Лабораторная работа №4	10	2023
ISA	Лихтар Анна Викторовна	

Инструментарий и требования к работе: работа выполняется на C/C++ (C11 и новее / C++20), Python (3.11.5). Используемый язык программирования – C++, Версия – C++17.

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-riscv-likhhtar

Результат работы написанной программы

Вывод программы для теста из репозитория.

```
.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, 0x100
   10098:       fddff0ef            jal ra, 0x10074 <main>
   1009c:       00050593           addi a1, a0, 0
   100a0:       00a00893           addi a7, zero, 10
   100a4:       0ff0000f          fence iorw, iorw
   100a8:       00000073          ecall

000100ac        <mmul>:
   100ac:       00011f37            lui t5, 0x11
   100b0:       124f0513           addi a0, t5, 292
   100b4:       65450513           addi a0, a0, 1620
   100b8:       124f0f13           addi t5, t5, 292
   100bc:       e4018293           addi t0, gp, -448
   100c0:       fd018f93           addi t6, gp, -48
   100c4:       02800e93           addi t4, zero, 40

000100c8        <L2>:
   100c8:       fec50e13           addi t3, a0, -20
   100cc:       000f0313           addi t1, t5, 0
   100d0:       000f8893           addi a7, t6, 0
   100d4:       00000813           addi a6, zero, 0

000100d8        <L1>:
   100d8:       00088693           addi a3, a7, 0
   100dc:       000e0793           addi a5, t3, 0
   100e0:       00000613           addi a2, zero, 0
```

```
000100e4        <L0>:
   100e4:       00078703            lb a4, 0(a5)
   100e8:       00069583            lh a1, 0(a3)
   100ec:       00178793          addi a5, a5, 1
   100f0:       02868693          addi a3, a3, 40
   100f4:       02b70733           mul a4, a4, a1
   100f8:       00e60633           add a2, a2, a4
   100fc:       fea794e3           bne a5, a0, 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
```

Теория

RISC-V – это открытая архитектура процессора, которая разрабатывается с учетом современных требований к вычислительной мощности, энергоэффективности и масштабируемости.

RISC-V предоставляет модульную архитектуру, которая позволяет разработчикам создавать процессоры с различным количеством инструкций и уровнем сложности в зависимости от конкретных потребностей.

Набор команд RV32I и RV32M относятся к архитектуре RISC-V и представляют собой набор инструкций, которые определяют операции, которые может выполнять процессор, построенный на основе этой архитектуры.

1. RV32I:

Набор команд RV32I представляет базовый набор инструкций для 32-битной реализации RISC-V. Он включает в себя основные операции, такие как арифметические, логические, загрузка/выгрузка данных, переходы и управление памятью. Этот набор инструкций обеспечивает минимальный функционал для работы процессора и является основой для других расширений.

2. RV32M:

Набор команд RV32M представляет собой расширение базового набора инструкций RV32I и включает в себя инструкции для выполнения операций над целыми числами с плавающей запятой. Это расширение добавляет инструкции для умножения, деления и остатка от деления целых чисел.

В моей работе поддерживаются наборы команд RV32I, RV32M.

Каждый набор команд состоит из **инструкций** (Рис 1.)

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

RV32I Base Instruction Set

imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20|10:1|11|19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Рис. 1.

1. R-Type (Register Type):

Инструкции R-Type предназначены для операций, которые работают с регистрами процессора.

2. I-Type (Immediate Type):

Инструкции I-Type используют непосредственные значения в качестве операндов.

3. S-Type (Store Type):

Инструкции S-Type используются для записи данных в память. Они принимают два операнда: базовый адрес памяти и смещение, а затем записывают значение из регистра в указанное место в памяти.

4. B-Type (Branch Type):

Инструкции B-Type используются для выполнения условных переходов или ветвлений. Они проверяют условие и осуществляют переход к другому месту в программе в зависимости от результата проверки.

5. U-Type (Upper Immediate Type):

Инструкции U-Type используются для загрузки больших непосредственных значений (более 12 бит) в регистры.

6. J-Type (Jump Type):

Инструкции J-Type используются для выполнения безусловных переходов или прыжков.

