

|                        |                                |      |
|------------------------|--------------------------------|------|
| Лабораторная работа №3 | Б10                            | 2022 |
| ISA                    | Барковская Мария Александровна |      |

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

**Инструментарий и требования к работе:** работа выполнена на C++ с использованием компилятора “Apple clang version 13.1.6 (clang-1316.0.21.2.5)”

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

# Описание системы кодирования команд RISC-V

RISC-V — открытая и свободная система команд и процессорная архитектура на основе концепции RISC (англ.: reduced instruction set computer — вычислитель с набором упрощённых/редуцированных команд — архитектурный подход к проектированию процессоров, в которой быстродействие увеличивается за счёт такого кодирования инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим) для микропроцессоров и микроконтроллеров.

В архитектуре RISC-V имеется базовое подмножество команд (набор инструкций I — Integer), обязательных для реализации, и несколько стандартных опциональных расширений. В базовый набор входят инструкции условной и безусловной передачи управления/ветвления, минимальный набор арифметических/битовых операций на регистрах, операций с памятью (load/store), а также небольшое число служебных инструкций.

В RISC-V предусмотрены реализации архитектур с 32, 64 и 128-битными регистрами общего назначения и операциями: RV32I, RV64I и RV128I соответственно (при одинаковой кодировке инструкций).

Упомянем несколько стандартных опциональных расширений:

- **C — Compressed extension:** для наиболее часто используемых инструкций стандартизовано применение их аналогов в более компактной (16-битной) кодировке.
- **M — Multiply extension:** операции умножения, деления и вычисления остатка (они не входят в минимальный набор инструкций)
- **A — Atomic extension:** набор атомарных операций

Согласно нашему заданию, должен поддерживаться следующий набор команд: RISC-V RV32I, RV32M, то есть базовый набор с целочисленными операциями (32-битный) и целочисленное умножение и деление соответственно.

Система кодирования RISC-V строится, как показано на рисунке 1, где opcode (operation code) и funct (что 3, что 7) определяют выполняемую операцию, rd (register destination) — регистр, куда попадает результат, rs (register source) — регистр, откуда берется значение (причем rs1 — номер регистра в котором находится первый операнд, rs2 — номер регистра в котором находится второй операнд), imm (immediate) в терминах RISC это константа, которую мы можем получить из кода, не обращаясь к памяти.

| 31                    | 27 | 26 | 25 | 24  | 20 | 19  | 15 | 14     | 12 | 11          | 7 | 6      | 0 |        |
|-----------------------|----|----|----|-----|----|-----|----|--------|----|-------------|---|--------|---|--------|
| funct7                |    |    |    | rs2 |    | rs1 |    | funct3 |    | rd          |   | opcode |   | R-type |
| imm[11:0]             |    |    |    |     |    | rs1 |    | funct3 |    | rd          |   | opcode |   | I-type |
| imm[11:5]             |    |    |    | rs2 |    | rs1 |    | funct3 |    | imm[4:0]    |   | opcode |   | S-type |
| imm[12 10:5]          |    |    |    | rs2 |    | rs1 |    | funct3 |    | imm[4:1 11] |   | opcode |   | B-type |
| imm[31:12]            |    |    |    |     |    |     |    |        |    | rd          |   | opcode |   | U-type |
| imm[20 10:1 11 19:12] |    |    |    |     |    |     |    |        |    | rd          |   | opcode |   | J-type |

Рисунок 1 – Типы Команд

На рисунке 1 наглядно видно, что инструкции бывают нескольких типов. К какому типу относится команда можно определить, используя opcode и funct3. Чтобы же определить команду внутри типа для типа R ещё добавляется funct7. Для примера, на рисунке 2 показаны все конструкции типа R, где различные значения funct7 + funct3 определяют различные команды.

|         |     |     |     |    |         |      |
|---------|-----|-----|-----|----|---------|------|
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | ADD  |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | SUB  |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | SLL  |
| 0000000 | rs2 | rs1 | 010 | rd | 0110011 | SLT  |
| 0000000 | rs2 | rs1 | 011 | rd | 0110011 | SLTU |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | XOR  |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | SRL  |
| 0100000 | rs2 | rs1 | 101 | rd | 0110011 | SRA  |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | OR   |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | AND  |

Рисунок 2: Примеры конструкций RV32I типа R.

Теперь, чуть подробнее о типах инструкций:

- **R инструкции:** используются для работы на регистрах (данные считываются из rs1, rs2 и результат записывается в rd)
- **I инструкции:** используются для действий, где одна из переменных берётся прямо из инструкции из imm.
- **S инструкции:** используются для записи значений в память (imm используются для определения дополнительного сдвига адреса памяти)
- **B инструкции:** используются для условных переходов
- **J инструкции:** используются для того, что совершить переход (“прыжок”) в другое место.
- **U инструкции:** используются для записи верхних 20 бит в какой-либо регистр.

