САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №_4__

«ISA. Ассемблер, дизассемблер»

Выполнил(а): Мавлютов Эрвин Акимович

Номер ИСУ: 334918

студ. гр. М3139

Санкт-Петербург

2021

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

Инструментарий и требования к работе: работа была выполнена на языке программирования Java.

Теоретическая часть

RISC-V – открытая и свободная система команд и процессорная архитектура на основе концепции RISC для микропроцессоров и микроконтроллеров. В архитектуре RISC-V имеется обязательное для реализации небольшое подмножество команд (набор инструкций I — Integer) и несколько стандартных опциональных расширений. Архитектура использует только модель little-endian — первый байт операнда в памяти соответствует наименее значащим битам значений регистрового операнда. Операции умножения, деления и вычисления остатка не входят в минимальный набор инструкций, а выделены в отдельное расширение (M — Multiply extension). Имеется ряд доводов в пользу разделения и данного набора на два отдельных (умножение и деление). Для наиболее часто используемых инструкций стандартизовано применение их аналогов в более компактной 16-битной кодировке (C — Compressed extension).

Elf-файл (Executable and Linkable Format) — формат двоичных файлов, используемый во многих современных UNIX-подобных операционных системах. Каждый elf-файл состоит из заголовка, таблицы заголовков программы, таблицы заголовков секций и таблиц секций. Исполняемый код в зашифрованном виде находится в секции .text. В секции .symtab хранится таблица символов, в секции .strtab — таблица строк, необходимые для расшифрования файла.

Практическая часть

Далее представлен код программы-транслятора, с помощью которой можно преобразовать машинный код в код на языке ассемблера. В программе поддерживаются только команды в формате RISC-V (т.е. elf-файлы, содержащие команды в формате RVC, не будут работать корректно). Программа принимает в качестве аргументов командной строки имена входного и выходного файла.

Входной файл представляет собой elf-файл. Программа читает его в бинарном виде, далее находит Header, читаются все секции заголовков. Читается каждая из секций .text и .symtab. В выходном файле представлены команды ассемблера в формате RISC-V.

В программе поддерживаются команды RV32-I, RV32-M. Необходимо использовать 32-битный elf-файл.

Листинг

Main.java

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;


public class Main {
   private static int[] bytes;
   private static FileWriter out;
   private static DataInputStream in;
   private static Map<String, Section> sections;


   public static void main(String[] args) {
       try {
           in = new DataInputStream(new
FileInputStream(args[0]));
           out = new FileWriter(args[1],
StandardCharsets.UTF_8);
           enterData();
           parseHeader();
           parseText();
           parseSymtab();
           in.close();
           out.close();
       } catch (IOException e) {
           System.out.println("Error after open files: " +
e.getMessage());
       }
   }


   private static void enterData() throws IOException {
       byte[] bytes1 = in.readAllBytes();
```

```
        bytes = new int[bytes1.length];
        for (int i = 0; i < bytes1.length; i++) {
            bytes[i] = ((bytes1[i] < 0) ? 256 : 0) + bytes1[i];
        }
    }

    private static void parseHeader() {
        int e_shoff = cnt(32, 4);
        int e_shentsiz = cnt(46, 2);
        int e_shnum = cnt(48, 2);
        int e_shstrndx = cnt(50, 2);
        int go = e_shoff + e_shentsiz * e_shstrndx;
        int sh_offset12 = cnt(go + 16, 4);
        sections = new HashMap<>();
        for (int j = 0, tmp = e_shoff; j < e_shnum; j++, tmp +=
40) {
            String name = getName(sh_offset12 + cnt(tmp, 4));
            int[] param = new int[10];
            for (int r = 0; r < 10; r++) {
                param[r] = cnt(tmp + 4 * r, 4);
            }
            if (name.length() == 0) {
                name = ".";
            }
            sections.put(name.substring(1), new Section(param));
        }
    }

    private static void parseText() throws IOException {
        out.write(".text\n");

        int ind = sections.get("text").getOffset();
        int num = sections.get("text").getSize() / 4;
```