Список команд и необходимую информацию для их определения я взяла по следующим ссылкам:

RV32I: https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#lui

RV32M: https://msyksphinz-self.github.io/riscv-isadoc/html/rvm.html

Opcode (Operation Code), Func7, Func3 – это части инструкции, которые определяют операцию, которую нужно выполнить.

RS1, RS2, RD – это регистры источника и регистр назначения.

Imm – константа.

Elf-файл – это стандартный формат исполняемых файлов, общий для многих операционных систем

Заголовок ELF-файла - это часть файла, которая содержит основную информацию о файле, такую как тип файла, архитектура процессора, точка входа программы и другие метаданные.

Я брала структуру заголовка на следующем сайте: https://refspecs.linuxbase.org/elf/gabi4+/ch4.eheader.html

Таблица заголовков программы (Program Header Table) - это часть ELF-файла, которая содержит информацию о различных сегментах программы.

Таблица заголовков секций (Section Header Table) - это часть ELF-файла, которая содержит информацию о различных секциях программы, таких как .text (код), .symtab (таблица символов, которая содержит информацию о всех глобальных и статических символах в программе), .strtab (таблица строк, которая содержит все имена символов, указанные в .symtab) и другие.

Структуру брала на следующем сайте: https://refspecs.linuxbase.org/elf/gabi4+/ch4.sheader.html

Описание работы написанного кода

```
typedef struct {
    unsigned char      e_mag[4];
    unsigned char      e_class;
    unsigned char      e_data;
    unsigned char      e_version1;
    unsigned char      e_osabi;
    unsigned char      e_abiversion;
    unsigned char      e_pad[7];
    unsigned short     e_type;
    unsigned short     e_machine;
    unsigned int       e_version2;
    unsigned int       e_entry;
    unsigned int       e_phoff;
    unsigned int       e_shoff;
    unsigned int       e_flags;
    unsigned short     e_ehsize;
    unsigned short     e_phentsize;
    unsigned short     e_phnum;
    unsigned short     e_shentsize;
    unsigned short     e_shnum;
    unsigned short     e_shstrndx;
} Elf32_Ehdr;

Elf32_Ehdr elf_header;
if(fin.read(reinterpret_cast<char *>(&elf_header),
sizeof(Elf32_Ehdr)).gcount() < sizeof (Elf32_Ehdr)) {
    cerr << "Wrong header size";
    return 1;
}
if(!check_header(&elf_header)) {
    cerr << "Wrong header";
    return 1;
}
```

Считываем header, проверяем, что размер равен размеру структуры, а также, что значения соответствуют в ТЗ (check_header).

```
bool check_header(Elf32_Ehdr* header) {
    if (header->e_mag[0] == 0x7f && header->e_mag[1] == 0x45
        && header->e_mag[2] == 0x4c && header->e_mag[3] == 0x46
        && header->e_class == 0x01 && header->e_data == 0x01
        && header->e_machine == 0xf3 && header->e_version1 == 0x01
        && header->e_version2 == 0x01) return true;
    return false;
}
```

Понять, с какими константами необходимо сравнивать, помогла ссылка https://habr.com/ru/articles/480642/.

```
typedef struct {
    unsigned int   sh_name;
    unsigned int   sh_type;
    unsigned int   sh_flags;
    unsigned int   sh_addr;
    unsigned int   sh_offset;
    unsigned int   sh_size;
    unsigned int   sh_link;
    unsigned int   sh_info;
    unsigned int   sh_addralign;
    unsigned int   sh_entsize;
} Elf32_Shdr;
```

```
fin.seekg(elf_header.e_shoff, std::ios::beg);
std::vector<Elf32_Shdr> section_headers(elf_header.e_shnum);
fin.read(reinterpret_cast<char*>(section_headers.data()), elf_header.e_shnum *
elf_header.e_shentsize);
```

Сначала находится section headers. Для этого сначала отступаем от начала файла на e_shoff (указатель на начало таблицы заголовков секций). Далее считывается elf_header.e_shnum * elf_header.e_shentsize битов (количество записей в таблице заголовков секций * размер таблицы заголовков секций).

```
fin.seekg(section_headers[elf_header.e_shstrndx].sh_offset, std::ios::beg);
std::vector<char>
section_names(section_headers[elf_header.e_shstrndx].sh_size);
fin.read(section_names.data(),
section_headers[elf_header.e_shstrndx].sh_size);
```