RISC-V имеет 32 (или 16 для встраиваемых применений) целочисленных регистра (команды для вещественных используют дополнительные «вещественные» регистры). Целочисленные регистры обозначаются как

сочетание “х” и номера регистра (для вещественных используется буква “f”). На рисунке 3 можно увидеть соглашение по названию регистров.

| Register name                                | Symbolic name | Description                            | Saved by |
|--|---------------|--|----------|
| <b>32 integer registers</b>                  |               |  |          |
| x0   | Zero          | Always zero                            |          |
| x1   | ra            | Return address                         | Caller   |
| x2   | sp            | Stack pointer                          | Callee   |
| x3   | gp            | Global pointer                         |          |
| x4   | tp            | Thread pointer                         |          |
| x5   | t0            | Temporary / alternate return address   | Caller   |
| x6–7   | t1–2          | Temporary                              | Caller   |
| x8   | s0/fp         | Saved register / frame pointer         | Callee   |
| x9   | s1            | Saved register                         | Callee   |
| x10–11                                       | a0–1          | Function argument / return value       | Caller   |
| x12–17                                       | a2–7          | Function argument                      | Caller   |
| x18–27                                       | s2–11         | Saved register                         | Callee   |
| x28–31                                       | t3–6          | Temporary                              | Caller   |
| <b>32 floating-point extension registers</b> |               |  |          |
| f0–7   | ft0–7         | Floating-point temporaries             | Caller   |
| f8–9   | fs0–1         | Floating-point saved registers         | Callee   |
| f10–11                                       | fa0–1         | Floating-point arguments/return values | Caller   |
| f12–17                                       | fa2–7         | Floating-point arguments               | Caller   |
| f18–27                                       | fs2–11        | Floating-point saved registers         | Callee   |
| f28–31                                       | ft8–11        | Floating-point temporaries             | Caller   |

Рисунок 3 - соглашение по названию регистров

# Описание структуры файла ELF

ELF (англ. Executable and Linkable Format — формат исполнимых и компокуемых файлов) — формат исполняемых двоичных файлов, используемый во многих современных UNIX-подобных операционных системах, таких как FreeBSD, Linux, Solaris и др. Он определяет структуру бинарных файлов, библиотек, и файлов ядра и позволяет операционной системе корректно интерпретировать содержащиеся в файле машинные команды. Файл elf имеет двоичный формат и, как правило, является результирующим файлом работы компилятора или линкера.

Рассмотрим структуру elf-файла. Каждый elf файл состоит из следующих частей: заголовок файла и сами данные. Подробнее стоит поговорить про заголовок, так как основные взаимодействия происходят с ним или через него. Заголовок elf -файла состоит из трёх частей - сам elf header, Program header и Section Header-ы.

В elf header-е дана основная информация о файле: тип, версия формата, архитектура процессора, размеры, смещения остальных частей файла. Он имеет размер 52 байта для 32-битных файлов (индексация с 0).

Первые четыре байта — так называемые “магические” числа самого elf -файла — 0x7F и идущие за ними закодированные буквы ELF (45 4c 46). Дальше идёт байт формата файла (1/2 для 32-/64-битного формата соответственно). За ним байт, который определяет endian (1/2 для little/big endian соответственно). Дальше идёт байт версии elf, всегда должно равняться 1. Стоит упомянуть еще e\_shoff, который хранит в себе смещение таблицы заголовков секций от начала файла в байтах, e\_shentsize, который хранит в себе размер одного section header-a, e\_shnum, который хранит в себе количество section header-ов, e\_shstrndx хранит индекс секции, которая хранит в себе индекс записи, описывающей таблицу названий секций. Полную информацию о структуре elf header-a можно увидеть на рисунке 4

| HEXADESIMAL DUMP |  | ASCII DUMP | FIELDS            | VALUES                           | EXPLANATION                              |
|------------------|--|------------|-------------------|----------------------------------|--|
|                  |  |            | 1 e_ident         |                                  |  |
|                  |  |            | EI_MAG            | 0x7F, "ELF"                      | CONSTANT SIGNATURE                       |
|                  |  |            | EI_CLASS, EI_DATA | 1 LITTLE_ENDIAN, 1 LITTLE_ENDIAN | 32 BITS, LITTLE-ENDIAN                   |
|                  |  |            | EI_VERSION        | 1                                | ALWAYS 1                                 |
|                  |  |            | e_type            | 2 EXECUTABLE                     | EXECUTABLE                               |
|                  |  |            | e_machine         | 28 ARM                           | ARM PROCESSOR                            |
|                  |  |            | e_version         | 1                                | ALWAYS 1                                 |
|                  |  |            | e_entry           | 0x00000000                       | ADDRESS WHERE EXECUTION STARTS           |
|                  |  |            | e_phoff           | 0x40                             | PROGRAM HEADERS' OFFSET                  |
|                  |  |            | e_shoff           | 0x00                             | SECTION HEADERS' OFFSET                  |
|                  |  |            | e_eshize          | 0x34                             | ELF HEADER'S SIZE                        |
|                  |  |            | e_phentsize       | 0x20                             | SIZE OF A SINGLE PROGRAM HEADER          |
|                  |  |            | e_phnum           | 1                                | COUNT OF PROGRAM HEADERS                 |
|                  |  |            | e_shentsize       | 0x28                             | SIZE OF A SINGLE SECTION HEADER          |
|                  |  |            | e_shnum           | 4                                | COUNT OF SECTION HEADERS                 |
|                  |  |            | e_shstrndx        | 3                                | INDEX OF THE NAMES' SECTION IN THE TABLE |

