САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

 Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 4

**«ISA. Ассемблер, дизассемблер»**

Выполнил: Тяпкин Пётр Алексеевич

Номер ИСУ: 334844

студ. гр. M3138

Санкт-Петербург

2021

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

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

**Теоретическая часть.**

ELF — это сокращение от Executable and Linkable Format (формат исполняемых и связываемых файлов) и определяет структуру бинарных файлов, библиотек, и файлов ядра (core files). Спецификация формата позволяет операционной системе корректно интерпретировать содержащиеся в файле машинные команды. Файл ELF, как правило, является выходным файлом компилятора или линкера и имеет двоичный формат. С помощью подходящих инструментов он может быть проанализирован и изучен.

Структура ELF файла (см. рисунок №1):

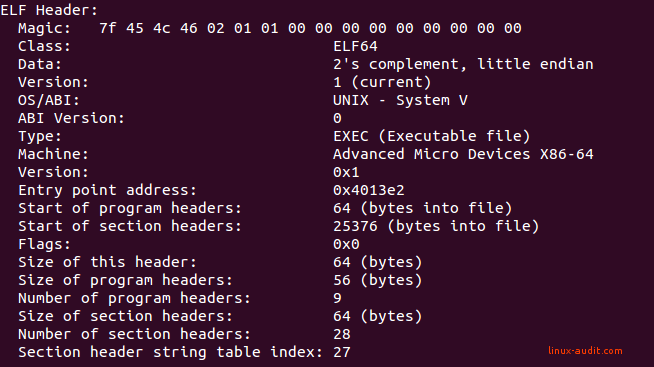


Рисунок № 1 – Структура ELF файла

1. Заголовок файла (ELF Header) имеет фиксированное расположение в начале файла и содержит общее описание структуры файла и его основные характеристики, такие как: тип, версия формата, архитектура процессора, виртуальный адрес точки входа, размеры и смещения остальных частей файла. Заголовок имеет размер 52 байта для 32-битных файлов.
2. Таблицы программных заголовков (program header table), которая служит для описания сегментов файла. Данная таблица может располагаться в любом месте файла, её местоположение (смещение относительно начала файла) описывается в поле e\_phoff заголовка ELF.
3. Таблица заголовков секций (section header table), которая характеризует секции файла. Секция состоит из набора подряд идущих байтов.

Все названия секций находятся в «.shstrtab»(таблица строк заголовок секций). Чтобы получить имя секции нужно прочитать с начала ссылки до первого нулевого байта.

Когда мы найдем секцию «.text» перейдем в ее offset и считаем все команды которые там находятся. Команда представляют собой 4 байта и формируются определенным образом.

RISC-V является архитектурой с открытым набором команд (ISA) основанной на принципах компьютера с сокращённым набором команд (RISC). Описание RISC-V включает сравнительно небольшое число стандартных инструкций, около 50 штук, многие из которых были типичны ещё для ранних RISC-I 1980 года. Стандартные расширения (M, A, F и D) расширяют набор на 53 инструкции, сжатый формат C определяет 34 команды. Используется 6 типов кодирования инструкций (форматов)(см рисунок №3).

RISC-V является little-endian архитектурой. Одна из главных особенностей архитектуры RISC-V это её расширяемость, что позволяет создавать разнообразные реализации. Минимальная спецификация архитектуры RISC-V включает в себя только инструкции для записи, сохранения, переходов и целочисленной арифметики. Код инструкций не зависит от размера регистра, минимальная спецификация обозначается RV32I, RV64I или RV128I для размеров операнда 4, 8 или 16 байт соответственно.