Затем ищем таблицу имен секций. В e_shstrndx хранится индекс таблицы заголовков разделов записи, связанной со строковой таблицей имен разделов. Поэтому сначала отступаем от начала файла на e_offset (смещение от начала файла, где лежат необходимые данные). Далее считываются имена секций.

```
Elf32_Shdr *symtab_section;
Elf32_Shdr *text_section;

for (int i = 1; i < elf_header.e_shnum; i++) {
    if(strcmp(&section_names[section_headers[i].sh_name], ".text") == 0)
text_section = &section_headers[i];
    if(strcmp(&section_names[section_headers[i].sh_name], ".symtab") == 0)
symtab_section = &section_headers[i];
}
```

Ищем указатель на секции text и symtab.

```
typedef struct {
    unsigned int   st_name;
    unsigned int   st_value;
    unsigned int   st_size;
    unsigned char  st_info;
    unsigned char  st_other;
    unsigned short     st_shndx;
} Elf32_Sym;

unsigned int symbols_size = symtab_section->sh_size / symtab_section
->sh_entsize;
auto* symbols = new Elf32_Sym[symbols_size];
fin.seekg(symtab_section->sh_offset, std::ios::beg);
fin.read(reinterpret_cast<char *>(symbols), symtab_section->sh_size);
```

Считываем таблицу символов.

```
char* symbols_names = new char[section_headers[symtab_section
->sh_link].sh_size];
fin.seekg(section_headers[symtab_section->sh_link].sh_offset, std::ios::beg);
fin.read(symbols_names, section_headers[symtab_section->sh_link].sh_size);
```

Таким же образом ищем имена символов.

На этом этапе уже можно перейти к рассмотрению вывода для .symtab.

```
void print_symtab(FILE * f, Elf32_Sym* symbols, Elf32_Shdr *symtab_section,
char *symbols_names) {
    fprintf(f, "\n.symtab\n");
    fprintf(f, "\nSymbol Value              Size Type     Bind     Vis
Index Name\n");
    for (int i = 0; i * symtab_section->sh_entsize < symtab_section->sh_size;
i++) {
        print_symbol(f, &symbols[i], i, symbols_names);
    }
}
```

Идем циклом по всем символам и вызываем следующую функцию вывода.

```
void print_symbol(FILE * f, Elf32_Sym *symbol, const int i, char
*symbols_names) {
    fprintf(f, "[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\n", i,
           symbol->st_value,
           symbol->st_size,
           get_type(ELF32_ST_TYPE(symbol->st_info)).c_str(),
           get_binds(ELF32_ST_BIND(symbol->st_info)).c_str(),
           get_visibility(ELF32_ST_VISIBILITY(symbol->st_other)).c_str(),
           get_index(symbol->st_shndx).c_str(),
           &symbols_names[symbol->st_name]);
}
int ELF32_ST_BIND(int i) {
    return i >> 4;
}

int ELF32_ST_TYPE(int i) {
    return ((i)&0xf);
}
```

```
int ELF32_ST_VISIBILITY(int o) {
    return  ((o)&0x3);
}
```
Эти три функции представляют собой маскирование и сдвиг битов входного целочисленного значения для извлечения конкретных полей, определенных в спецификации ELF.

1. ELF32_ST_BIND(int i): Эта функция выполняет операцию сдвига битов вправо на 4 позиции, чтобы извлечь значение, представляющее связывание символа.
2. ELF32_ST_TYPE(int i): Эта функция использует маску 0xf (бинарное представление 1111) для извлечения типа символа из входного значения.
3. ELF32_ST_VISIBILITY(int o): Эта функция также использует маску, в данном случае 0x3 (бинарное представление 11), чтобы извлечь значение видимости символа.

