САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

 Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA. Ассемблер, дизассемблер»

Выполнил: Панюхин Никита Константинович

Номер ИСУ: 334964

студ. гр. M3138

Санкт-Петербург

2021

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

**Инструментарий и требования к работе:** работа может быть выполнена на любом из следующих языков: C/C++, Python, Java.

# **Теоретическая часть**

Задание состоит из двух частей: парсера ELF-файла и дизассемблера поднабора команд RISC-V, а именно RV32I, RV32M, RVC.

Формат ELF описывает двоичный файл с данными. Структура файла состоит из нескольких частей, не обязательно стоящих в заранее известном порядке, на каждую из которых ссылается хотя бы одна предыдущая часть. Так, например, в начале файла, помимо его типа и других опознавательных символов записана таблица, которая называется заголовком ELF-файла (ELF header, File header). В основном в ней хранится информация, относящаяся непосредственно к самому файлу в целом, а также ссылки на другие части, следующие за заголовком. Вторым по порядку разбора объектом в ELF-файле являются программные заголовки (Program header table), которые в свою очередь задают заголовки секций (Section header). Секции – области в файле, которые хранят основную информацию, в том числе программный код. Таким образом, чтобы получить нужную нам информацию из ELF-файла необходимо сначала распарсить заголовок файла, затем заголовки программ, после чего перейти к заголовкам секции и только после этого нам будут доступны адреса (offset) данных в каждой из секций файла. Отмечу, что в реализации оказалось не обязательным парсить заголовки программ, так как начало таблицы заголовков секции также закодировано в заголовке самого ELF-файла.

По заданию нам необходимы лишь две секции – .text и .symtab, однако для получения текстовых значений также пригодится секция .strtab. Описание формата кодирования полей в заголовках секции здесь приведено не будет, его легко найти [в интернете](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#Section_header). После получения всех трёх таблиц и преобразования их в удобно читаемые и используемые объекты ЯП, можно вывести таблицу .symtab в требуемом формате и приступить к выполнению второй части задания – декодированию секции .text, содержащую команды RISC-V.

Для данной работы была использована [спецификация RISC-V версии 2.2](https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf), в которой описаны все необходимые инструкции, а именно как они хранятся и кодируются. Например, на странице 104 указаны инструкции набора RV32I. Каждая такая инструкция имеет фиксированную длину (32 бита для RV32 и 16 бит для RVC) и состоит из блоков, каждый из которых подробно описан в спецификации. Например, блоки “” или “” обозначают регистры, а блоки “” и “” – константы. Константы в инструкциях могут быть записаны не одной последовательностью бит или биты могут стоять не в нужном порядке. В таком случае в спецификации указано, как нужно переставить биты местами, чтобы получилось нужное число. Например, означает, что данные кодируют сначала 20 бит числа, зачем с 10 по 1, затем 11 и так далее.

После расшифровки инструкций по условию задания создаются недостающие в .symtab метки и выводится результат в указанном формате.

# **Практическая часть**

Программа-транслятор была написана на языке Python 3.

В основном процесс написания программы повторяет указанные в теоретической части и спецификации действия. Чтобы сократить количество кода программы сделать её более простой, все инструкции были поделены на несколько типов. Не обязательно советующих указанным в спецификации типам. Таким каждая инструкция принадлежит одной из групп, а внутри одной группы инструкции в основном не различаются. Группы и инструкции хранятся в текстовых файлах RV32.txt и RVC.txt для соответствующих поднаборов RISC-V из условия. При инициализации программа читает эти файлы, обрабатывает их и сохраняет в удобном для себя формате структур ЯП. Каждая группа инструкций имеет свои правила вывода и обработки параметров, например знаковых или беззнаковых целых чисел, имена регистров и т.п.

# **Листинг**

**disassembler.py**

# ┌───────────────────────────────────────────┐

# │ Copyright (c) 2021 Nikita Paniukhin │

# │ Licensed under the MIT license │

# └───────────────────────────────────────────┘

#

# ======================================================================================================================

import sys

import os

sys.path.append("src")

from instructions import Instruction

from section\_constants import \*

from symbol\_constants import \*

from asserts import \*

from utils import \*

# ======================================================================================================================

class Program:

def \_\_init\_\_(self, data=None):