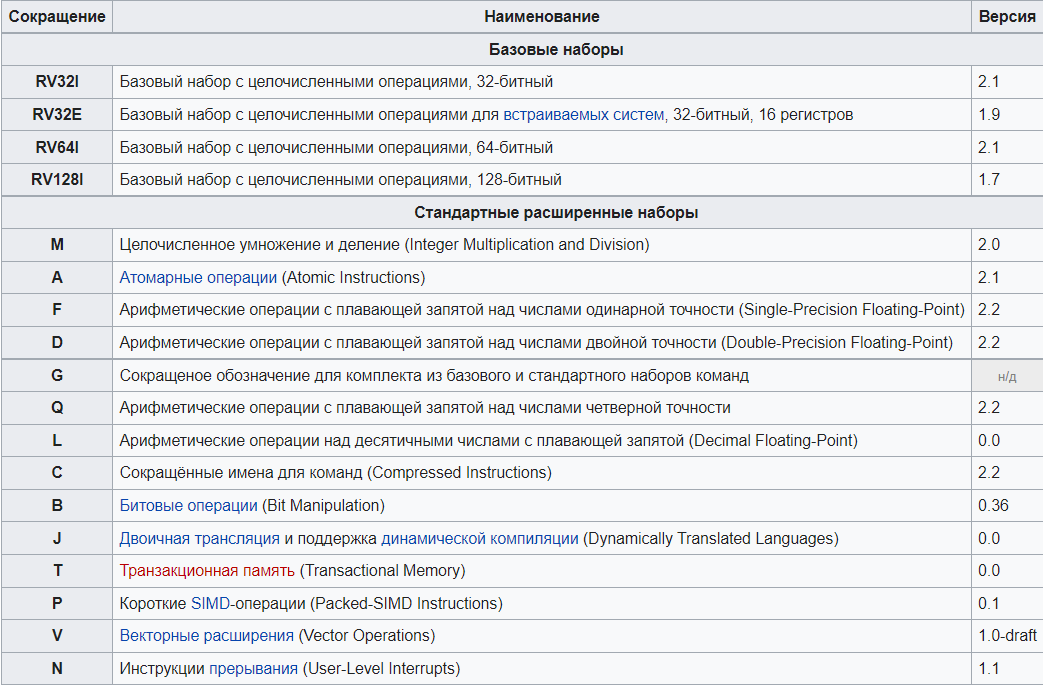


Рисунок № 3 – Список наборов команд RISC-V

Рассмотрим, как устроены команды, которые нам нужно уметь обрабатывать – RV32I (см рисунок № 4) и RV32M (см. рисунок № 5).

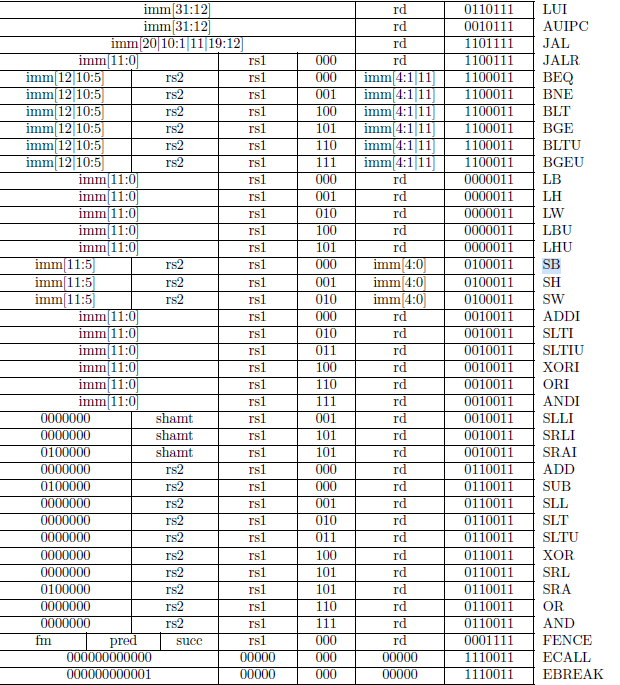


Рисунок № 4 – устройство команд RV32I

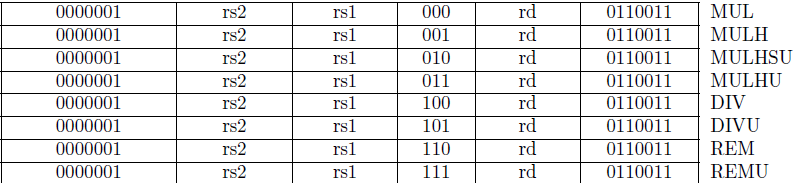


Рисунок № 5 – устройство команд RV32M

* rs1 — номер регистра в котором находится первый операнд
* rs2 — номер регистра в котором находится второй операнд
* rd — номер регистра в который будет записан результат
* immediate(imm) – аргумент функции

**Листинг кода**

OpenJDK 64-Bit Server VM Temurin-16.0.2+7 (build 16.0.2+7, mixed mode, sharing)

**Parser.java**

import java.io.\*;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Objects;

public class Parser {

public static boolean checkElf(final byte[] input) {

return (input[0] == 0x7f && input[1] == 0x45 && input[2] == 0x4c && input[3] == 0x46);