Логика функций get_type, get_binds, get_visibility состоит в switch по значениям. Информацию о этих данных взяла в следующем ресурсе https://refspecs.linuxbase.org/elf/gabi4+/ch4.sheader.html.

```
auto *texts = new short[text_section->sh_size << 1];
fin.seekg(text_section->sh_offset, std::ios::beg);
fin.read((char *)texts, text_section->sh_size);
```
Ищем тектовую часть elf.

```
std::map<unsigned int, std::string> make_names(Elf32_Sym * symbols, char*
symbols_names,  Elf32_Shdr *symtab_section) {
    std::map<unsigned int, std::string> names;
    for (int i = 0; i * symtab_section->sh_entsize < symtab_section->sh_size;
i++) {
        if(symbols[i].st_name != 0) names[symbols[i].st_value] =
&symbols_names[symbols[i].st_name];
    }
    return names;
}
```
Создаем структуру, которая будет удобна в дальнейшем. Функция make_names извлекает имена символов из ELF-файла и создает отображение адресов символов на их имена, используя std::map.

```
int l_num = 0;
std::vector<program> programs;
for(unsigned int i = 0, j = text_section->sh_addr; i * 4 < text_section-
>sh_size; i++, j += 4) {
    auto text1 = texts[i * 2];
    auto text2 = texts[i * 2 + 1];
    program pr = program_create(text1, text2);
    auto prog = get_prog(pr);
    programs.push_back(prog);
    int off = 0;
    if (prog.type == "J") {
        unsigned int offset = (((pr.data >> 21) & 0b1111111111) | (((pr.data
>> 20) & 0b1)  << 10)  |
                                ((pr.data >> 12) & 0b11111111) << 11) |
(((pr.data >> 31) & 0b1) << 19)) << 1;
        off = expen(j + offset, 0);
        if (symbol_addr_names.count(off) == 0) {
            symbol_addr_names[off] = "L" + std::to_string(l_num++);
        }
    }
    else if (pr.type == "B") {
        unsigned int offset = (((pr.data >> 8) & 0b1111) | (((pr.data >> 25) &
0b111111) << 4)  |
                                ((pr.data >> 7) & 0b1) << 10) | (((pr.data >>
31) & 0b1) << 11)) << 1;
        off = j + expen(offset, 1);
        if (symbol_addr_names.count(off) == 0) {
            symbol_addr_names[off] = "L" + std::to_string(l_num++);
        }
    }
}
```

Считываем инструкции. Функция program_create создает переменную типа program.

```
typedef struct {
    unsigned char func7;
    unsigned char rs2;
    unsigned char rs1;
    unsigned char func3;
    unsigned char rd;
    unsigned char opcode;
    std::string name;
    std::string type;
    unsigned char succ;
    unsigned char pred;
    unsigned char mid;
    unsigned int data;
} program;
```

```
program program_create(unsigned short text1, unsigned short text2) {
    program pr;
    pr.data = text1 | (text2 << 16);
    pr.opcode = (text1 & 0b1111111);
    pr.rd = (text1 >> 7) & 0b11111;
    pr.func3 = (text1 >> 12) & 0b111;
    pr.rs1 = ((text1 >> 15) & 0x1) | ((text2 & 0xf) << 1);
    pr.rs2 = (text2 >> 4) & 0b11111;
    pr.func7 = (text2 >> 9);
    pr.succ = (text2 >> 4) & 0b1111;
    pr.pred = (text2 >> 8) & 0b1111;
    pr.mid = text2 >> 12;
    pr.name = "invalid_operation";
    pr.type = "no_type";
    return pr;
}
```

Используются битовые операции для получения необходимого поля.

Далее вызывается функция get_prog, которая изменит поля name и type на подходящие. Эта функция вышла довольно длинная, т.к. состоит из switch и if и пытается покрыть все команды RV32I и RV32M. Все команды и как их распознать я узнала по ссылкам, которые указала выше. Также там используется функция get_reg. Данные для нее взяла здесь https://riscv.org/wp-content/uploads/2015/01/riscv-calling.pdf.

Проверяем значение переменной prog.type. Если оно равно "J", то выполняются следующие действия:

- Вычисляется значение переменной offset с помощью битовых сдвигов и побитовых операций над значением переменной pr.data.

- Значение переменной off устанавливается равным результату вызова функции expen с аргументами j + offset и 0.

