Описание системы кодирования команд RISC-V

ISA - архитектура набора команд, определяющая системную реализацию работы процессора.

RISC-V — реализация **ISA**, расширяемая 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 и несколько стандартных опциональных расширений.

Базовые наборы

- RV32I — Базовый набор с целочисленными операциями, 32-битный
- RV64I — Базовый набор с целочисленными операциями, 64-битный
- RV32E — Базовый набор с целочисленными операциями для встраиваемых систем, 32-битный
- RV128I — Базовый набор с целочисленными операциями, 128-битный

Некоторые дополнительные расширения наборов

- ~M — Целочисленное умножение и деление
- ~A — Атомарные операции (то есть операции, которые не могут быть выполнены частично; они либо выполняются, либо нет)
- ~F — Арифметические операции с плавающей запятой над числами одинарной точности
- ~D — Арифметические операции с плавающей запятой над числами двойной точности
- ~Q — Арифметические операции с плавающей запятой над числами четверной точности
- ~L — Арифметические операции с плавающей запятой над десятичными числами
- ~C — Набор с сокращенными названиями команд
- ~B — Битовые операции
- ~J — Набор с эмуляцией набора команд с поддержкой динамической компиляцией во время запуска
- ~P — SIMD-операции
- ~V — Векторная обработка данных (параллельно для скорости)

Наборы RV32I и RV32M

RV32I — базовый набор для работы с 32-битными числами, включает 39 целочисленных инструкций. Эти инструкции делят на группы: *R, I, S, B, J, U*. У каждой группы свой отдельный идентификатор, записанный в виде нескольких бит в конце каждой команды, а также отдельная структура описания инструкции.

Выполняемая программа имеет доступ к 32 регистрам с разными именами, с которыми может выполнять необходимые задачи.

Инструкции могут (Рис. 1):

- Совершить операцию с двумя регистрами, записать результат в третий;
- Совершить операцию с регистром и константой, записать результат в регистр;
- Загрузить часть регистра из памяти;
- Записать часть регистра в память;
- Перейти к заданной инструкции, если заданное условие выполняется;
- Перейти к заданной инструкции;
- Приостановить выполнение и передать контроль операционной системе или отладчику.

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

Рис.1: Все инструкции базового набора RV32I.

RV32M — стандартное расширение к базовому набору RV32I, добавляет функционал работы с умножением и делением. Даёт операции умножения, остатка, деления нацело (Рис. 2).

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)
mul	MUL	R	0110011	0x0	0x01	rd = (rs1 * rs2)[31:0]
mulh	MUL High	R	0110011	0x1	0x01	rd = (rs1 * rs2)[63:32]
mulsu	MUL High (S) (U)	R	0110011	0x2	0x01	rd = (rs1 * rs2)[63:32]
mulu	MUL High (U)	R	0110011	0x3	0x01	rd = (rs1 * rs2)[63:32]
div	DIV	R	0110011	0x4	0x01	rd = rs1 / rs2
divu	DIV (U)	R	0110011	0x5	0x01	rd = rs1 / rs2
rem	Remainder	R	0110011	0x6	0x01	rd = rs1 % rs2
remu	Remainder (U)	R	0110011	0x7	0x01	rd = rs1 % rs2

Рис.2: Инструкции расширения RV32M.

Представление инструкций

В памяти все инструкции RV32I и RV32M представляются в виде 32 бит. То, как именно они представляются, зависит от их типа **FMT** и заданной этому типу структуре (см. Рис. 3)

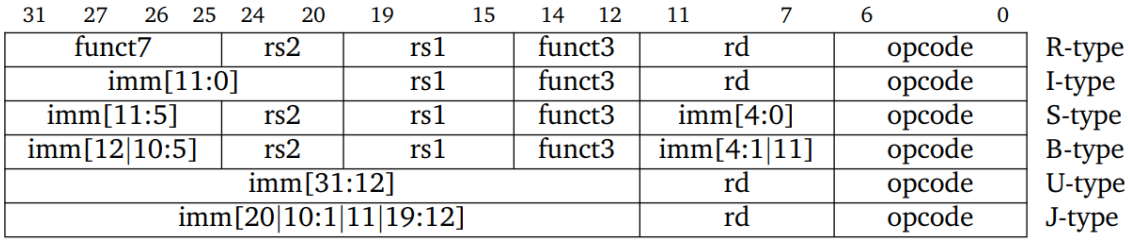

Рис.3: Структура инструкций

В целом, почти все инструкции в **RISC-V** задаются в виде 32 бит, кроме расширения RV32C, который сокращает инструкции до 16 бит.

Сама инструкция может быть определена по идентификатору группы **opcode** и значению **funct3** (в случае R-типа — дополнительно по значению **funct7**, так как количество инструкций в этой группе превышает лимит **funct3**). Всё задается в формате *little-endian*, то есть чтобы можно было прочитать инструкцию в формате слева направо как на табличке, четыре байта нужно повернуть в обратном порядке (**0X**232EF4FC -> **0X**FCF42E23). Поля **rs1** и **rs2** представляют собой регистры (*адреса регистров) над которыми производятся операции, **rd —** регистр для записи результата работы операции над **rs1** и **rs2**. Поле **imm** в зависимости от типа может представлять собой как константу для совершения с ней операции над регистром, так и значение, на которое должен быть совершен переход, например в операциях типа *B* и *J*.