рисунок 4 - структура elf header-a

Блок section header-ов хранит в себе секции, на которые разбит весь код. В разных секциях лежат данные разного назначения. Так, из тех секций, которые нам пригодятся, есть `text` – это основной исполняемый код, `symbol table (symtab)` – здесь хранится вся информация, чтобы найти или переместить нужные переменные и ссылки, `string table (strtab)` – здесь хранится информация про имена переменных, `shstrtab` – здесь хранятся названия всех секций (название каждой секции – это строка из символов таблицы названий секций начиная с индекса `sh_name` и заканчивая символом “\0”, не включая его), индекс к которой нам известен из упомянутого выше `e_shstrndx`.

Вернёмся к section header-ам. На начало первого указывает `e_shoff`. У каждой секции есть собственный section header размера `e_shentsize`, которые идут в памяти друг за другом. Аналогично в заголовке каждой секции есть немало полезной информации, но для нас будут иметь значения `sh_offset` – начало секции по сравнению с началом файла, `sh_size` – размер самой секции в байтах.

# Описание работы написанного кода

Схема работы программы довольно проста:

- Открываем `elf_file`;
- Считываем `elf_header`, проверяем некоторые из значений (то, что файл действительно elf (`e_ident[1]+..+e_ident[3] = 'ELF'`), то, что он 32-битный, в little endian и для risc-v)
- Считываем все `section_header`-ы
- Прочитаем секцию `shstrtab`, чтобы узнать имена всех секций и, таким образом, найти индексы секций `symtab`, `text` и `strtab` (то есть, их местоположение (и их `section_header`-ов) относительно остальных)
- Парсим `symtab` (считывая все символы по порядку в цикле)
- Получаем имена символов (воспользовавшись вспомогательным адресом `section_headers[elf_header.e_shstrndx].sh_offset`, для каждого символа сдвигаемся от него (адреса) на `symbol.st_name` и считываем имя, параллельно запоминая имена функций в отдельный `vector`)
- Парсим и пишем в выходной файл `text (disassembler)`: в цикле считываем команду и в зависимости от ее типа выводим соответствующую строку в выходной файл (то есть используем соответствующий формат строки) с информацией об этой строке кода
- Пишем в выходной файл `symtab` (просто в цикле для каждого символа выводим информацию о нем, такую как индекс, значение, тип, имя и так далее)

Стоит упомянуть также функцию `format`. Она позволяет форматировать `std::string` в привычном варианте, при этом работая с `ofstream` (реализация не моя, нашла в интернете (в источниках есть ссылка))

# Результат работы написанной программы на приложенном к заданию файле (дизассемблер и таблицу СИМВОЛОВ)

Результат работы программы таков:

```
.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://ru.wikipedia.org/wiki/RISC-V>
- <https://en.wikipedia.org/wiki/RISC-V#Immediates>
- <https://ru.wikipedia.org/wiki/RISC>
- [https://inst.eecs.berkeley.edu/~cs61c/resources/su18\\_lec/Lecture7.pdf](https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture7.pdf)
- [https://ru.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format](https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format)
- [https://en.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format)
- [https://upload.wikimedia.org/wikipedia/commons/e/e4/ELF\\_Executable\\_and\\_Linkable\\_Format\\_diagram\\_by\\_Ange\\_Albertini.png](https://upload.wikimedia.org/wikipedia/commons/e/e4/ELF_Executable_and_Linkable_Format_diagram_by_Ange_Albertini.png)
- <https://habr.com/ru/post/480642/>
- [https://docs.oracle.com/cd/E26502\\_01/html/E26507/chapter6-79797.html#chapter6-tbl-23](https://docs.oracle.com/cd/E26502_01/html/E26507/chapter6-79797.html#chapter6-tbl-23) (это, кажется, ссылка только на symbol table, но на сайте, в целом, много полезных разделов)
- <https://habr.com/ru/post/318962/> (отсюда была позаимствована функция format, упомянутая ранее)
- <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>
- <https://v2020e.ru/blog/sozdanie-protssessora-so-svobodnoj-arkhitekturoj-risc-v-chast-1>

# Листинг кода