```
        int addr = sections.get("text").getAddr();
        for (int i = 0; i < num; i++) {
            String[] ans = getAns(ind + 4 * i);
            try {
                out.write(String.format("%08x", addr + 4 * i));
                for (int j = 0; j < 5; j++) {
                    if (ans[j].length() > 0) {
                        out.write(String.format(" %s", ans[j]));
                        if (j == 2 || j == 3) {
                            out.write(", ");
                        }
                    }
                }
                out.write("\n");
            } catch (ArrayIndexOutOfBoundsException e) {
                //System.out.println(e.getMessage());
            }
        }
    }

    private static void parseSymtab() throws IOException {
        out.write("\n.symtab\n");
        int ind = sections.get("symtab").getOffset();
        int num = sections.get("symtab").getSize() / 16;
        out.write(String.format("%s %-15s %7s %-8s %-8s %-8s %6s
%s\n",
                "Symbol", "Value", "Size", "Type", "Bind",
"Vis", "Index", "Name"));
        for (int i = 0; i < num; i++) {
            int name = cnt(ind + i * 16, 4);
            int value = cnt(ind + i * 16 + 4, 4);
            int size = cnt(ind + i * 16 + 8, 4);
            StringBuilder sb = new StringBuilder();
```

```
            for (int j = 0; j < 4; j++) {
                sb.append(new StringBuilder(decToBin(bytes[ind +
i * 16 + 12 + j])).reverse());
            }
            int info = Integer.parseInt(new
StringBuilder(sb.substring(0, 4)).reverse().toString(), 2);
            int shndx = Integer.parseInt(new
StringBuilder(sb.substring(4, 8)).reverse().toString(), 2);
            int other = Integer.parseInt(new
StringBuilder(sb.substring(8)).reverse().toString(), 2);
            out.write(String.format("[%4d] 0x%-15X %5d %-8s %-8s
%-8s %6s %s\n",
                    i, value, size, ParserSymtab.getType(info),
ParserSymtab.getBind(shndx), ParserSymtab.getVis(other),
                    ParserSymtab.getIndex(other >> 8),
getName(name + sections.get("strtab").getOffset())));
        }
    }

    private static int cnt(int left, int num) {
        int ans = 0;
        for (int i = num - 1; i >= 0; i--) {
            ans = ans * 256 + bytes[left + i];
        }
        return ans;
    }

    private static String getName(int left) {
        StringBuilder sb = new StringBuilder();
        while (bytes[left] != 0) {
            sb.append((char) bytes[left++]);
        }
        return sb.toString();
    }
```

```
    private static String[] getAns(int left) {
        if (decToBin(bytes[left]).endsWith("11")) {
            return parseRiscV(left);
        }
        return parseRcv(left);
    }

    private static String[] parseRcv(int left) {
        throw new AssertionError("This function isn't work");
    }

    private static String[] parseRiscV(int left) {
        StringBuilder sb = new StringBuilder()
                .append(decToBin(bytes[left +
3])).append(decToBin(bytes[left + 2]))
                .append(decToBin(bytes[left +
1])).append(decToBin(bytes[left]));
        String opcode = sb.substring(25, 32);
        String rd = sb.substring(20, 25);
        String func3 = sb.substring(17, 20);
        String rs1 = sb.substring(12, 17);
        String rs2 = sb.substring(7, 12);
        String func7 = sb.substring(0, 7);

        //System.out.println(opcode + " " + func3 + " " +
func7);

        if (opcode.equals("0110011")) {
            return new String[]{"", ParserText.parseR(func7,
func3), reg(rd), reg(rs1), reg(rs2)};
        } else if (opcode.equals("0110111") ||
opcode.equals("0010111")) {
            int imm_u = Integer.parseInt(sb.substring(0, 20),
2);
```

```
            return new String[]{"", ParserText.parseU(opcode),
reg(rd), imm_u + ""};

        } else if
(sb.toString().equals("00000000000100000000000001110011")) {

            return new String[]{"", "EBREAK", "", "", ""};

        } else if
(sb.toString().equals("00000000000000000000000001110011")) {

            return new String[]{"", "ECALL", "", "", ""};

        } else if (opcode.equals("1100011")) {

            int imm_b = (int) Long.parseLong(

                    (sb.charAt(0) + "").repeat(20) +
sb.charAt(24) + sb.substring(1, 7) + sb.substring(20, 24) +
"0",2

            );

            return new String[]{"", ParserText.parseB(func3),
reg(rs1), reg(rs2), imm_b + ""};

        } else if (opcode.equals("0100011")) {

            int imm_s = (int) Long.parseLong(

                    (sb.charAt(0) + "").repeat(20) +
sb.substring(0, 7) + sb.substring(20, 25), 2

            );

            return new String[]{"", ParserText.parseS(func3),
"", reg(rs2), imm_s + ""};

        } else if (opcode.equals("0010011")) {

            try {

                int imm_i = (int) Long.parseLong((sb.charAt(0) +
"").repeat(20) + sb.substring(0, 12), 2);

                return new String[]{"",
ParserText.parseIArifm(func3), reg(rd), reg(rs1), imm_i + ""};