Регистры, над которыми совершают операции имеют свои имена и предназначение (см. Рис. 4). Первые 16 идут в использование набору RV32I, остальные 16 предназначены для работы с нецелыми числами, например для расширений RV32F или RV32F, поэтому здесь рассматриваться не будут.

Поле **imm** имеет такую странную структуру с разными частями в совершенно разных позициях инструкции по той причине, что на уровне аппаратной реализации эти значения извлекает мультиплексор; данная реализация помогает уменьшить количество раз, которое мультиплексор затрачивает на получение значений, уменьшая коллизии битов;

соответственно в некоторых случаях можно однозначно определить некоторые биты, не прочитав их, по другим уже прочитанным битам. Это в итоге повышает общую производительность.

Register	ABI Name	Description	Saver
x0	zero	Zero constant	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Fn args/return values	Caller
x12-x17	a2-a7	Fn args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP args/return values	Caller
f12-17	fa2-7	FP args	Caller
f18-27	fs2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller

Рис.4: имена регистров, их идентификаторы в полях **rs1**, **rs2** и **rd**

ELF файлы

Эльфы, Старшие Дети этого мира, были племенем прекрасным и благородным; владыками их были Эльдар, ныне покинувшие эту землю, – Народ Великого Перехода, Народ Звёзд. (с) Властелин Колец

ELF — формат исполняемых двоичных файлов, используемый во многих современных UNIX-подобных операционных системах, таких как FreeBSD, Linux и Solaris.

Обычно ELF делится на два типа — один для 32-разрядной архитектуры и второй, позже появившийся, для 64-битной. Здесь мы будем рассматривать 32-разрядную архитектуру.

Заголовок файла

В 32-битной реализации состоит из 52 байт, описывает самые важные поля, такие как начало выполняемого кода, начало полей с информацией и количество этой информации:

- *e_ident* — содержит некоторую информацию по файлу и архитектуре. Первые четыре байта обязаны быть 0x7f, 0x45, 0x4c и 0x46, иначе нас обманули и это не ELF файл;
- *e_type* — тип исполняемого файла;
- *e_machine* — тип архитектуры, для реализации RISC-V будет 0xF3;
- *e_version* — версия, на данный момент всегда 1;
- *e_entry* — адрес, откуда начинается выполнение файла;
- *e_phoff* — позиция таблицы заголовков программы (в данной работе эта таблица не нужна, но необходима операционной системе для подготовки к запуску);
- *e_shoff* — позиция таблицы заголовков секций;
- *e_flags* — связанные с файлом флаги для использования процессором;
- *e_ehsize* — размер заголовка файла, для 32-битной реализации равен 52;
- *e_phentsize* — размер одного заголовка программы;
- *e_phnum* — число заголовков программы;
- *e_shentsize* — размер одного заголовка секции;
- *e_shnum* — число заголовков секции;
- *e_shstrndx* — индекс записи в таблице заголовков секций, описывающей таблицу названий секций.

Таблица заголовков секций

Таблица заголовков секций содержит атрибуты секций файла. Секции передают информацию, либо для исполнения, либо для использования другими секциями. Каждый заголовок секции описывает отдельную секцию:

- *sh_name* — указатель на имя секции в таблице названий секций, о ней будет сказано позже;
- *sh_type* — тип секции;
- *sh_flags* — атрибуты секции;
- *sh_addr* — при необходимости предварительной загрузки секции указывает адрес, куда её можно загрузить;
- *sh_offset* — расположение секции относительно начала файла;
- *sh_size* — размер секции в байтах;
- *sh_link* — индекс ассоциированной секции;
- *sh_info* — другая дополнительная информация;
- *sh_addralign* — выравнивание секции;
- *sh_entsize* — размер в байтах каждого элемента в секции.

Секции

Секции описывают различные части файла, в итоге формируя исполняемый файл. Сфокусируемся на *.strtab*, *.symtab* и *.text*:

- *.strtab* — содержит список строк, разделенных нулями. Эти строки являются именами секций и именами символов из секции *.symtab*, представленных в заданном ELF файле. Именно в этой секции другие секции указывают на своё имя полем *sh_name.*
- *.symtab* — в этой секции содержатся все символы, которые компоновщик использует как во время компиляции, так и во время выполнения приложения. В реализации 32-битного ELF файла содержит 16-байтные структуры, называемые символами и имеющие имя, значение, размер и другую информацию. Инструкции RISC-V в итоге используют эти символы для описания блока функции.
- *.text* — сама исполняемая программа; содержит 4-байтные блоки инструкций, в данном случае инструкции заданы в системе кодирования **RISC-V**.

Описание работы написанного кода

Общее описание

Код был написан на Java с использованием JDK 19.

Все важные части: инструкции, секции, символы, заголовки описаны своими классами и при создании достают по заданному адресу информацию о себе и записывают для дальнейшего использования. Также описан общий класс Pair для удобства, так как, к сожалению, язык не поддерживает данную структуру в отличие от C++.

Все описываемые файлы находятся на одном уровне с запускаемым файлом Main.java.