```
main.cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <stdarg.h>
#include <map>

std::map <unsigned int, std::string> symbol_types = {
    {0, "NOTYPE"},
    {1, "OBJECT"},
    {2, "FUNC"},
    {3, "SECTION"},
    {4, "FILE"},
    {5, "COMMON"},
    {6, "TLS"},
    {10, "LOOS"},
    {12, "HIOS"},
    {13, "LOPROC"},
    {15, "HIPROS"}
};

std::map <unsigned int, std::string> symbol_binds = {
    {0, "LOCAL"},
    {1, "GLOBAL"},
    {2, "WEAK"},
    {10, "LOOS"},
    {12, "HIOS"},
    {13, "LOPROC"},
    {15, "HIPROS"}
};

std::map <int, std::string> symbol_visibilities = {
    {0, "DEFAULT"},
    {1, "INTERNAL"},
    {2, "HIDDEN"},
    {3, "PROTECTED"},
    {4, "EXPORTED"},
    {5, "SINGLETON"},
    {6, "ELIMINATE"}
};

std::map <int, std::string> symbol_indexes = {
    {0, "UNDEF"},
    {0xff00, "LOPROC"},
    {0xff1f, "HIPROC"},
    {0xff20, "LOOS"},
    {0xffff1, "ABS"},
    {0xffff2, "COMMON"},
    {0xfffff, "HIRESERVE"}
};

struct elf_file_header {
    unsigned char e_ident[16];
    uint16_t e_type;
    uint16_t e_machine;
    uint32_t e_version;
    uint32_t e_entry;
```

```

        uint32_t e_phoff;
        uint32_t e_shoff;
        uint32_t e_flags;
        uint16_t e_ehsize;
        uint16_t e_phentsize;
        uint16_t e_phnum;
        uint16_t e_shentsize;
        uint16_t e_shnum;
        uint16_t e_shstrndx;
};

struct section_header {
    uint32_t sh_name;
    uint32_t sh_type;
    uint32_t sh_flags;
    uint32_t sh_addr;
    uint32_t sh_offset;
    uint32_t sh_size;
    uint32_t sh_link;
    uint32_t sh_info;
    uint32_t sh_addralign;
    uint32_t sh_entsize;
};

struct elf_symbol {
    uint32_t      st_name;
    uint32_t      st_value;
    uint32_t      st_size;
    unsigned char st_info;
    unsigned char st_other;
    uint16_t      st_shndx;
};

int parse_symtab(std::ifstream& elf_file, section_header symtab_section_header,
std::vector<elf_symbol>& symtab) {
    elf_file.seekg(symtab_section_header.sh_offset, std::ios::beg);

    for (int i = 0; i < symtab_section_header.sh_size /
symtab_section_header.sh_entsize; i++) {
        elf_symbol elem;

        if (!elf_file.read((char*)&elem, symtab_section_header.sh_entsize).good()) {
            std::cerr <<"Error: Could not read symbol in symtab" << std::endl;
            return 1;
        }

        symtab.push_back(elem);
    }

    return 0;
}

std::string get_symbol_type(int index_of_symbol) {
    int index = index_of_symbol & 0xf;

    if (symbol_types.find(index) == symbol_types.end()) {
        index = 0;
    }

    return symbol_types[index];
}

```

```

std::string get_symbol_bind(int index_of_symbol) {
    int index = index_of_symbol >> 4;

    if (symbol_binds.find(index) == symbol_binds.end()) {
        index = 0;
    }

    return symbol_binds[index];
}

std::string get_symbol_visability(int index_of_symbol) {
    int index = index_of_symbol & 0x3;

    if (symbol_visibilities.find(index) == symbol_visibilities.end()) {
        index = 0;
    }

    return symbol_visibilities[index];
}

std::string get_symbol_index (int index) {
    if (symbol_indexes.find(index) != symbol_indexes.end()) {
        return symbol_indexes[index];
    }

    return std::to_string(index);
}

std::string format(const char *fmt, ...) { //полезная функция позволяющая
форматировать на строках (нашла реализацию в интернете) //TODO: не забыть упомянуть
об этом в отчете
    va_list args;
    va_start(args, fmt);
    std::vector<char> v(1024);

    while (true) {
        va_list args2;
        va_copy(args2, args);
        int res = vsnprintf(v.data(), v.size(), fmt, args2);

        if ((res >= 0) && (res < static_cast<int>(v.size()))) {
            va_end(args);
            va_end(args2);
            return std::string(v.data());
        }

        size_t size;

        if (res < 0) {
            size = v.size() * 2;
        } else {
            size = static_cast<size_t>(res) + 1;
        }

        v.clear();
        v.resize(size);
        va_end(args2);
    }
}

```

```

int get_names_of_symbols(std::vector<elf_symbol> &symtab, std::ifstream& elf_file,
int help_address, std::vector<std::string>& names_of_symbols, std::map<int,
std::string>& addresses_of_function_names) {
    for (int i = 0; i < symtab.size(); i++) {
        elf_symbol symbol = symtab[i];
        std::string name;

        elf_file.seekg(help_address + symbol.st_name, std::ios::beg);

        char char_buffer = ' ';
        while (char_buffer != '\0') {
            if (!elf_file.read((char *) &char_buffer, 1).good()) {
                std::cerr <<"Error: Could not read name of symbol" << i << std::endl;
                return 1;
            }
            if (char_buffer != '\0') {
                name += char_buffer;
            }
        }

        names_of_symbols.push_back(name);

