbs[i].name.c_str());
          int j = buffer.size() - 1;
          while (buffer[j--] == '\0')
                buffer.pop_back();
          output << buffer << std::endl;
   }
}

std::string ElfParser::recognize_and_print_instruct(uint32_t command_data,
uint32_t command_addr, std::map<uint32_t, std::string>& marks)
{

   uint8_t opcode = command_data & 0x7f;
   uint8_t rd___imm_4_0 = (command_data >> 7) & 0x1f;
   uint8_t funct3 = (command_data >> 12) & 0x7;
   uint8_t rs1 = (command_data >> 15) & 0x1f;
   uint8_t rs2___shamt = (command_data >> 20) & 0x1f;
   uint16_t imm_11_0 = command_data >> 20;
```

```
   int16_t signed_imm_11_0 = ((int16_t)imm_11_0) - (imm_11_0 >> 11) * 2 * (1
<< 11);
   uint8_t imm_11_5 = command_data >> 25;
   if (opcode == 0b0110111 || opcode == 0b0010111)
   {
          uint32_t imm_31_12 = (command_data >> 12);
          uint32_t imm = imm_31_12;
          if (opcode == 0b0110111)
                return Printer::format_2_arg("lui", reg_name(rd___imm_4_0),
std::to_string(imm));
          else
                return Printer::format_2_arg("auipc", reg_name(rd___imm_4_0),
std::to_string(imm));
   } else if (opcode == 0b1101111) {
          uint8_t imm_20 = (command_data >> 31) & 1;
          uint16_t imm_10_1 = (command_data >> 21) & 0x3ff;
          uint8_t imm_11 = (command_data >> 20) & 1;
          uint16_t imm_19_12 = (command_data >> 12) & 0xff;
          int32_t imm = (imm_20 << 20 | imm_19_12 << 12 | imm_11 << 11 |
imm_10_1 << 1) - imm_20 * 2 * (1 << 20);
          uint32_t new_addr = command_addr + imm;
          auto mark = marks.find(new_addr);
          std::string mark_str;
          if (mark != marks.end())
                mark_str = mark->second;
          else {
                mark_str = "L" + std::to_string(mark_cnt++);
                marks.insert({ new_addr, mark_str });
          }
          return Printer::format_2_arg("jal", reg_name(rd___imm_4_0), ("0x" +
Printer::get_hex(new_addr) + " <" + mark_str + ">"));
   } else if (opcode == 0b1100111)
          return Printer::format_load_store("jalr",
reg_name(rd___imm_4_0),std::to_string(signed_imm_11_0), reg_name(rs1));
   else if (opcode == 0b1100011) {
          uint8_t imm_12 = imm_11_5 >> 6;
          uint8_t imm_10_5 = imm_11_5 & 0x3f;
          uint8_t imm_4_1 = rd___imm_4_0 >> 1;
          uint8_t imm_11 = rd___imm_4_0 & 1;
          int32_t imm = (imm_12 << 12 | imm_11 << 11 | imm_10_5 << 5 | imm_4_1
<< 1) - imm_12 * 2 * (1 << 12);
          uint32_t new_addr = command_addr + imm;
          auto mark = marks.find(new_addr);
          std::string mark_str;
          if (mark != marks.end())
                mark_str = mark->second;
          else {
                mark_str = "L" + std::to_string(mark_cnt++);
                marks.insert({ new_addr, mark_str });
          }

          if (funct3 == 0b000)
                return Printer::format_3_arg("beq", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
          else if (funct3 == 0b001)
                return Printer::format_3_arg("bne", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
```