Существует ещё один файл, Constants.java, содержащий необходимые переменные, константы, словари, к которым исполняемый код обращается по мере надобности. Тут же и описана функция extractBytes, которая читает информацию из файла в формате little-endian:

```
public static long extractBytes(long offset, long byteNum) throws Exception {
    if (byteNum <= 0 || byteNum > 4) {
        throw new Exception("Wrong number of bytes to read in extractBytes()");
    }
    long result = 0;
    for(long i = 0; i < byteNum; i++){
        if (offset + byteNum - i - 1 < 0 || offset + byteNum - i - 1 >= TOTAL) {
            throw new Exception("Index out of boundaries when trying to read in
extractBytes()");
        }
        result = result * 256 + (bytes[(int) (offset + byteNum - i - 1)]);
    }
    return result;
}
```

Эта функция используется, например в классе Header.java для читания ELF файла:

```
public Header() throws Exception {
    e_type = Constants.extractBytes(16, 2);
    e_machine = Constants.extractBytes(18, 2);
    e_version = Constants.extractBytes(20, 4);
    e_entry = Constants.extractBytes(24, 4);
    e_phoff = Constants.extractBytes(28, 4);
    e_shoff = Constants.extractBytes(32, 4);
    e_flags = Constants.extractBytes(36, 4);
    e_ehsize = Constants.extractBytes(40, 2);
    e_phentsize = Constants.extractBytes(42, 2);
    e_phnum = Constants.extractBytes(44, 2);
    e_shentsize = Constants.extractBytes(46, 2);
    e_shnum = Constants.extractBytes(48, 2);
    e_shstrndx = Constants.extractBytes(50, 2);
}
```

Ещё есть функция для преобразования беззнакового типа в знаковый, используется для преобразования поля **imm:**

```
public static long immToSigned(long n, long where){
    if ((n & (1L << (where - 1))) != 0) {
        return (n | -(1L << where));
    }
    else {
        return n;
    }
}
```

И функция которая помогает выделить необходимые биты из данных:

```
public static long cutInstruction(long bin, long from, long to) {
    bin = bin & ((1L << (from + 1)) - 1);
    bin = bin & (-(1L << to));
    return (bin >> to);
}
```

Также в Constants.java описаны словари для каждого **FMT** типа, которые по полям **funct3** и **funct7** инструкции RISC-V однозначно определяют операцию. Например, для типа *S* словарь имеет следующий вид:

```
public static final Map<Pair<Integer, Integer>, String> FmtS = Map.ofEntries(
        entry(new Pair<>(0, 0), "sb"),
        entry(new Pair<>(0x1, 0), "sh"),
        entry(new Pair<>(0x2, 0), "sw")
);
```

Парсинг

Берём аргументы командной строки, если мало — ошибка. Проверяем наличие файлов читки и записи, если кого-то не существует — ошибка. Далее читаем поток байтов в массив Constants.bytes, преобразуем из signed byte в положительный long. Даём классу Header прочитать свои 52 байта и выделить поля, проверяем некоторые из них, такие как первые четыре **магических** бита, тип архитектуры, размер заголовка, если что-то не совпадает — ошибка.

Далее, читаем все заголовки секций классами SectionHeader, ищем их настоящие имена в *.strtab*, который лежит по позиции найденного ранее поля e_shstrndx из заголовка файла. Для удобства ложим каждую секцию в словарь по ключу имени из *.strtab* и значению самого класса секции.

Далее, достанем из этого словаря секции *.strtab* и *.symtab*. Теперь можно достать все символы классом Symbol из *.symtab* и соответствующие

им имена из *.strtab*. Также добавим символы в словарь для дальнейшего использования инструкиями.

Теперь можно достать секцию *.text,* где собственно и будет выполняться само дизассемблирование. Выделяя по четыре байта, передаем парсинг классу Instruction, который, используя словари **FMT** типов, определяет операции и читает их поля соответствующе своему типу. Определив тип, он находит имена используемых регистров. Если вдруг это инструкция, подразумевающая переход (типа *B* или *J*), то необходимо ещё проверить наличие метки (символа из *.symtab*) в словаре (про который было сказано выше) по адресу с переходом. Если её нет, то добавим её в словарь по заданному правилу, после чего увеличим счётчик этих локальных меток. Добавим инструкцию в список чтобы в конце вывести их все.

Для вывода сначала проходим по списку символов (не словарю с добавленными дополнительно локальными метками, а именно изначальному списку из *.symtab*), выводим каждый по заданному шаблону. Далее берём список инструкций. Если вдруг адрес инструкции указывает на адрес метки в словаре, то выводим её. Независимо от наличия метки выводим саму инструкцию согласно её типу и заданному формату.

В процессе всех действий в случае любой непонятной ситуации резко бросаем ошибку.