}

public static boolean checkElf32(final byte[] input) {

return (input[4] == 0x1);

}

public static boolean checkLittleEndian(final byte[] input) {

return (input[5] == 0x1);

}

public static boolean checkRiscv(final byte[] input) {

return ((input[18] + 256) == 0xf3);

}

public static void checkFormat(final byte[] input) {

if (input.length < 4) {

System.out.println("Incorrect input data");

System.exit(0);

}

if (!checkElf(input)) {

System.out.println("Input file not in elf format");

System.exit(0);

}

if (!checkElf32(input)) {

System.out.println("Input file not in elf32 format");

System.exit(0);

}

if (!checkLittleEndian(input)) {

System.out.println("Encoding is not Little Endian");

System.exit(0);

}

if (!checkRiscv(input)) {

System.out.println("Elf-file is not RISC-V");

System.exit(0);

}

}

public static int getInt4(final int x1, final int x2, final int x3, final int x4) {

return (x4 << 24) + (x3 << 16) + (x2 << 8) + x1;

}

public static int getInt2(final int x1, final int x2) {

return (x2 << 8) + x1;

}

public static int getE\_shoff(final int[] bytes) { // 32 - 35

return getInt4(bytes[32], bytes[33], bytes[34], bytes[35]);

}

public static int getE\_shnum(final int[] bytes) { // 48 - 49

return getInt2(bytes[48], bytes[49]);

}

public static int getE\_shstrndx(final int[] bytes) { // 50 - 51

return getInt2(bytes[50], bytes[51]);

}

public static int getShstrtab(final int e\_shoff, final int e\_shstrndx) {

return (e\_shoff + e\_shstrndx \* 40);

}

public static int getOffset(final int[] bytes, final int pos) {

return getInt4(bytes[pos + 16], bytes[pos + 17], bytes[pos + 18], bytes[pos + 19]);

}

public static int getSz(final int[] bytes, final int pos) {

return getInt4(bytes[pos + 20], bytes[pos + 21], bytes[pos + 22], bytes[pos + 23]);

}

public static int getIndex(final int[] bytes, final int offsetShstrtab, final String cur) {

StringBuilder sb = new StringBuilder();

int ans = -1;

for (int i = offsetShstrtab; i < bytes.length; i++) {

if (bytes[i] == 0) {

String now = sb.toString();

sb.setLength(0);

if (now.equals(cur)) {

ans = i - now.length();

break;

}

} else {

sb.append((char) bytes[i]);

}

}

return ans;

}

public static String getName(final int[] bytes, final int idName, final int offsetStrtab) {

StringBuilder sb = new StringBuilder();

int start = idName + offsetStrtab;

while (start < bytes.length && bytes[start] != 0) {

sb.append((char) bytes[start++]);

}

return sb.toString();

}

public static Symtab getSymtab(final int[] bytes, final int pos, final int offsetStrtab) {

int idName = getInt4(bytes[pos], bytes[pos + 1], bytes[pos + 2], bytes[pos + 3]);

String name = getName(bytes, idName, offsetStrtab);

int value = getInt4(bytes[pos + 4], bytes[pos + 5], bytes[pos + 6], bytes[pos + 7]);

int size = getInt4(bytes[pos + 8], bytes[pos + 9], bytes[pos + 10], bytes[pos + 11]);

int info = bytes[pos + 12];

int other = bytes[pos + 13];

int shndx = getInt2(bytes[pos + 14], bytes[pos + 15]);

return new Symtab(name, value, size, info, other, shndx);

}

public static int getaddress(final int[] bytes, final int id) {

return getInt4(bytes[id + 12], bytes[id + 13], bytes[id + 14], bytes[id + 15]);

}

public static void printAns(final Symtab[] tab, final ArrayList<Command> commands, final String output) {

try (FileWriter writer = new FileWriter(output)) {

writer.write(".symtab" + System.lineSeparator());

writer.write("Symbol Value Size Type Bind Vis Index Name" + System.lineSeparator());

for (int i = 0; i < tab.length; i++) {

writer.write(String.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s %s\n",

i, tab[i].getValue(), tab[i].getSize(), tab[i].getType(), tab[i].getBind(),

tab[i].getVis(), tab[i].getIndex(), tab[i].getName()));

writer.write(System.lineSeparator());