# Segment type

self.p\_type = data[0x00:0x04] if data is not None else None

# Offset of the segment in the file image

self.p\_offset = bytes2int(data[0x04:0x08]) if data is not None else None

# Virtual address of the segment in memory

self.p\_vaddr = data[0x08:0x0C] if data is not None else None

# Segment's physical address

self.p\_paddr = data[0x0C:0x10] if data is not None else None

# Size in bytes of the segment in the file image. May be 0

self.p\_filesz = bytes2int(data[0x10:0x14]) if data is not None else None

# Size in bytes of the segment in memory. May be 0

self.p\_memsz = bytes2int(data[0x14:0x18]) if data is not None else None

# Segment-dependent flags (position for 32-bit structure)

self.p\_flags = data[0x18:0x1C] if data is not None else None

# Alignment

self.p\_align = data[0x1C:0x20] if data is not None else None

class Section:

def \_\_init\_\_(self, data=None):

# This code is PEP8 compliant but unreadable

self.sh\_name = data[0x00:0x04] if data is not None else None

self.sh\_type = data[0x04:0x08] if data is not None else None

self.sh\_flags = data[0x08:0x0C] if data is not None else None

self.sh\_addr = data[0x0C:0x10] if data is not None else None

self.sh\_offset = data[0x10:0x14] if data is not None else None

self.sh\_size = data[0x14:0x18] if data is not None else None

self.sh\_link = data[0x18:0x1C] if data is not None else None

self.sh\_info = data[0x1C:0x20] if data is not None else None

self.sh\_addralign = data[0x20:0x24] if data is not None else None

self.sh\_entsize = data[0x24:0x28] if data is not None else None

self.sh\_name = bytes2int(self.sh\_name)

self.sh\_type = bytes2int(self.sh\_type)

self.sh\_addr = bytes2int(self.sh\_addr)

self.sh\_offset = bytes2int(self.sh\_offset)

self.sh\_size = bytes2int(self.sh\_size)

class Elf32\_Sym:

def \_\_init\_\_(self, data=None):

self.st\_name = bytes2int(data[0x00:0x04]) # Elf32\_Word

self.st\_value = bytes2int(data[0x04:0x8]) # Elf32\_Addr

self.st\_size = bytes2int(data[0x8:0x0C]) # Elf32\_Word

self.st\_info = bytes2int(data[0x0C:0x0D]) # unsigned char

self.st\_other = bytes2int(data[0x0D:0x0E]) # unsigned char

self.st\_shndx = bytes2int(data[0x0E:0x10]) # Elf32\_Half

self.st\_bind = self.st\_info >> 4

self.st\_type = self.st\_info & 0xF

self.st\_info = (self.st\_bind << 4) + (self.st\_type & 0xF)

self.st\_visibility = self.st\_other & 0x3

Elf32\_Sym\_SIZE = 0x10

# ======================================================================================================================

def strtab\_extract(data, strtab, offset):

string\_pos = strtab.sh\_offset + offset

string = ""

while data[string\_pos + len(string)] != 0x00:

string += chr(data[string\_pos + len(string)])

return string

def parse(input\_path, output\_path):

input\_path, output\_path = mkpath(input\_path), mkpath(output\_path)

print("Parsing \"{}\"...".format(input\_path))

assert\_cond(os.path.isfile(input\_path), "File not found")

with open(input\_path, 'rb') as fin:

data = fin.read()

# =================================================== ELF HEADER ===================================================

assert\_equal(data[0x00], 0x7F, "ELF file not detected")

assert\_equal(data[0x01], ord('E'), "ELF file not detected")

assert\_equal(data[0x02], ord('L'), "ELF file not detected")

assert\_equal(data[0x03], ord('F'), "ELF file not detected")

# data[04] = {1: 32-bit, 2: 64-bit}

assert\_equal(data[0x04], 1, "Should be 32-bit elf file")

# data[05] = {1: little-endian, 2: big-endian}

assert\_equal(data[0x05], 1, "Should be coded in little-endian")

# data[06] = Version (always 1)

assert\_equal(data[0x06], 1)

# data[07-08] = ABI

skip(data[0x07])

skip(data[0x08])

# data[09-0F] = Unused, should be 0

assert\_all\_equal(data[0x09:0x0F], 0)

# data[10-11] = File type

skip(data[0x10])

skip(data[0x11])

# data[12-13] = Instruction set architecture

skip(data[0x12:0x14])

# data[14-17] = Elf version

e\_version = data[0x14:0x18]

# assert\_equal(e\_version, b'\x01\x00\x00\x00', "Warning: elf version {} != 1".format(e\_version))

# data[18-1B] = Memory address of the entry point

e\_entry = data[0x18:0x1C]