Компиляция и запуск

Компиляцию можно произвести командой ***javac Main.java***

Запуск командой ***java Main rv3 input_file output_file***

Результат работы на заданном файле

```
.text
00010074   <main>:
   10074:       ff010113   addi sp, sp -16
   10078:       00112623     sw ra, 12(sp)
   1007c:       030000ef    jal ra, 0x100ac <mmul>
   10080:       00c12083     lw ra, 12(sp)
   10084:       00000513   addi a0, zero 0
   10088:       01010113   addi sp, sp 16
   1008c:       00008067   jalr zero, 0(ra)
   10090:       00000013   addi zero, zero 0
   10094:       00100137    lui sp, 256
   10098:       fddff0ef    jal ra, 0x10074 <main>
   1009c:       00050593   addi a1, a0 0
   100a0:       00a00893   addi a7, zero 10
   100a4:       0ff0000f        unknown_instruction
   100a8:       00000073  ecall
000100ac   <mmul>:
   100ac:       00011f37    lui t5, 17
   100b0:       124f0513   addi a0, t5 292
   100b4:       65450513   addi a0, a0 1620
   100b8:       124f0f13   addi t5, t5 292
   100bc:       e4018293   addi t0, gp -448
   100c0:       fd018f93   addi t6, gp -48
   100c4:       02800e93   addi t4, zero 40
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)
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://en.wikipedia.org/wiki/RISC-V

https://blog.k3170makan.com/2018/10/introduction-to-elf-format-part-vi.html

https://github.com/jameslzhu/riscv-card/blob/master/riscv-card.pdf

https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html

https://en.wikipedia.org/wiki/Instruction_set_architecture

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

Листинг кода

Main.java

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.LinkedList;
import java.util.List;


public class Main {
    public static void main(String[] args) throws Exception {
        if(args.length < 3){
            throw new Exception("Not enough arguments");
        }
        String input_file = args[1];
        String output_file = args[2];

        File f1 = new File(input_file);
        if(!f1.exists() || f1.isDirectory()) {
            System.out.println("Input file not found");
            return;
        }

        File f2 = new File(output_file);
        if(!f2.exists() || f2.isDirectory()) {
            System.out.println("Output file not found");
            return;
        }

        byte[] bytes_tmp;
        try (DataInputStream reader = new DataInputStream(new
FileInputStream(input_file))) {
            bytes_tmp = reader.readAllBytes();
        } catch(Exception ex) {
            throw new Exception("Error reading file: " + ex.getMessage());
        }

        Constants.TOTAL = bytes_tmp.length;
        Constants.bytes = new long[(int) Constants.TOTAL];
        for(long i = 0; i < Constants.TOTAL; i++) {
            long c = ((bytes_tmp[(int) i] < 0) ? 256 : 0) + bytes_tmp[(int) i];
            Constants.bytes[(int) i] = c;
        }

        long magic = Constants.extractBytes(0, 4);
        if(magic != Constants.ELF_MAGIC) {
            throw new Exception("Not an ELF file");
        }
        // Extract elf header fields
        Header header = new Header();

        if(header.e_machine != 0xF3) {
            throw new Exception("File is not for RISC-V architecture");
        }

        if(header.e_version != 1) {
            throw new Exception("Wrong file version");
        }
```

```
        if(header.e_ehsize != 52) {
            throw new Exception("Wrong header size");
        }

        // Parse section headers and their names
        SectionHeader sectionTable = new SectionHeader(header.e_shoff,
header.e_shstrndx, header.e_shentsize);
        for(long i = 0; i < header.e_shnum; i++){
            SectionHeader sectionHeader = new SectionHeader(header.e_shoff, i,
header.e_shentsize);

            long pos = sectionTable.sh_offset + sectionHeader.sh_name;

            StringBuilder name = new StringBuilder();
            while(Constants.bytes[(int) pos] != 0){
                name.append((char) Constants.bytes[(int) pos]);
                pos++;
            }
            Constants.sections.put(name.toString(), sectionHeader);
        }

        // Parsing symtab
        SectionHeader symtab = Constants.sections.get(".symtab");
        SectionHeader strtab = Constants.sections.get(".strtab");

        for (long i = 0; i < symtab.sh_size / 16; i++) {
            Symbol symbol = new Symbol(symtab.sh_offset + i * 16,
strtab.sh_offset);
            Constants.symbols.add(symbol);
            Constants.tags.put(symbol.st_value, "<" + symbol.name + ">");
        }
        // Parsing text