        if (get_symbol_type(symbol.st_info) == "FUNC" and !name.empty()) {
            addresses_of_function_names[symbol.st_value] = name;
        }
    }
    return 0;
}

int write_symtab(std::vector<elf_symbol> &symtab, std::ifstream& elf_file,
std::ofstream& fout, std::vector<std::string>& names_of_symbols) {

    fout << '\n' << ".symtab" << '\n';

    fout << "Symbol Value          Size Type      Bind      Vis      Index Name" <<
'\n';

    for (int i = 0; i < symtab.size(); i++) {
        elf_symbol symbol = symtab[i];

        std::string temp_str = format("[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s", i,
symbol.st_value, symbol.st_size,
get_symbol_type(symbol.st_info).c_str(),
get_symbol_bind(symbol.st_info).c_str(),
get_symbol_visability(symbol.st_other).c_str(),
get_symbol_index(symbol.st_shndx).c_str(),
names_of_symbols[i].c_str()
);

        fout << temp_str << '\n';
    }

    return 0;
}

const uint32_t OPCODE_SIZE = 7;
const uint32_t RD_SIZE = 5;
const uint32_t FUNCT3_SIZE = 3;
const uint32_t RS_SIZE = 5;
const uint32_t FUNCT7_SIZE = 7;

std::map <int, std::string> reg = {

```

```

    {0, "zero"},
    {1, "ra"},
    {2, "sp"},
    {3, "gp"},
    {4, "tp"},
    {5, "t0"},
    {6, "t1"},
    {7, "t2"},
    {8, "s0"},
    {9, "s1"},
    {10, "a0"},
    {11, "a1"},
    {12, "a2"},
    {13, "a3"},
    {14, "a4"},
    {15, "a5"},
    {16, "a6"},
    {17, "a7"},
    {18, "s2"},
    {19, "s3"},
    {20, "s4"},
    {21, "s5"},
    {22, "s6"},
    {23, "s7"},
    {24, "s8"},
    {25, "s9"},
    {26, "s10"},
    {27, "s11"},
    {28, "t3"},
    {29, "t4"},
    {30, "t5"},
    {31, "t6"}
};

struct R_type {
    uint32_t    opcode;
    uint32_t    rd;
    uint32_t    funct3;
    uint32_t    rs1;
    uint32_t    rs2;
    uint32_t    funct7;

    std::map<std::vector<uint32_t>, std::string> command = {
        {{0, 0}, "add"},
        {{(1 << 5), 0}, "sub"},
        {{0, 1}, "sll"},
        {{0, 2}, "slt"},
        {{0, 3}, "sltu"},
        {{0, 4}, "xor"},
        {{0, 5}, "sr1"},
        {{(1 << 5), 5}, "sra"},
        {{0, 6}, "or"},
        {{0, 7}, "and"},
        {{1, 0}, "mul"},
        {{1, 1}, "mulh"},
        {{1, 2}, "mulhsu"},
        {{1, 3}, "mulhu"},
        {{1, 4}, "div"},
        {{1, 5}, "divu"},
        {{1, 6}, "rem"},
        {{1, 7}, "remu"}
    };
};

```

```

};

void parse (uint32_t command) {
    opcode = command % (1 << OPCODE_SIZE);
    command = command >> OPCODE_SIZE;

    rd = command % (1 << RD_SIZE);
    command = command >> RD_SIZE;

    funct3 = command % (1 << FUNCT3_SIZE);
    command = command >> FUNCT3_SIZE;

    rs1 = command % (1 << RS_SIZE);
    command = command >> RS_SIZE;

    rs2 = command % (1 << RS_SIZE);
    command = command >> RS_SIZE;

    funct7 = command;
}
};

struct I_type {
    uint32_t    opcode;
    uint32_t    rd;
    uint32_t    funct3;
    uint32_t    rs1;
    int32_t     cnst;

    std::map <std::vector <uint32_t>, std::string> command = {
        {{3, 0},"lb"},
        {{3, 1},"lh"},
        {{3, 2},"lw"},
        {{3, 4},"lbu"},
        {{3, 5},"lhu"},
        {{19, 0},"addi"},
        {{19, 2},"slti"},
        {{19, 3},"sltiu"},
        {{19, 4},"xori"},
        {{19, 6},"ori"},
        {{19, 7},"andi"},
        {{19,1}, "slli"},
        {{19,5,0}, "srli"},
        {{18,5,(1<<5)}, "srai"},
        {{103, 0},"jalr"}
    };
};

void parse (uint32_t command) {
    opcode = command % (1 << OPCODE_SIZE);
    command = command >> OPCODE_SIZE;

    rd = command % (1 << RD_SIZE);
    command = command >> RD_SIZE;

    funct3 = command % (1 << FUNCT3_SIZE);
    command = command >> FUNCT3_SIZE;

    rs1 = command % (1 << RS_SIZE);
    command = command >> RS_SIZE;

    cnst = command;
    cnst -= (command >> 11) * (2 << 11);
}

```



```

    }
};