}

writer.write(".text" + System.lineSeparator());

for (int i = 0; i < commands.size(); i++) {

writer.write(commands.get(i).toString() + System.lineSeparator());

}

} catch (FileNotFoundException e) {

System.out.println("Output file does not exist " + e.getMessage());

} catch (IOException e) {

System.out.println("File I/O error! " + e.getMessage());

}

}

public static void main(String[] args) {

try (InputStream reader = new FileInputStream(args[0])) {

byte[] input = reader.readAllBytes();

checkFormat(input);

int length = input.length;

int[] bytes = new int[length];

for (int i = 0; i < length; i++) {

bytes[i] = (int) (input[i] & 0xFF);

}

int e\_shoff = getE\_shoff(bytes);

int e\_shnum = getE\_shnum(bytes);

int e\_shstrndx = getE\_shstrndx(bytes);

int shstrtab = getShstrtab(e\_shoff, e\_shstrndx);

int offsetShstrtab = getOffset(bytes, shstrtab);

int idText = getIndex(bytes, offsetShstrtab, ".text") - offsetShstrtab;

int idSymtab = getIndex(bytes, offsetShstrtab, ".symtab") - offsetShstrtab;

int idStrtab = getIndex(bytes, offsetShstrtab, ".strtab") - offsetShstrtab;

int e\_shoffLen = e\_shoff + 40 \* e\_shnum;

int posText = -1, posSymtab = -1, posStrtab = -1;

for (int i = e\_shoff; i < e\_shoffLen; i += 40) {

int id = getInt4(bytes[i], bytes[i + 1], bytes[i + 2], bytes[i + 3]);

if (id == idText) {

posText = i;

}

if (id == idSymtab) {

posSymtab = i;

}

if (id == idStrtab) {

posStrtab = i;

}

}

// parse symtab

int offsetSymtab = getOffset(bytes, posSymtab);

int szSymtab = getSz(bytes, posSymtab);

Symtab[] tab = new Symtab[szSymtab >> 4];

for (int i = offsetSymtab; i < offsetSymtab + szSymtab; i += 16) {

tab[(i - offsetSymtab) / 16] = getSymtab(bytes, i, getOffset(bytes, posStrtab));

}

// parse Text

int offsetText = getOffset(bytes, posText);

int szText = getSz(bytes, posText);

int addressText = getaddress(bytes, posText);

ArrayList<Command> commands = new ArrayList<>();

HashMap<Integer, String> labels = new HashMap<>();

for (int i = offsetText; i < offsetText + szText; i += 4) {

parseCommand4(labels, addressText, addressText + i - offsetText,

(long) bytes[i] + ((long) bytes[i + 1] << 8) + ((long) bytes[i + 2] << 16) + ((long) bytes[i + 3] << 24),

tab, commands);

}

printAns(tab, commands, args[1]);

} catch (FileNotFoundException e) {

System.out.println("Input file does not exist " + e.getMessage());

} catch (IOException e) {

System.out.println("File I/O error! " + e.getMessage());

}

}

static int cntLoc = 0;

public static String getRegister(final int id) {

return switch (id) {

case 0 -> "zero";

case 1 -> "ra";

case 2 -> "sp";

case 3 -> "gp";

case 4 -> "tp";

case 5 -> "t0";

case 6 -> "t1";

case 7 -> "t2";

case 8 -> "s0";

case 9 -> "s1";

case 10 -> "a0";

case 11 -> "a1";

case 12 -> "a2";

case 13 -> "a3";

case 14 -> "a4";

case 15 -> "a5";

case 16 -> "a6";

case 17 -> "a7";

case 18 -> "s2";

case 19 -> "s3";

case 20 -> "s4";

case 21 -> "s5";

case 22 -> "s6";

case 23 -> "s7";

case 24 -> "s8";

case 25 -> "s9";

case 26 -> "s10";

case 27 -> "s11";

case 28 -> "t3";

case 29 -> "t4";

case 30 -> "t5";

case 31 -> "t6";

default -> "unknown\_register";

};

}

public static int getFromItoJ(final long command, final int l, final int r) {

int ans = 0;

for (int i = l; i <= r; i++) {

if ((command & (1L << i)) == (1L << i)) {

ans += (1 << (i - l));

}

}

return ans;