        SectionHeader text = Constants.sections.get(".text");
        List<Pair<Long, Instruction>> instructions = new LinkedList<>();
        long addr = header.e_entry;
        for(long off = text.sh_offset; off < text.sh_offset + text.sh_size; off +=
4){
            Instruction instruction = new Instruction(off, addr);
            instructions.add(new Pair<>(addr, instruction));
            addr += 4;
        }

        try (Writer writer = new BufferedWriter(new OutputStreamWriter(
                new FileOutputStream(output_file), StandardCharsets.UTF_8))) {
            writer.write(".text\n");
            for(Pair<Long, Instruction> a : instructions) {
                if(Constants.tags.containsKey(a.getFirst())) {
                    writer.write(String.format("%08x   %s:\n", a.getFirst(),
Constants.tags.get(a.getFirst())));
                }
                Instruction inst = a.getSecond();
                switch (inst.type) {
                    case I -> writer.write(String.format("   %05x:\t%08x\t%7s\t%s,
%s %d\n",
                            a.getFirst(), inst.bin, inst.name, inst.rd, inst.rs1,
inst.imm));
                    case IJalr, ILoad -> writer.write(String.format("
%05x:\t%08x\t%7s\t%s, %d(%s)\n",
```

```
                            a.getFirst(), inst.bin, inst.name, inst.rd, inst.imm,
inst.rs1));
                    case IEnv -> writer.write(String.format("
%05x:\t%08x\t%7s\n", a.getFirst(), inst.bin, inst.name));
                    case RM -> writer.write(String.format("
%05x:\t%08x\t%7s\t%s, %s, %s\n",
                            a.getFirst(), inst.bin, inst.name, inst.rd, inst.rs1,
inst.rs2));
                    case S -> writer.write(String.format("   %05x:\t%08x\t%7s\t%s,
%d(%s)\n",
                            a.getFirst(), inst.bin, inst.name, inst.rs2, inst.imm,
inst.rs1));
                    case J -> writer.write(String.format("   %05x:\t%08x\t%7s\t%s,
0x%x %s\n",
                            a.getFirst(), inst.bin, inst.name, inst.rd,
a.getFirst() + inst.imm,
                            Constants.tags.get(a.getFirst() + inst.imm)));
                    case UAui, ULui -> writer.write(String.format("
%05x:\t%08x\t%7s\t%s, %d\n",
                            a.getFirst(), inst.bin, inst.name, inst.rd,
inst.imm));
                    case B -> writer.write(String.format("   %05x:\t%08x\t%7s\t%s,
%s 0x%x %s\n",
                            a.getFirst(), inst.bin, inst.name, inst.rs1, inst.rs2,
                            a.getFirst() + inst.imm,
Constants.tags.get(a.getFirst() + inst.imm)));
                    case unknown_instruction -> writer.write(String.format("
%05x:\t%08x\t\t%7s\n", a.getFirst(), inst.bin, inst.type));
                    default -> throw new Exception("Unknown instruction
encountered while printing");
                }
            }
            writer.write("Symbol Value              Size Type  Bind   Vis    Index
Name\n");
            int symInd = 0;
            for(Symbol symbol : Constants.symbols) {
                String type = switch((int) (symbol.st_info & 15)){
                    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 -> throw new Exception("Unknown symbol type");
                };
                String bind = switch((int) (symbol.st_info >> 4)){
                    case 0  -> "LOCAL";
                    case 1  -> "GLOBAL";
                    case 2  -> "WEAK";
                    case 10 -> "LOOS";
                    case 12 -> "HIOS";
                    case 13 -> "LOPROC";
                    case 15 -> "HIPROC";
                    default -> throw new Exception("Unknown symbol binding");
```

```
                };
                String vis = switch((int) symbol.st_other){
                    case 0 -> "DEFAULT";
                    case 1 -> "INTERNAL";
                    case 2 -> "HIDDEN";
                    case 3 -> "PROTECTED";
                    default -> throw new Exception("Unknown symbol visibility");
                };
                String index = switch((int) symbol.st_shndx){
                    case 0xfff1 -> "ABS";
                    case 0 -> "UNDEF";
                    case 0xffff -> "XINDEX";
                    default -> Integer.toString((int) symbol.st_shndx);
                };
                writer.write(String.format("[%4d] 0x%-15X %5d %-8s %-8s %-8s %6s
%s\n", symInd, symbol.st_value, symbol.st_size, type, bind, vis, index,
symbol.name));
                symInd++;
            }
        } catch(Exception ex) {
            throw new Exception("Error writing to file: " + ex.getMessage());
        }
    }
}
```