struct S_type {
    uint32_t    opcode;
    uint32_t    cnst4_0;
    uint32_t    funct3;
    uint32_t    rs1;
    uint32_t    rs2;
    uint32_t    cnst11_5;
    int32_t     cnst;

    std::map <std::vector <uint32_t>, std::string> command = {
        {{35, 0}, "sb"},
        {{35, 1}, "sh"},
        {{35, 2}, "sw"},
    };

    void parse (uint32_t command) {
        opcode = command % (1 << OPCODE_SIZE);
        command = command >> OPCODE_SIZE;

        cnst4_0 = command % (1 << RD_SIZE);
        command = command >> RD_SIZE;

        funct3 = command % (1 << FUNCT3_SIZE);
        command = command >> FUNCT3_SIZE;

        rs1 = command % (1 << RS_SIZE);
        command = command >> RS_SIZE;

        rs2 = command % (1 << RS_SIZE);
        command = command >> RS_SIZE;

        cnst11_5 = command;

        cnst = (cnst11_5 << 5) + cnst4_0;
        cnst -= (cnst >> 11) * (2 << 11);
    }
};

struct B_type {
    uint32_t    opcode;
    uint32_t    funct3;
    uint32_t    rs1;
    uint32_t    rs2;
    uint32_t    cnst4_1;
    uint32_t    cnst10_5;
    uint32_t    cnst11;
    uint32_t    cnst12;
    int32_t     cnst;

    std::map <std::vector <uint32_t>, std::string> command = {
        {{99, 0}, "beq"},
        {{99, 1}, "bne"},
        {{99, 4}, "blt"},
        {{99, 5}, "bge"},
        {{99, 6}, "bltu"},
        {{99, 7}, "bgeu"},
    };

    void parse (uint32_t command) {

```

```

opcode = command % (1 << OPCODE_SIZE);
command = command >> OPCODE_SIZE;

cnst11 = command % (1 << 1);
command = command >> 1;

cnst4_1 = command % (1 << 4);
command = command >> 4;

funct3 = command % (1 << FUNCT3_SIZE);
command = command >> FUNCT3_SIZE;

rs1 = command % (1 << RS_SIZE);
command = command >> RS_SIZE;

rs2 = command % (1 << RS_SIZE);
command = command >> RS_SIZE;

cnst10_5 = command % (1 << 6);
command = command >> 6;

cnst12 = command;

cnst = ((((((cnst12 << 1) + cnst11) << 6) + cnst10_5) << 4) + cnst4_1) << 1;
cnst -= (cnst >> 12) * (2 << 12);

    }
};

struct U_type {
    uint32_t    opcode;
    uint32_t    rd;
    int64_t     cnst;

    std::map <std::vector <uint32_t>, std::string> command = {
        {{55}, "lui"},
        {{23}, "auipc"}
    };

    void parse (uint32_t command) {
        opcode = command % (1 << OPCODE_SIZE);
        command = command >> OPCODE_SIZE;

        rd = command % (1 << RD_SIZE);
        command = command >> RD_SIZE;

        cnst = command;
    }
};

struct J_type {
    uint32_t    opcode;
    uint32_t    rd;
    uint32_t    cnst_19_12;
    uint32_t    cnst_11;
    uint32_t    cnst_10_1;
    uint32_t    cnst_20;
    int64_t     cnst;

    std::map <std::vector <uint32_t>, std::string> command = {
        {{111}, "jal"}
    };
};

```

```

void parse (uint32_t command) {
    opcode = command % (1 << OPCODE_SIZE);
    command = command >> OPCODE_SIZE;

    rd = command % (1 << RD_SIZE);
    command = command >> RD_SIZE;

    cnst_19_12 = command % (1 << 8);
    command = command >> 8;

    cnst_11 = command % (1 << 1);
    command = command >> 1;

    cnst_10_1 = command % (1 << 10);
    command = command >> 10;

    cnst_20 = command;

    cnst = ((((((cnst_20 << 8) + cnst_19_12) << 1) + cnst_11) << 10) + cnst_10_1)
<< 1;
    cnst -= (cnst >> 20) * (2 << 20);
}
};

int parse_and_write_text_or_disassembler(std::vector<elf_symbol> &symtab,
std::ifstream& elf_file, std::ofstream& fout, section_header text, std::map<int,
std::string>& addresses_of_function_names) {
    uint32_t addr = text.sh_addr;

    fout << ".text" << '\n';

    int counter = 0;

    for (int i = 0; i < text.sh_size / 4; i++) {
        if (addresses_of_function_names.count(addr)) {
            std::string temp_str = format("%08x    <%s>:\n", addr,
addresses_of_function_names[addr].c_str());
            fout << temp_str;
        }

        uint32_t command;
        elf_file.seekg(text.sh_offset + i * 4, std::ios::beg);

        if (!elf_file.read((char *) &command, sizeof(command)).good()) {
            std::cerr <<"Error: Could not read command" << std::endl;
            return 1;
        }

        uint32_t opcode = command % (1 << OPCODE_SIZE);

        if (opcode == 51) {
            R_type type;
            type.parse(command);

            std::string temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, %s\n", addr,
command,
                type.command[{type.funct7, type.funct3}].c_str(),
                reg[type.rd].c_str(), reg[type.rs1].c_str(),
reg[type.rs2].c_str()
);

```