- Проводится проверка, содержит ли контейнер symbol_addr_names элемент с ключом off. Если элемент с таким ключом отсутствует, то создается новый элемент в контейнере с ключом off, а его значением становится строка "L" + текущее значение переменной l_num, преобразованное в строку.

3. Если значение переменной pr.type равно "B", то выполняются следующие действия:

- Вычисляется значение переменной offset с помощью битовых сдвигов и побитовых операций над значением переменной pr.data.

- Значение переменной off устанавливается равным результату вызова функции expen с аргументами j + offset и 1.

- Снова делаем проверку на наличие названия метки.

```
int expen(unsigned int n, bool tw) {
    int res;
    if (tw) {
        // Получаем значение, игнорируя старшие 20 бит
        res = n & 0xFFF;
        // Если самый старший бит установлен, выполняем знаковое расширение
        if (n & (1 << 11)) {
            res |= 0xFFFFF000;  // Знаковое расширение
        }
    }
    else{
        // Получаем значение, игнорируя старшие 12 бит
        res = n & 0xFFFFF;

        // Если самый старший бит установлен, выполняем знаковое расширение
        if (n & (1 << 19)) {
            res |= 0xFFF00000;  // Знаковое расширение
        }
    }
    return res;
}


for (unsigned int i = 0, j = text_section->sh_addr; i * 4 < text_section
->sh_size; i++, j += 4) {
    auto pr = programs[i];
    print_prog(fout, j, programs[i], symbol_addr_names);
}
```

Выводим инструкции.

```
void print_prog(FILE *f, unsigned int addr, program &pr, std::map<unsigned
int, std::string> &symbol_addr_names) {
    if (pr.name == "invalid_operation") {
        fprintf(f, "   %05x:\t%08x\t%-7s\n", addr, pr.data, pr.name.c_str());
        return;
    }
    if (symbol_addr_names.count(addr) > 0) {
        fprintf(f, "\n%08x \t<%s>:\n", addr, symbol_addr_names[addr].c_str());
    }
    if (pr.type == "I") {
        i_print(f, addr, pr);
    }
    else if (pr.type == "R")  {
        fprintf(f, "   %05x:\t%08x\t%7s\t%s, %s, %s\n", addr, pr.data,
pr.name.c_str(),
                get_register(pr.rd).c_str(), get_register(pr.rs1).c_str(),
get_register(pr.rs2).c_str());
    }
    else if (pr.type == "S")  {
     fprintf(f, "   %05x:\t%08x\t%7s\t%s, %d(%s)\n", addr, pr.data,
pr.name.c_str(), get_register(pr.rs2).c_str(), expen(pr.rd + (pr.func7 << 5),
1),
get_register(pr.rs1).c_str());}
```

```
    else if (pr.type == "J")  {
        j_print(f, addr, pr, symbol_addr_names);
    }
    else if (pr.type == "B")    {
        b_print(f, addr, pr, symbol_addr_names);
    }
    else if (pr.type == "U") {
        fprintf(f, "   %05x:\t%08x\t%7s\t%s, 0x%x\n", addr, pr.data,
pr.name.c_str(),
         get_register(pr.rd).c_str(), expen(pr.data >> 12, 0));
    }
}
```

Если имя инструкции равно "invalid_operation", выводится следующая строка с данными data и "invalid_operation".

Если в symbol_addr_names присутствует ключ addr, выводится addr и имя.

В зависимости от типа инструкции, выводится соответствующая информация.

Если тип R, то выводим регистры rs1, rs2 и rd.

Если тип S, то выводим rs2, pr.rd + (pr.func7 << 5) и rs1.

Если тип U, то выводим addr, data, name, rd и (pr.data >> 12).