}

public static void parseU(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rd = getFromItoJ(command, 7, 11);

int imm = getFromItoJ(command, 12, 31);

commands.add(new Command(String.format("%s %s, %d", commandName, getRegister(rd), imm), address, label, isLabel));

}

public static void parseI(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rd = getFromItoJ(command, 7, 11);

int rs1 = getFromItoJ(command, 15, 19);

int imm = getFromItoJ(command, 20, 31);

commands.add(new Command(

String.format("%s %s, %d(%s)", commandName, getRegister(rd), imm, getRegister(rs1)),

address, label, isLabel));

}

public static void parseI1(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rd = getFromItoJ(command, 7, 11);

int rs1 = getFromItoJ(command, 15, 19);

int imm = getFromItoJ(command, 20, 31);

commands.add(new Command(

String.format("%s %s, %s, %s", commandName, getRegister(rd), getRegister(rs1), imm),

address, label, isLabel));

}

public static void parseRV32MOrRV32IR(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rd = getFromItoJ(command, 7, 11);

int rs1 = getFromItoJ(command, 15, 19);

int rs2 = getFromItoJ(command, 20, 24);

commands.add(new Command(

String.format("%s %s, %s, %s", commandName, getRegister(rd), getRegister(rs1), getRegister(rs2)),

address, label, isLabel));

}

public static void parseS(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rs1 = getFromItoJ(command, 15, 19);

int rs2 = getFromItoJ(command, 20, 24);

int imm = getFromItoJ(command, 7, 11) + (getFromItoJ(command, 25, 31) << 5);

if (getFromItoJ(command, 25, 25) == 1) {

imm -= (1 << 13);

}

commands.add(new Command(

String.format("%s %s, %d(%s)", commandName, getRegister(rs1), imm, getRegister(rs2)),

address, label, isLabel));

}

public static void parseRWithShamt(final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel) {

int rd = getFromItoJ(command, 7, 11);

int rs1 = getFromItoJ(command, 15, 19);

int shamt = getFromItoJ(command, 20, 24);

commands.add(new Command(

String.format("%s %s, %s %d", commandName, getRegister(rd), getRegister(rs1), shamt),

address, label, isLabel));

}

public static void parseB(final int firstAddress, final int address, final String commandName, final long command, ArrayList<Command> commands,

final String label, final boolean isLabel, HashMap<Integer, String> labels, final Symtab[] tab) {

int rs1 = getFromItoJ(command, 15, 19);

int rs2 = getFromItoJ(command, 20, 24);

int imm = (getFromItoJ(command, 31, 31) << 12) + (getFromItoJ(command, 25, 30) << 5)

+ (getFromItoJ(command, 7, 7) << 11) + (getFromItoJ(command, 8, 11) << 1);

commands.add(new Command(

String.format("%s %s, %s, %s", commandName, getRegister(rs1), getRegister(rs2),

getLabel(firstAddress, address, imm + address, commands, labels, tab)), address, label, isLabel));

}

public static String checkLabel(final int address, final Symtab[] tab, HashMap<Integer, String> labels) {

if (labels.containsKey(address)) {

return labels.get(address);

}

for (Symtab cur : tab) {

if (cur.getValue() == address && cur.getType().equals("FUNC")) {

return cur.getName();

}

}

return "";

}

public static String newLabel(final int cnt) {

return (String.format("%s\_%05x", "LOC", cnt));

}

public static String getLabel(final int firstAddress, final int address, final int addressLabel, final ArrayList<Command> commands,

HashMap<Integer, String> labels, final Symtab[] tab) {

String label = "";

if (addressLabel < address) {

int id = (addressLabel - firstAddress) / 4;

Command command = commands.get(id);

if (!command.isLabel) {

command.isLabel = true;

command.label = newLabel(cntLoc++);

}

label = command.label;

}

if (label.equals("")) {

String cur = checkLabel(addressLabel, tab, labels);

if (cur.equals("")) {

cur = newLabel(cntLoc++);

}

labels.put(addressLabel, cur);

label = cur;

}

return label;

}

public static void parseCommand4(HashMap<Integer, String> labels, final int firstAddress, final int address,

final long command, final Symtab[] tab, ArrayList<Command> commands) {

int opcode = getFromItoJ(command, 0, 6);

String label = checkLabel(address, tab, labels);

boolean isLabel = !Objects.equals(label, "");

if (opcode == 0b0110111) {

parseU(address, "lui", command, commands, label, isLabel); // LUI U-type

return;