```
        else if (funct3 == 0b100)
              return Printer::format_3_arg("blt", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
        else if (funct3 == 0b101)
              return Printer::format_3_arg("bge", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
        else if (funct3 == 0b110)
              return Printer::format_3_arg("bltu", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
        else if (funct3 == 0b111)
              return Printer::format_3_arg("bgeu", reg_name(rs1),
reg_name(rs2___shamt), ("0x" + Printer::get_hex(new_addr) + " <" + mark_str +
">"));
   } else if (opcode == 0b0000011) {
        if (funct3 == 0b000)
              return Printer::format_load_store("lb",
reg_name(rd___imm_4_0), std::to_string(signed_imm_11_0), reg_name(rs1));
        else if (funct3 == 0b001)
              return Printer::format_load_store("lh",
reg_name(rd___imm_4_0), std::to_string(signed_imm_11_0), reg_name(rs1));
        else if (funct3 == 0b010)
              return Printer::format_load_store("lw",
reg_name(rd___imm_4_0), std::to_string(signed_imm_11_0), reg_name(rs1));
        else if (funct3 == 0b100)
              return Printer::format_load_store("lbu",
reg_name(rd___imm_4_0), std::to_string(signed_imm_11_0), reg_name(rs1));
        else if (funct3 == 0b101)
              return Printer::format_load_store("lhu",
reg_name(rd___imm_4_0), std::to_string(signed_imm_11_0), reg_name(rs1));
   } else if (opcode == 0b0100011) {
        int32_t imm = ((uint16_t)imm_11_5 << 5 | rd___imm_4_0) - (imm_11_5 >>
6) * 2 * (1 << 11);
        if (funct3 == 0b000)
              return Printer::format_load_store("sb", reg_name(rs2___shamt),
std::to_string(imm), reg_name(rs1));
        else if (funct3 == 0b001)
              return Printer::format_load_store("sh", reg_name(rs2___shamt),
std::to_string(imm), reg_name(rs1));
        else if (funct3 == 0b010)
              return Printer::format_load_store("sw", reg_name(rs2___shamt),
std::to_string(imm), reg_name(rs1));
   } else if (opcode == 0b0010011) {
        if (funct3 == 0b000)
              return Printer::format_3_arg("addi", reg_name(rd___imm_4_0)
,reg_name(rs1), std::to_string(signed_imm_11_0));
        else if (funct3 == 0b001)
              return Printer::format_3_arg("slli", reg_name(rd___imm_4_0),
reg_name(rs1), std::to_string(rs2___shamt));
        else if (funct3 == 0b010)
              return Printer::format_3_arg("slti", reg_name(rd___imm_4_0),
reg_name(rs1), std::to_string(signed_imm_11_0));
        else if (funct3 == 0b011)
              return Printer::format_3_arg("sltiu", reg_name(rd___imm_4_0),
reg_name(rs1), std::to_string(signed_imm_11_0));
        else if (funct3 == 0b100)
```

```
                return Printer::format_3_arg("xori", reg_name(rd___imm_4_0),
reg_name(rs1), std::to_string(signed_imm_11_0));
          else if (funct3 == 0b101) {
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("srli",
reg_name(rd___imm_4_0), reg_name(rs1), std::to_string(rs2___shamt));
                else if (imm_11_5 == 0b0100000)
                      return Printer::format_3_arg("srai",
reg_name(rd___imm_4_0), reg_name(rs1), std::to_string(rs2___shamt));
          }
          else if (funct3 == 0b110)
                      return Printer::format_3_arg("ori",
reg_name(rd___imm_4_0), reg_name(rs1), std::to_string(signed_imm_11_0));
          else if (funct3 == 0b111)
                      return Printer::format_3_arg("andi",
reg_name(rd___imm_4_0), reg_name(rs1), std::to_string(signed_imm_11_0));
    } else if (opcode == 0b0110011) {
          if (funct3 == 0b000)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("add",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0100000)
                      return Printer::format_3_arg("sub",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                      return Printer::format_3_arg("mul",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b001)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("sll",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                      return Printer::format_3_arg("mulh",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b010)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("slt",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                      return Printer::format_3_arg("mulhsu",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b011)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("sltu",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                      return Printer::format_3_arg("mulhu",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b100)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("xor",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                      return Printer::format_3_arg("div",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b101)
                if (imm_11_5 == 0b0000000)
                      return Printer::format_3_arg("srl",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
```

```
                else if (imm_11_5 == 0b0100000)
                        return Printer::format_3_arg("sra",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                        return Printer::format_3_arg("divu",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b110)
                if (imm_11_5 == 0b0000000)
                        return Printer::format_3_arg("or",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                        return Printer::format_3_arg("rem",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
          if (funct3 == 0b111)
                if (imm_11_5 == 0b0000000)
                        return Printer::format_3_arg("and",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
                else if (imm_11_5 == 0b0000001)
                        return Printer::format_3_arg("remu",
reg_name(rd___imm_4_0), reg_name(rs1), reg_name(rs2___shamt));
   } else if (opcode == 0b1110011) {
          if (imm_11_0 == 0b000000000000)
                return Printer::format_0_arg("ecall");
          if (imm_11_0 == 0b000000000001)
                return Printer::format_0_arg("ebreak");
   }

   return Printer::format_0_arg("unknown_instruction");
}
```