```

        fout << temp_str;
    } else if (opcode == 3 || opcode == 19 || opcode == 103) {
        I_type type;
        type.parse(command);

        std::string temp_str;

        if (opcode == 19) {
            if (type.funct3 == 5) {
                if ((type.cnst >> RS_SIZE) == 0) {
                    temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, %s\n", addr,
command,
                                "srli", reg[type.rd].c_str(),
                                std::to_string(type.cnst).c_str()
                                );
                } else {
                    temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, %s\n", addr,
command,
                                "srai", reg[type.rd].c_str(),
                                std::to_string(type.cnst - (1 <<
10)).c_str()
                                );
                }
            } else {
                temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, %s\n", addr,
command,
                                type.command[{opcode, type.funct3}].c_str(),
                                reg[type.rd].c_str(), reg[type.rs1].c_str(),
                                std::to_string(type.cnst).c_str()
                                );
            }
        } else {
            temp_str = format("    %05x:\t%08x\t%7s\t%s, %s(%s)\n", addr, command,
type.funct3}.c_str(),
                                type.command[{opcode,
reg[type.rs1].c_str()
                                },
                                reg[type.rd].c_str(),
                                std::to_string(type.cnst).c_str(),
                                );
        }

        fout << temp_str;
    } else if (opcode == 35) {
        S_type type;
        type.parse(command);

        std::string temp_str = format("    %05x:\t%08x\t%7s\t%s, %s(%s)\n", addr,
command,
                                type.command[{opcode, type.funct3}].c_str(),
                                reg[type.rs2].c_str(),
                                std::to_string(type.cnst).c_str(), reg[type.rs1].c_str()
                                );

        fout << temp_str;
    } else if (opcode == 99) {
        B_type type;
        type.parse(command);

        std::string temp_str;

```

```

        if (addresses_of_function_names.count(addr+type.cnst)) {
            temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, 0x%05x <%s>\n", addr,
command,
                                type.command[{opcode, type.funct3}].c_str(),
                                reg[type.rs1].c_str(), reg[type.rs2].c_str(), addr
+ type.cnst,
                                addresses_of_function_names[addr+type.cnst].c_str()
                                );
        } else {
            temp_str = format("    %05x:\t%08x\t%7s\t%s, %s, 0x%05x <L%d>\n",
addr, command,
                                type.command[{opcode, type.funct3}].c_str(),
                                reg[type.rs1].c_str(), reg[type.rs2].c_str(), addr
+ type.cnst, counter
                                );
            addresses_of_function_names[addr + type.cnst] = ("L" +
std::to_string(counter));
            counter++;
        }
        fout << temp_str;
    } else if (opcode == 23 || opcode == 55) {
        U_type type;
        type.parse(command);

        std::string temp_str = format("    %05x:\t%08x\t%7s\t%s, %s\n", addr,
command,
                                type.command[{opcode}].c_str(),
                                reg[type.rd].c_str(), std::to_string(type.cnst).c_str()
                                );

        fout << temp_str;
    } else if (opcode == 111) {
        J_type type;
        type.parse(command);

        std::string temp_str;

        if (addresses_of_function_names.count(addr + type.cnst)) {
            temp_str = format("    %05x:\t%08x\t%7s\t%s, 0x%05x <%s>\n", addr,
command,
                                type.command[{opcode}].c_str(),
                                reg[type.rd].c_str(), addr + type.cnst,
addresses_of_function_names[addr + type.cnst].c_str()
                                );
        } else {
            temp_str = format("    %05x:\t%08x\t%7s\t%s, 0x%05x <L%d>\n", addr,
command,
                                type.command[{opcode}].c_str(),
                                reg[type.rd].c_str(), addr + type.cnst, counter
                                );

            addresses_of_function_names[addr + type.cnst] = ("L" +
std::to_string(counter));
            counter++;
        }

        fout << temp_str;
    } else if (opcode == 115) {
        uint32_t parse_command = command;
        parse_command = parse_command >> (OPCODE_SIZE + RD_SIZE + FUNCT3_SIZE +
RS_SIZE);
        uint32_t rd = (command >> OPCODE_SIZE) % (1 << RD_SIZE);

```