}

if (opcode == 0b0010111) {

parseU(address, "auipc", command, commands, label, isLabel); // AUIPC U-type

return;

}

if (opcode == 0b1101111) { // JAL J-type

int rd = getFromItoJ(command, 7, 11);

int imm = (getFromItoJ(command, 31, 31) << 20) + (getFromItoJ(command, 21, 30) << 1)

+ (getFromItoJ(command, 20, 20) << 11) + (getFromItoJ(command, 12, 19) << 12);

// commands.add(new Command(

// String.format("%s %s, %d", "jal", getLabel(imm + address), imm),

// address, label, isLabel));

return;

}

if (opcode == 0b1100111) { // JALR

//

return;

}

if (opcode == 0b0000011) { // I-type

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseI(address, "lb", command, commands, label, isLabel); // LB

return;

case 0b0010000:

parseI(address, "lh", command, commands, label, isLabel); // LH

return;

case 0b0100000:

parseI(address, "lw", command, commands, label, isLabel); // LW

return;

case 0b1000000:

parseI(address, "lbu", command, commands, label, isLabel); // LBU

return;

case 0b1010000:

parseI(address, "lhu", command, commands, label, isLabel); // LHU

return;

}

}

if (opcode == 0b0110011) { // R-type

switch (getFromItoJ(command, 25, 31)) {

case 0b0000001: // RV32M Standard Extension

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseRV32MOrRV32IR(address, "mul", command, commands, label, isLabel); // MUL

return;

case 0b0010000:

parseRV32MOrRV32IR(address, "mulh", command, commands, label, isLabel); // MULH

return;

case 0b0100000:

parseRV32MOrRV32IR(address, "mulhsu", command, commands, label, isLabel); // MULHSU

return;

case 0b0110000:

parseRV32MOrRV32IR(address, "mulhu", command, commands, label, isLabel); // MULHU

return;

case 0b1000000:

parseRV32MOrRV32IR(address, "div", command, commands, label, isLabel); // DIV

return;

case 0b1010000:

parseRV32MOrRV32IR(address, "divu", command, commands, label, isLabel); // DIVU

return;

case 0b1100000:

parseRV32MOrRV32IR(address, "rem", command, commands, label, isLabel); // REM

return;

case 0b1110000:

parseRV32MOrRV32IR(address, "remu", command, commands, label, isLabel); // REMU

return;

}

case 0b0000000:

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseRV32MOrRV32IR(address, "add", command, commands, label, isLabel); // ADD

return;

case 0b0010000:

parseRV32MOrRV32IR(address, "sll", command, commands, label, isLabel); // SLL

return;

case 0b0100000:

parseRV32MOrRV32IR(address, "slt", command, commands, label, isLabel); // SLT

return;

case 0b0110000:

parseRV32MOrRV32IR(address, "sltu", command, commands, label, isLabel); // SLTU

return;

case 0b1000000:

parseRV32MOrRV32IR(address, "xor", command, commands, label, isLabel); // XOR

return;

case 0b1010000:

parseRV32MOrRV32IR(address, "srl", command, commands, label, isLabel); // SRL

return;

case 0b1100000:

parseRV32MOrRV32IR(address, "or", command, commands, label, isLabel); // OR

return;

case 0b1110000:

parseRV32MOrRV32IR(address, "and", command, commands, label, isLabel); // AND

return;

}

case 0b0100000:

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseRV32MOrRV32IR(address, "sub", command, commands, label, isLabel); // SUB

return;

case 0b1010000:

parseRV32MOrRV32IR(address, "sra", command, commands, label, isLabel); // SRA

return;

}

}

}

if (opcode == 0b0100011) { // S -type

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseS(address, "sb", command, commands, label, isLabel); // SB

return;

case 0b0010000:

parseS(address, "sh", command, commands, label, isLabel); // SH

return;

case 0b0100000:

parseS(address, "sw", command, commands, label, isLabel); // SW

return;

}

}

if (opcode == 0b1100011) { // B-type

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseB(firstAddress, address, "beq", command, commands, label, isLabel, labels, tab); // BEQ

return;

case 0b0010000:

parseB(firstAddress, address, "bne", command, commands, label, isLabel, labels, tab); // BNE

return;

case 0b1000000:

parseB(firstAddress, address, "blt", command, commands, label, isLabel, labels, tab); // BLT

return;

case 0b1010000:

parseB(firstAddress, address, "bge", command, commands, label, isLabel, labels, tab); // BGE

return;

case 0b1100000:

parseB(firstAddress, address, "bltu", command, commands, label, isLabel, labels, tab); // BLTU

return;

case 0b1110000:

parseB(firstAddress, address, "bgeu", command, commands, label, isLabel, labels, tab); // BGEU

return;