Instruction.java

```
import java.util.Objects;

public class Instruction{
    public String name;
    public String rd;
    public String rs1;
    public String rs2;
    public long imm;
    public Constants.FMT type;
    public long bin;
    public Instruction(long offset, long address) throws Exception {
        bin = Constants.extractBytes(offset, 4);
        long opcode = Constants.cutInstruction(bin, 6, 0);
        name = "unknown instruction";
        type = Constants.FMT.unknown_instruction;

        if(opcode == 0b0110011){
            type = Constants.FMT.RM;
            long rdId = Constants.cutInstruction(bin, 11, 7);
            long funct3 = Constants.cutInstruction(bin, 14, 12);
            long funct7 = Constants.cutInstruction(bin, 31, 25);
            long rs1Id = Constants.cutInstruction(bin, 19, 15);
            long rs2Id = Constants.cutInstruction(bin, 24, 20);
            name = Constants.FmtRM.get(new Pair<>((int)funct3, (int)funct7));
            imm = 0;
            rd = Constants.registerNames.get((int)rdId);
            rs1 = Constants.registerNames.get((int)rs1Id);
            rs2 = Constants.registerNames.get((int)rs2Id);
        }
        if(opcode == 0b0010011) {
            type = Constants.FMT.I;
```

```
            long rdId = Constants.cutInstruction(bin, 11, 7);
            long funct3 = Constants.cutInstruction(bin, 14, 12);
            long funct7 = 0;
            long rs1Id = Constants.cutInstruction(bin, 19, 15);
            imm = Constants.cutInstruction(bin, 31, 20);
            imm = Constants.immToSigned(imm, 12);


            if(funct3 == 0x5) {
                funct7 = Constants.cutInstruction(bin, 31, 25);
            }
            name = Constants.FmtI.get(new Pair<>((int)funct3, (int)funct7));
            rd = Constants.registerNames.get((int)rdId);
            rs1 = Constants.registerNames.get((int)rs1Id);
            rs2 = "";
        }
        if(opcode == 0b0000011) {
            type = Constants.FMT.ILoad;
            long rdId = Constants.cutInstruction(bin, 11, 7);
            long funct3 = Constants.cutInstruction(bin, 14, 12);
            long rs1Id = Constants.cutInstruction(bin, 19, 15);
            imm = Constants.immToSigned(Constants.cutInstruction(bin, 31, 20),
12);
            name = Constants.FmtILoad.get(new Pair<>((int)funct3, 0));
            rd = Constants.registerNames.get((int)rdId);
            rs1 = Constants.registerNames.get((int)rs1Id);
            rs2 = "";
        }
        if(opcode == 0b1100111) {
            type = Constants.FMT.IJalr;
            long rdId = Constants.cutInstruction(bin, 11, 7);
            long rs1Id = Constants.cutInstruction(bin, 19, 15);
            imm = Constants.immToSigned(Constants.cutInstruction(bin, 31, 20),
12);

            name = "jalr";
            rd = Constants.registerNames.get((int)rdId);
            rs1 = Constants.registerNames.get((int)rs1Id);
            rs2 = "";
        }
        if(opcode == 0b1110011) {
            type = Constants.FMT.IEnv;
            long rdId = Constants.cutInstruction(bin, 11, 7);
            long funct7 = Constants.cutInstruction(bin, 31, 20);
            long rs1Id = Constants.cutInstruction(bin, 19, 15);
            imm = Constants.immToSigned(Constants.cutInstruction(bin, 31, 20),
12);

            name = (funct7 == 0 ? "ecall" : "ebreak");
            rd = Constants.registerNames.get((int)rdId);
            rs1 = Constants.registerNames.get((int)rs1Id);
            rs2 = "";
        }
        if(opcode == 0b0100011) {
            type = Constants.FMT.S;
            imm = 0;
            imm |= (Constants.cutInstruction(bin, 31, 25) >> 5);
            imm |= Constants.cutInstruction(bin, 11, 7);
```

```
        long funct3 = Constants.cutInstruction(bin, 14, 12);
        long rs1Id = Constants.cutInstruction(bin, 19, 15);
        long rs2Id = Constants.cutInstruction(bin, 24, 20);
        name = Constants.FmtS.get(new Pair<>((int)funct3, 0));
        imm = Constants.immToSigned(imm, 12);
        rd = "";
        rs1 = Constants.registerNames.get((int)rs1Id);
        rs2 = Constants.registerNames.get((int)rs2Id);
    }
    if(opcode == 0b1100011) {
        type = Constants.FMT.B;
        long imm12_105 = Constants.cutInstruction(bin, 31, 25);
        long imm41_11 = Constants.cutInstruction(bin, 11, 7);

        long imm12th = Constants.cutInstruction(imm12_105, 6, 6);
        long imm105 = Constants.cutInstruction(imm12_105, 5, 0);
        long imm41 = Constants.cutInstruction(imm41_11, 4, 1);
        long imm11th = Constants.cutInstruction(imm41_11, 0, 0);
        imm = 0;
        imm |= (imm12th << 12);
        imm |= (imm11th << 11);
        imm |= (imm105 << 5);
        imm |= (imm41 << 1);


        long funct3 = Constants.cutInstruction(bin, 14, 12);
        long rs1Id = Constants.cutInstruction(bin, 19, 15);
        long rs2Id = Constants.cutInstruction(bin, 24, 20);

        name = Constants.FmtB.get(new Pair<>((int)funct3, 0));
        imm = Constants.immToSigned(imm, 13);
        if(!Constants.tags.containsKey(address + imm)){
            String nextTag = "<L" + Constants.TAGINDEX + ">";
            Constants.tags.put(address + imm, nextTag);
            Constants.TAGINDEX++;
        }
        rd = "";
        rs1 = Constants.registerNames.get((int)rs1Id);
        rs2 = Constants.registerNames.get((int)rs2Id);
    }
    if(opcode == 0b1101111) {
        type = Constants.FMT.J;
        imm = 0;
        long imm20th = Constants.cutInstruction(bin, 31, 31);
        long imm101 = Constants.cutInstruction(bin, 30, 21);
        long imm11th = Constants.cutInstruction(bin, 20, 20);
        long imm1912 = Constants.cutInstruction(bin, 19, 12);

        imm |= (imm20th << 20);
        imm |= (imm101 << 1);
        imm |= (imm11th << 11);
        imm |= (imm1912 << 12);

        imm = Constants.immToSigned(imm, 21);

        long rdId = Constants.cutInstruction(bin, 11, 7);
        name = "jal";
        if(!Constants.tags.containsKey(address + imm)) {
```

```
                String nextTag = "<L" + Constants.TAGINDEX + ">";
                Constants.tags.put(address + imm, nextTag);
                Constants.TAGINDEX++;
            }
            imm = Constants.immToSigned(imm, 21);
            rd = Constants.registerNames.get((int)rdId);
            rs1 = "";
            rs2 = "";
        }
        if(opcode == 0b0110111) {
            type = Constants.FMT.ULui;
            name = "lui";
            imm = Constants.immToSigned(Constants.cutInstruction(bin, 31, 12),
21);
            long rdId = Constants.cutInstruction(bin, 11, 7);
            rd = Constants.registerNames.get((int)rdId);
            rs1 = "";
            rs2 = "";
        }
        if(opcode == 0b0010111) {
            type = Constants.FMT.UAui;
            name = "auipc";
            imm = Constants.immToSigned(Constants.cutInstruction(bin, 31, 12),
21);
            long rdId = Constants.cutInstruction(bin, 11, 7);
            rd = Constants.registerNames.get((int)rdId);
            rs1 = "";
            rs2 = "";
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Instruction that = (Instruction) o;
        return imm == that.imm && bin == that.bin && Objects.equals(name,
that.name) && Objects.equals(rd, that.rd) && Objects.equals(rs1, that.rs1) &&
Objects.equals(rs2, that.rs2) && type == that.type;
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, rd, rs1, rs2, imm, type, bin);
    }
}
```