```

        uint32_t funct3 = (command >> (OPCODE_SIZE + RD_SIZE)) % (1 <<
FUNCT3_SIZE);
        uint32_t rs = (command >> (OPCODE_SIZE + RD_SIZE + FUNCT3_SIZE)) % (1 <<
RS_SIZE);
        uint32_t cnst = int32_t (parse_command - (parse_command >> 11) * (2 <<
11));

        if (rd == 0 && funct3 == 0 && rs == 0) {
            if (cnst == 0) {
                std::string temp_str = format("    %05x:\t%08x\t%s\n", addr,
command, "    ecall");
                fout << temp_str;
            } else {
                std::string temp_str = format("    %05x:\t%08x\t%s\n", addr,
command, "    ebreak");
                fout << temp_str;
            }
        } else {
            std::string temp_str = format("    %05x:\t%08x\t%s\n", addr, command, "
unknown_instruction");
            fout << temp_str;
        }
    } else {
        std::string temp_str = format("    %05x:\t%08x\t%s\n", addr, command, "
unknown_instruction");
        fout << temp_str;
    }
    }

    addr += 4;
}

return 0;
}

int parse_elf_file(std::ifstream& elf_file, std::string fout_name) {
    elf_file_header elf_header;

    if (!elf_file.read((char*)&elf_header, sizeof(elf_header)).good()) {
        std::cerr <<"Error: Could not read e_ident" << std::endl;
        return 1;
    }

    if (!(elf_header.e_ident[1] == 'E' && elf_header.e_ident[2] == 'L' &&
elf_header.e_ident[3] == 'F')) {
        std::cerr <<"Error: Wrong file format (not elf file)\n";
        return 1;
    }

    if ((elf_header.e_ident[4] != 1)) {
        std::cerr <<"Error: Wrong file format (work only with ELFCLASS32)\n";
        return 1;
    }

    if ((elf_header.e_ident[5] != 1)) {
        std::cerr <<"Error: Wrong file format (work only with ELFDATA2LSB (little
endian))\n";
        return 1;
    }

    if (elf_header.e_machine != 0xF3) {
        std::cerr <<"Error: Wrong file format (work only with EM_RISCV system)\n";
        return 1;
    }

```

```

    }

    std::vector<section_header> section_headers;

    elf_file.seekg(elf_header.e_shoff, std::ios::beg);

    for (int i = 0; i < elf_header.e_shnum; i++) {
        section_header temp;

        if (!elf_file.read((char*)&temp, sizeof(section_header)).good()) {
            std::cerr <<"Error: Could not read section header with number:" << i <<
std::endl;
            return 1;
        }

        section_headers.push_back(temp);
    }

    int index_of_symtab = -1;
    int index_of_text = -1;
    int index_of_strtab = -1;

    int help_address = section_headers[elf_header.e_shstrndx].sh_offset;

    for (int i = 0; i < elf_header.e_shnum; i++) {
        std::string name;

        elf_file.seekg(help_address + section_headers[i].sh_name, std::ios::beg);

        char char_buffer = ' ';
        while (char_buffer != '\0') {
            if (!elf_file.read((char *) &char_buffer, 1).good()) {
                std::cerr <<"Error: Could not read name of section header with
number:" << i << std::endl;
                return 1;
            }
            if (char_buffer != '\0') {
                name += char_buffer;
            }
        }

        if (name == ".symtab") {
            index_of_symtab = i;
        } else if (name == ".text") {
            index_of_text = i;
        } else if (name == ".strtab") {
            index_of_strtab = i;
        }
    }

    if (index_of_symtab == -1) {
        std::cerr <<"Error: Could not find symtab block" << std::endl;
        return 1;
    }

    if (index_of_text == -1) {
        std::cerr <<"Error: Could not find text block" << std::endl;
        return 1;
    }

    if (index_of_strtab == -1) {
        std::cerr <<"Error: Could not find strtab block" << std::endl;
    }

```

```

        return 1;
    }

    std::ofstream fout;
    fout.open(fout_name, std::ios::binary);

    if (!fout.is_open()) {
        std::cerr << "Error: Could not open output file" << std::endl;
        return 1;
    }

    std::vector<std::string> names_of_symbols;
    std::map<int, std::string> addresses_of_function_names;
    std::vector<elf_symbol> symtab;

    if (parse_symtab(elf_file, section_headers[index_of_symtab], symtab)) {
        std::cerr << "Error: Could not parse symtab" << std::endl;
        return 1;
    }

    if (get_names_of_symbols(symtab, elf_file,
        section_headers[index_of_strtab].sh_offset, names_of_symbols,
        addresses_of_function_names)) {
        std::cerr << "Error: Could not get names of symbol" << std::endl;
        return 1;
    }

    if (parse_and_write_text_or_disassembler(symtab, elf_file, fout,
        section_headers[index_of_text], addresses_of_function_names)) {
        std::cerr << "Error: Could not work with text" << std::endl;
        return 1;
    }

    if (write_symtab(symtab, elf_file, fout, names_of_symbols)) {
        std::cerr << "Error: Could not write symtab" << std::endl;
        return 1;
    }

    fout.close();

    return 0;
}

int main(int argc, char const* argv[]) {
    const char* fin_name = argv[1];
    const char* fout_name = argv[2];
    std::ifstream fin;
    fin.open(fin_name, std::ios::binary);

    if (!fin.is_open()) {
        std::cerr << "Error: Could not open elf file" << std::endl;
        return 0;
    }

    if (parse_elf_file(fin, fout_name) == 1) {
        std::cerr << "Error: Could not parse elf file" << std::endl;
        return 0;
    }

    fin.close();
    return 0;
};

```