            } catch (AssertionError ignored) {

                return new String[]{"",
ParserText.parseISr(func3, func7), reg(rd), reg(rs1),
sb.substring(7, 11)};

            }

        } else if (opcode.equals("0000011")) {

            int imm_i = (int) Long.parseLong((sb.charAt(0) +
"").repeat(20) + sb.substring(0, 12), 2);
```

```
            return new String[]{"", ParserText.parseIL(func3),
"", reg(rd), imm_i + ""};
        } else if (opcode.equals("1101111")) {
            int imm_j = (int) Long.parseLong(
                    (sb.charAt(0) + "").repeat(12) +
sb.substring(12, 20)
                            + (sb.charAt(20) + "") +
sb.substring(1, 11) + "0", 2
            );
            return new String[]{"", ParserText.parseJ(),
reg(rd), "", imm_j + ""};
        } else if (opcode.equals("1100111")) {
            int imm_i = (int) Long.parseLong((sb.charAt(0) +
"").repeat(20) + sb.substring(0, 12), 2);
            return new String[]{"", "jalr", reg(rd), "", imm_i +
""};
        } else if (opcode.equals("1110011")) {
            return new String[]{"", ParserText.parseICsr(func3),
reg(rd), reg(sb.substring(0, 12)), reg(rs1)};
        } else {
            throw new AssertionError("Instruction not found: " +
sb);
        }
    }

    private static String reg(String a) {
        int reg = Integer.parseInt(a, 2);
        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) {
            return "t0";
        } else if (6 <= reg && reg <= 7) {
            String s = "t";
            s += (char) (reg - 5 + '0');
            return s;
        } else if (reg == 8) {
            return "s0";
        } else if (reg == 9) {
            return "s1";
        } else if (10 <= reg && reg <= 11) {
            String s = "a";
            s += (char) (reg - 10 + '0');
            return s;
        } else if (12 <= reg && reg <= 17) {
            String s = "a";
            s += (char) (reg - 10 + '0');
            return s;
        } else if (18 <= reg && reg <= 27) {
            String s = "s";
            s += (char) (reg - 16 + '0');
            return s;
        } else if (28 <= reg && reg <= 31) {
            String s = "t";
            s += (char) (reg - 25 + '0');
            return s;
        }
        return null;
    }

    private static String decToBin(int b) {
        StringBuilder sb = new StringBuilder();
```

```
        for (int i = 0; i < 8; i++) {
            sb.append(b % 2);
            b /= 2;
        }
        return sb.reverse().toString();
    }
}
```

ParserSymtab.java

```
public class ParserSymtab {
   public static String getIndex(int other) {
       return switch (other) {
           case 0 -> "UND";
           case 0xfff1 -> "ABS";
           case 0xff00 -> "LOPROC";
           case 0xff1f -> "HIPROC";
           case 0xfff2 -> "COMMON";
           case 0xffff -> "XINDEX";
           case 0xff3f -> "HIOS";
           case 0xff10 -> "LOOS";
           case 0xff01 -> "AFTER";
           default -> other + "";
       };
   }

   public static String getVis(int other) {
       other = (other % 4) / 2;
       return switch (other) {
           case 0 -> "DEFAULT";
           case 1 -> "HIDDEN";
           default -> throw new AssertionError("Unknown vis");
       };
   }
```

```
    public static String getBind(int shndx) {
        return switch (shndx) {
            case 0 -> "LOCAL";
            case 1 -> "GLOBAL";
            case 2 -> "WEAK";
            case 13 -> "LOPROC";
            case 15 -> "HIPROC";
            default -> throw new AssertionError("Unknown bind");
        };
    }

    public static String getType(int info) {
        return switch (info) {
            case 0 -> "NOTYPE";
            case 1 -> "OBJECT";
            case 2 -> "FUNC";
            case 3 -> "SECTION";
            case 4 -> "FILE";
            case 13 -> "LOPROC";
            case 15 -> "HIPROC";
            default -> throw new AssertionError("Unknown type");
        };
    }
}
```

ParserText.java

```
public class ParserText {
    public static String parseIL(String func3) {
        return switch (func3) {
            case "000" -> "lb";
            case "001" -> "lh";
            case "010" -> "lw";
            case "100" -> "lbu";
```