Если тип I

```
void i_print(FILE *f, unsigned int addr, program &pr) {
    if (pr.name == "jalr" || pr.name[0] == 'l') {
        fprintf(f, "   %05x:\t%08x\t%7s\t%s, %d(%s)\n", addr, pr.data,
pr.name.c_str(),
                get_register(pr.rd).c_str(), expen((pr.func7 << 5) |
(pr.rs2), 1), get_register(pr.rs1).c_str());
    }
    else if (pr.name == "fence") {
        fprintf(f, "   %05x:\t%08x\t%7s\t%s, %s\n", addr, pr.data,
pr.name.c_str(),
                fence(pr.pred).c_str(), fence(pr.succ).c_str());
    }
    else if (pr.opcode == 0b1110011) {
        if (pr.func3 != 0)
            fprintf(f, "   %05x:\t%08x\t%7s\t%s, %d, %s\n", addr, pr.data,
pr.name.c_str(),
                    get_register(pr.rd).c_str(), (pr.func7 << 5) |
(pr.rs2), get_register(pr.rs1).c_str());
        else
            fprintf(f, "   %05x:\t%08x\t%7s\n", addr, pr.data,
pr.name.c_str());

    }
    else {
        fprintf(f, "   %05x:\t%08x\t%7s\t%s, %s, %s\n", addr, pr.data,
pr.name.c_str(),
                get_register(pr.rd).c_str(), get_register(pr.rs1).c_str(),
std::to_string(expen_pr(pr)).c_str());
    }
}
```

При выполнении условия if (pr.name == "jalr" || pr.name[0] == 'l'), функция записывает в файл информацию о команде jalr или командах, начинающихся с символа 'l'. Функция записывает в файл адрес, данные, имя команды, регистр rd, значение выражения expen((pr.func7 << 5) | (pr.rs2), 1) и регистр rs1.

При выполнении условия else if (pr.name == "fence"), функция записывает в файл информацию о команде fence. Функция записывает в файл адрес, данные, имя команды, значение функции fence(pr.pred) и значение функции fence(pr.succ).

При выполнении условия else if (pr.opcode == 0b1110011), функция записывает в файл информацию о командах с опкодом 0b1110011. Внутри этого условия есть еще одно условие, которое проверяет значение pr.func3. Если значение pr.func3 не равно 0, то функция записывает в файл адрес, данные, имя команды, регистр rd, значение выражения (pr.func7 << 5) | (pr.rs2) и регистр rs1. Если значение pr.func3 равно 0, то функция записывает в файл только адрес, данные и имя команды.

При выполнении всех предыдущих условий, функция записывает в файл информацию о всех остальных командах. Записываются адрес, данные, имя команды, регистр rd, регистр rs1 и значение функции std::to_string(expen_pr(pr)).

Функция fence осуществляет парсинг для соответствующей операции.

```
std::string fence(unsigned char x) {
    std::string f;
    if (x & 8) f += "i";
    if (x & 4) f += "o";
    if (x & 2) f += "r";
    if (x & 1) f += "w";
    return f;
}
```

Если тип J

```
void j_print(FILE *f, unsigned int addr, program &pr, std::map<unsigned int,
std::string> &symbol_addr_names) {
    unsigned int x = (((pr.data >> 21) & 0b1111111111) | (((pr.data >> 20) &
0b1)  << 10)  |
                      (((pr.data >> 12) & 0b11111111) << 11) | (((pr.data >>
31) & 0b1) << 19)) << 1;


    fprintf(f, "   %05x:\t%08x\t%7s\t%s, 0x%x <%s>\n", addr, pr.data,
pr.name.c_str(),
            get_register(pr.rd).c_str(), expen(addr + x, 0),
symbol_addr_names[expen(addr + x, 0)].c_str());
}
```

Внутри функции определяется сдвиг x. Затем функция выводит addr, data, name, rd, expen(addr + x) и имя по этому адресу.

Если тип B

```
void b_print(FILE *f, unsigned int addr, program &pr, std::map<unsigned int,
std::string> &symbol_addr_names) {
    unsigned int x = (((pr.data >> 8) & 0b1111) | (((pr.data >> 25) &
0b111111) << 4)  |
                      (((pr.data >> 7) & 0b1) << 10) | (((pr.data >> 31) &
0b1) << 11)) << 1;

    fprintf(f, "   %05x:\t%08x\t%7s\t%s, %s, 0x%x, <%s>\n", addr, pr.data,
pr.name.c_str(),
            get_register(pr.rs1).c_str(), get_register(pr.rs2).c_str(),
            addr + expen(x, 1), symbol_addr_names[addr + expen(x,
1)].c_str());
}
```

Внутри функции определяется сдвиг x. Затем функция выводит rs1, rs2, addr + expen(x) и имя по этому адресу.