Symbol.java

```
import java.util.Objects;

public class Symbol{
    public long st_name;
    public long st_value;
    public long st_size;
    public long st_info;
    public long st_other;
    public long st_shndx;
```

```
    public String name;
    public Symbol(long offset, long strtabOffset) throws Exception {
        st_name = Constants.extractBytes(offset + 0, 4);
        st_value = Constants.extractBytes(offset + 4, 4);
        st_size = Constants.extractBytes(offset + 8, 4);
        st_info = Constants.extractBytes(offset + 12, 1);
        st_other = Constants.extractBytes(offset + 13, 1);
        st_shndx = Constants.extractBytes(offset + 14, 2);
        name = "";
        int k = (int) (strtabOffset + st_name);
        while(Constants.bytes[k] != 0){
            name += (char) Constants.bytes[k];
            k++;
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Symbol symbol = (Symbol) o;
        return st_name == symbol.st_name && st_value == symbol.st_value && st_size
== symbol.st_size && st_info == symbol.st_info && st_other == symbol.st_other &&
st_shndx == symbol.st_shndx && Objects.equals(name, symbol.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(st_name, st_value, st_size, st_info, st_other,
st_shndx, name);
    }
}
```

Header.java

```
import java.util.Objects;

public class Header{
    public long e_type;
    public long e_machine;
    public long e_version;
    public long e_entry;
    public long e_phoff;
    public long e_shoff;
    public long e_flags;
    public long e_ehsize;
    public long e_phentsize;
    public long e_phnum;
    public long e_shentsize;
    public long e_shnum;
    public long e_shstrndx;
    public Header() throws Exception {
        e_type = Constants.extractBytes(16, 2);
        e_machine = Constants.extractBytes(18, 2);
        e_version = Constants.extractBytes(20, 4);
        e_entry = Constants.extractBytes(24, 4);
        e_phoff = Constants.extractBytes(28, 4);
        e_shoff = Constants.extractBytes(32, 4);
        e_flags = Constants.extractBytes(36, 4);
```

```
        e_ehsize = Constants.extractBytes(40, 2);
        e_phentsize = Constants.extractBytes(42, 2);
        e_phnum = Constants.extractBytes(44, 2);
        e_shentsize = Constants.extractBytes(46, 2);
        e_shnum = Constants.extractBytes(48, 2);
        e_shstrndx = Constants.extractBytes(50, 2);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Header header = (Header) o;
        return e_type == header.e_type && e_machine == header.e_machine &&
e_version == header.e_version && e_entry == header.e_entry && e_phoff ==
header.e_phoff && e_shoff == header.e_shoff && e_flags == header.e_flags &&
e_ehsize == header.e_ehsize && e_phentsize == header.e_phentsize && e_phnum ==
header.e_phnum && e_shentsize == header.e_shentsize && e_shnum == header.e_shnum
&& e_shstrndx == header.e_shstrndx;
    }

    @Override
    public int hashCode() {
        return Objects.hash(e_type, e_machine, e_version, e_entry, e_phoff,
e_shoff, e_flags, e_ehsize, e_phentsize, e_phnum, e_shentsize, e_shnum,
e_shstrndx);
    }
};
```

SectionHeader.java

```
import java.util.Objects;

public class SectionHeader{
    public long sh_name;
    public long sh_type;
    public long sh_flags;
    public long sh_addr;
    public long sh_offset;
    public long sh_size;
    public long sh_link;
    public long sh_info;
    public long sh_addralign;
    public long sh_entsize;

    public SectionHeader(long e_shoff, long index, long e_shentsize) throws
Exception {
        sh_name = Constants.extractBytes(e_shoff + index * e_shentsize + 0, 4);
        sh_type = Constants.extractBytes(e_shoff + index * e_shentsize + 4, 4);
        sh_flags = Constants.extractBytes(e_shoff + index * e_shentsize + 8, 4);
        sh_addr = Constants.extractBytes(e_shoff + index * e_shentsize + 12, 4);
        sh_offset = Constants.extractBytes(e_shoff + index * e_shentsize + 16, 4);
        sh_size = Constants.extractBytes(e_shoff + index * e_shentsize + 20, 4);
        sh_link = Constants.extractBytes(e_shoff + index * e_shentsize + 24, 4);
        sh_info = Constants.extractBytes(e_shoff + index * e_shentsize + 28, 4);
        sh_addralign = Constants.extractBytes(e_shoff + index * e_shentsize + 32,
4);
        sh_entsize = Constants.extractBytes(e_shoff + index * e_shentsize + 36,
```

```
4);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SectionHeader that = (SectionHeader) o;
        return sh_name == that.sh_name && sh_type == that.sh_type && sh_flags ==
that.sh_flags && sh_addr == that.sh_addr && sh_offset == that.sh_offset && sh_size
== that.sh_size && sh_link == that.sh_link && sh_info == that.sh_info &&
sh_addralign == that.sh_addralign && sh_entsize == that.sh_entsize;
    }

    @Override
    public int hashCode() {
        return Objects.hash(sh_name, sh_type, sh_flags, sh_addr, sh_offset,
sh_size, sh_link, sh_info, sh_addralign, sh_entsize);
    }
}
```