```
            case "101" -> "lhu";
            default -> throw new AssertionError("Unknown I-type
instruction");
        };
    }

    public static String parseICsr(String func3) {
        return switch (func3) {
            case "001" -> "csrrw";
            case "010" -> "csrrs";
            case "011" -> "csrrc";
            case "101" -> "csrrwi";
            case "110" -> "csrrsi";
            case "111" -> "csrrci";
            default -> throw new AssertionError("Unknown I-type
instruction");
        };
    }

    public static String parseJ() {
        return "jal";
    }

    public static String parseIArifm(String func3) {
        return switch (func3) {
            case "000" -> "addi";
            case "010" -> "slti";
            case "011" -> "sltiu";
            case "100" -> "xori";
            case "110" -> "ori";
            case "111" -> "andi";
            default -> throw new AssertionError("Unknown I-type
instruction");
        };
    }
```

```
   public static String parseISr(String func3, String func7) {
       return switch (func3) {
           case "001" -> "slli";
           case "101" -> switch (func7) {
               case "0000000" -> "srli";
               case "0100000" -> "srai";
               default -> throw new AssertionError("Unknown I-type
instruction");
           };
           default -> throw new AssertionError("Unknown I-type
instruction");
       };
   }

   public static String parseS(String func3) {
       return switch (func3) {
           case "000" -> "sb";
           case "001" -> "sh";
           case "010" -> "sw";
           default -> throw new AssertionError("Unknown S-type
instruction");
       };
   }

   public static String parseU(String opcode) {
       if (opcode.equals("0110111")) {
           return "lui";
       } else if (opcode.equals("0010111")) {
           return "auipc";
       }
       throw new AssertionError("Unknown U-type instruction");
   }

   public static String parseB(String func3) {
```

```
        return switch (func3) {
            case "000" -> "beq";
            case "001" -> "bne";
            case "100" -> "blt";
            case "101" -> "bge";
            case "110" -> "bltu";
            case "111" -> "bgeu";
            default -> throw new AssertionError("Unknown B-type
instruction");
        };
    }

    public static String parseR(String func7, String func3) {
        if (check(func7, "0000000", func3, "000")) {
            return "add";
        } else if (check(func7, "0100000", func3, "000")) {
            return "sub";
        } else if (check(func7, "0000000", func3, "001")) {
            return "sll";
        } else if (check(func7, "0000000", func3, "010")) {
            return "slt";
        } else if (check(func7, "0000000", func3, "011")) {
            return "sltu";
        } else if (check(func7, "0000000", func3, "100")) {
            return "xor";
        } else if (check(func7, "0000000", func3, "101")) {
            return "srl";
        } else if (check(func7, "0100000", func3, "101")) {
            return "sra";
        } else if (check(func7, "0000000", func3, "110")) {
            return "or";
        } else if (check(func7, "0000000", func3, "111")) {
            return "and";
        } else if (check(func7, "0000001", func3, "000")) {
```

```
            return "mul";
        } else if (check(func7, "0000001", func3, "001")) {
            return "mulh";
        } else if (check(func7, "0000001", func3, "010")) {
            return "mulhsu";
        } else if (check(func7, "0000001", func3, "011")) {
            return "mulhu";
        } else if (check(func7, "0000001", func3, "100")) {
            return "div";
        } else if (check(func7, "0000001", func3, "101")) {
            return "divu";
        } else if (check(func7, "0000001", func3, "110")) {
            return "rem";
        } else if (check(func7, "0000001", func3, "111")) {
            return "remu";
        }
        throw new AssertionError("Unknown R-type instruction");
    }

    private static boolean check(String func7, String eqFunc7, String
func3, String eqFunc3) {
        return func7.equals(eqFunc7) && func3.equals(eqFunc3);
    }
}
```

Section.java

```
public class Section {

    private final int name, type, flags, addr, offset, size, link, info,
addralign, entsize;


    public Section(int[] param) {
        name = param[0];
        type = param[1];
        flags = param[2];
        addr = param[3];
```

```
        offset = param[4];
        size = param[5];
        link = param[6];
        info = param[7];
        addralign = param[8];
        entsize = param[9];
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(name).append(" ").append(type).append("
").append(flags).append(" ").append(addr).append(" ")
                .append(offset).append(" ").append(size).append("
").append(link).append(" ").append(info).append(" ")
                .append(addralign).append(" ").append(entsize);
        return sb.toString();
    }

    public int getName() {
        return name;
    }

    public int getType() {
        return type;
    }

    public int getFlags() {
        return flags;
    }

    public int getAddr() {
        return addr;
    }
```

```
    public int getOffset() {
        return offset;
    }

    public int getSize() {
        return size;
    }

    public int getLink() {
        return link;
    }

    public int getInfo() {
        return info;
    }

    public int getAddralign() {
        return addralign;
    }

    public int getEntsize() {
        return entsize;
    }
}
```