}

}

if (opcode == 0b0010011) {

switch (getFromItoJ(command, 12, 14)) {

case 0b0000000:

parseI1(address, "addi", command, commands, label, isLabel); // ADDI

return;

case 0b0010000:

parseRWithShamt(address, "slli", command, commands, label, isLabel); // SLLI

return;

case 0b0100000:

parseI1(address, "slti", command, commands, label, isLabel); // SLTI

return;

case 0b0110000:

parseI1(address, "sltiu", command, commands, label, isLabel); // SLTIU

return;

case 0b1000000: // XORI

parseI1(address, "xori", command, commands, label, isLabel); // SLTIU

return;

case 0b1010000:

switch (getFromItoJ(command, 25, 31)) {

case 0b0000000:

parseRWithShamt(address, "srli", command, commands, label, isLabel); // SRLI

return;

case 0b0100000:

parseRWithShamt(address, "srai", command, commands, label, isLabel); // SRAI

return;

}

case 0b1100000:

parseI1(address, "ori", command, commands, label, isLabel); // ORI

return;

case 0b1110000:

parseI1(address, "andi", command, commands, label, isLabel); // ANDI

return;

}

}

if (opcode == 0b111001) {

switch (getFromItoJ(command, 20, 31)) {

case 0:

commands.add(new Command(String.format("%s", "ecall"), address, label, isLabel)); // ECALL

return;

case 1:

commands.add(new Command(String.format("%s", "ebreak"), address, label, isLabel)); // EBREAK

return;

}

}

System.out.println("unknown\_command");

}

}

**Command.java**

public class Command {

final String command;

final int address;

String label;

boolean isLabel = false;

public Command(String command, int address, String label, boolean isLabel) {

this.command = command;

this.address = address;

this.label = label;

this.isLabel = isLabel;

}

@Override

public String toString() {

return String.format("%08x %10s: %s\n", address, (isLabel ? label : ""), command);

}

}

**Symtab.java**

public class Symtab {

String name;

int value, size, info, other, shndx;

int type, bind, vis, index;

public Symtab(String name, int value, int size, int info, int other, int shndx) {

this.name = name;

this.value = value;

this.size = size;

this.info = info;

this.other = other;

this.shndx = shndx;

this.type = info % 16;

this.bind = info / 16;

this.vis = other;

this.index = shndx;

}

public String getName() {

return name;

}

public int getValue() {

return value;

}

public int getSize() {

return size;

}

public String getIndex() {

return switch (index) {

case (0) -> "UNDEF";

case (0xff00) -> "LORESERVE";

case (0xff1f) -> "HIPROC";

case (0xff20) -> "LOOS";

case (0xff3f) -> "HIOS";

case (0xfff1) -> "ABS";

case (0xfff2) -> "COMMON";

case (0xffff) -> "XINDEX";

default -> Integer.valueOf(index).toString();

};

}

public String getBind() {

return switch (bind) {

case (0) -> "LOCAL";

case (1) -> "GLOBAL";

case (2) -> "WEAK";

case (10) -> "LOOS";

case (12) -> "HIOS";

case (13) -> "LOPROC";

case (15) -> "HIPROC";

default -> "ERROR";

};

}

public String getType() {

return switch (type) {

case (0) -> "NOTYPE";

case (1) -> "OBJECT";

case (2) -> "FUNC";

case (3) -> "SECTION";

case (4) -> "FILE";

case (5) -> "COMMON";

case (6) -> "TLS";

case (10) -> "LOOS";

case (12) -> "HIOS";

case (13) -> "LOPROC";

case (15) -> "HIPROC";

default -> "ERROR";

};

}

public String getVis() {

return switch (vis) {

case (0) -> "DEFAULT";

case (1) -> "INTERNAL";

case (2) -> "HIDDEN";

case (3) -> "PROTECTED";

case (4) -> "EXPORTED";

case (5) -> "SINGLETON";

case (6) -> "ELIMINATE";

default -> "ERROR";

};

}

}