Constants.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static java.util.Map.entry;

public final class Constants {

    public static final long ELF_MAGIC = 0x464C457F;
    public static long[] bytes;
    public static long TAGINDEX = 0;
    public static long TOTAL;
    public static Map<String, SectionHeader> sections = new HashMap<>();
    public static Map<Long, String> tags = new HashMap<Long, String>();
    public static List<Symbol> symbols = new ArrayList<>();
    public static final Map<Pair<Integer, Integer>, String> FmtRM = Map.ofEntries(
            entry(new Pair<>(0, 0), "add"),
            entry(new Pair<>(0,0x20), "sub"),
            entry(new Pair<>(0x4,0), "xor"),
            entry(new Pair<>(0x6,0), "or"),
            entry(new Pair<>(0x7,0), "and"),
            entry(new Pair<>(0x1,0), "sli"),
            entry(new Pair<>(0x5,0), "srl"),
            entry(new Pair<>(0x5,0x20), "sra"),
            entry(new Pair<>(0x2,0), "slt"),
            entry(new Pair<>(0x3,0), "sltu"),
            entry(new Pair<>(0x0,0x1), "mul"),
            entry(new Pair<>(0x1,0x1), "mulh"),
            entry(new Pair<>(0x2,0x1), "mulsu"),
            entry(new Pair<>(0x3,0x1), "mulu"),
            entry(new Pair<>(0x4,0x1), "div"),
            entry(new Pair<>(0x5,0x1), "divu"),
            entry(new Pair<>(0x6,0x1), "rem"),
            entry(new Pair<>(0x7,0x1), "remu")
```

```
    );
    public static final Map<Pair<Integer, Integer>, String> FmtI = Map.ofEntries(
            entry(new Pair<>(0, 0), "addi"),
            entry(new Pair<>(0x4, 0), "xori"),
            entry(new Pair<>(0x6, 0), "ori"),
            entry(new Pair<>(0x7, 0), "0"),
            entry(new Pair<>(0x1, 0), "slli"),
            entry(new Pair<>(0x5, 0), "srli"),
            entry(new Pair<>(0x5, 0x20), "srai"),
            entry(new Pair<>(0x2, 0), "slti"),
            entry(new Pair<>(0x3, 0), "sltiu")
    );
    public static final Map<Pair<Integer, Integer>, String> FmtILoad =
Map.ofEntries(
            entry(new Pair<>(0, 0), "lb"),
            entry(new Pair<>(0x1, 0), "lh"),
            entry(new Pair<>(0x2, 0), "lw"),
            entry(new Pair<>(0x4, 0), "lbu"),
            entry(new Pair<>(0x5, 0), "lhu")
    );
    public static final Map<Pair<Integer, Integer>, String> FmtS = Map.ofEntries(
            entry(new Pair<>(0, 0), "sb"),
            entry(new Pair<>(0x1, 0), "sh"),
            entry(new Pair<>(0x2, 0), "sw")
    );
    public static final Map<Pair<Integer, Integer>, String> FmtB = Map.ofEntries(
            entry(new Pair<>(0, 0), "beq"),
            entry(new Pair<>(0x1, 0), "bne"),
            entry(new Pair<>(0x4, 0), "blt"),
            entry(new Pair<>(0x5, 0), "bge"),
            entry(new Pair<>(0x6, 0), "bltu"),
            entry(new Pair<>(0x7, 0), "bgeu")
    );
    public static final Map<Integer, String> registerNames = Map.ofEntries(
            entry(0, "zero"),
            entry(1, "ra"),
            entry(2, "sp"),
            entry(3, "gp"),
            entry(4, "tp"),
            entry(5, "t0"),
            entry(6, "t1"),
            entry(7, "t2"),
            entry(8, "s0 / fp"),
            entry(9, "s1"),
            entry(10, "a0"),
            entry(11, "a1"),
            entry(12, "a2"),
            entry(13, "a3"),
            entry(14, "a4"),
            entry(15, "a5"),
            entry(16, "a6"),
            entry(17, "a7"),
            entry(18, "s2"),
            entry(19, "s3"),
            entry(20, "s4"),
            entry(21, "s5"),
            entry(22, "s6"),
            entry(23, "s7"),
            entry(24, "s8"),
```

```
            entry(25, "s9"),
            entry(26, "s10"),
            entry(27, "s11"),
            entry(28, "t3"),
            entry(29, "t4"),
            entry(30, "t5"),
            entry(31, "t6")
    );
    public enum FMT{
        RM,
        I,
        ILoad,
        S,
        B,
        J,
        IJalr,
        ULui,
        UAui,
        IEnv,
        unknown_instruction
    };

    public static long extractBytes(long offset, long byteNum) throws Exception {
        if (byteNum <= 0 || byteNum > 4) {
            throw new Exception("Wrong number of bytes to read in
extractBytes()");
        }
        long result = 0;
        for(long i = 0; i < byteNum; i++){
            if (offset + byteNum - i - 1 < 0 || offset + byteNum - i - 1 >= TOTAL)
{
                throw new Exception("Index out of boundaries when trying to read
in extractBytes()");
            }
            result = result * 256 + (bytes[(int) (offset + byteNum - i - 1)]);
        }
        return result;
    }
    public static long cutInstruction(long bin, long from, long to) {
        bin = bin & ((1L << (from + 1)) - 1);
        bin = bin & (-(1L << to));
        return (bin >> to);
    }
    public static long immToSigned(long n, long where){
        if ((n & (1L << (where - 1))) != 0) {
            return (n | -(1L << where));
        }
        else {
            return n;
        }
    }
    private Constants(){

    }
}
```

Pair.java

```java
import java.util.Objects;

public class Pair<F, S>{
    private F first;
    private S second;
    public Pair(F first, S second) {
        this.first = first;
        this.second = second;
    }
    public void setFirst(F first) {
        this.first = first;
    }
    public void setSecond(S second) {
        this.second = second;
    }
    public F getFirst() {
        return first;
    }
    public S getSecond() {
        return second;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Pair<?, ?> pair = (Pair<?, ?>) o;
        return first.equals(pair.first) && second.equals(pair.second);
    }

    @Override
    public int hashCode() {
        return Objects.hash(first, second);
    }
}
```