# data[1C-1F] = Program header offset (for 32-bit = 0x34 = 52)

e\_phoff = bytes2int(data[0x1C:0x1F])

assert\_equal(e\_phoff, 52, "Warning: program header not after file header for 32-bit")

# data[20-23] = Section header offset

e\_shoff = bytes2int(data[0x20:0x24])

# data[24-27] = Smth, depends on the target architecture

skip(data[0x24:0x28])

# data[28-29] = Size of this header (for 32-bit = 0x34 = 52)

e\_ehsize = bytes2int(data[0x28:0x29])

assert\_equal(e\_ehsize, 52, "Warning: elf header size normally should be 52 bytes, not {}".format(e\_ehsize))

# data[2A-2B] = Size of a program header

e\_phentsize = bytes2int(data[0x2A:0x2C])

assert\_equal(e\_phentsize, 32, "Warning: program header size normally should be 32 bytes, not {}".format(e\_phentsize))

# data[2C-2D] = Number of entries in the program header

e\_phnum = bytes2int(data[0x2C:0x2E])

# data[2E-2F] = Size of a section header

e\_shentsize = bytes2int(data[0x2E:0x30])

assert\_equal(e\_shentsize, 0x28, "Warning: can only parse sections with size = {value2}, got size = {value1}")

# data[30-31] = Number of entries in the section header

e\_shnum = bytes2int(data[0x30:0x32])

# data[32-33] = Index of the section header that contains the section names

e\_shstrndx = bytes2int(data[0x32:0x34])

# ================================================ PROGRAM HEADER ==================================================

# offset = e\_phoff

# programs = []

# for \_ in range(e\_phnum):

# programs.append(Program(data[offset:offset + e\_phentsize]))

# offset += e\_phentsize

# del offset

# =================================================== SECTIONS =====================================================

unmapped\_sections = []

offset = e\_shoff

for \_ in range(e\_shnum):

unmapped\_sections.append(Section(data[offset:offset + e\_shentsize]))

offset += e\_shentsize

# Finding .strtab by type (SHT\_STRTAB):

strtab = None

for section in unmapped\_sections:

if section.sh\_type == SHT\_STRTAB:

name\_pos = section.sh\_offset + section.sh\_name

if data[name\_pos:name\_pos + len(".shstrtab")] == b".shstrtab":

strtab = section

break

assert\_cond(strtab is not None, "Can not find .strtab")

# Assign a name to every section:

sections = {strtab\_extract(data, strtab, section.sh\_name): section for section in unmapped\_sections}

assert\_cond(".text" in sections, "Can not find .text")

assert\_cond(".strtab" in sections, "Can not find .strtab")

assert\_cond(".symtab" in sections, "Can not find .symtab")

del unmapped\_sections, offset, strtab, name\_pos

# ==================================================== SYMTAB ======================================================

symtab = sections[".symtab"]

symbols = []

for symbol\_offset in range(symtab.sh\_offset, symtab.sh\_offset + symtab.sh\_size, Elf32\_Sym\_SIZE):

symbol = Elf32\_Sym(data[symbol\_offset:symbol\_offset + Elf32\_Sym\_SIZE])

symbol.st\_name = strtab\_extract(data, sections[".strtab"], symbol.st\_name)

symbols.append(symbol)

# ===================================================== TEXT =======================================================

labels = [0, {}]

for symbol in symbols:

if symbol.st\_type == STT\_FUNC and symbol.st\_name:

labels[1][symbol.st\_value] = symbol.st\_name

instructions = []

offset = sections[".text"].sh\_offset

while offset < sections[".text"].sh\_offset + sections[".text"].sh\_size:

instruction\_size = 4 if int2bits(data[offset]).endswith("11") else 2

instruction = Instruction(

sections[".text"].sh\_addr + offset - sections[".text"].sh\_offset,

bytes2bits(data[offset:offset + instruction\_size]).zfill(instruction\_size \* 8),

labels # Reference to `labels`

)

instructions.append(instruction)

offset += instruction\_size

# ==================================================== RESULT ======================================================

with open(output\_path, 'w', encoding="utf-8") as fout:

# print("; формат строк указан по правилам printf (Си)", file=fout)

print(".text", file=fout)

# print("; строки оформляются в следующем формате", file=fout)

# print("; с меткой: \"%08x %10s: %s %s, %s, %s\"", file=fout)

# print("; без метки: метка является пустой строкой", file=fout)

# print("; числа - десятичная запись", file=fout)

# print("; load/store", file=fout)

# print("; \"%08x %10s: %s %s, %s(%s)\"", file=fout)

# print("; для c.addi\*sp\* команд sp регистр прописывается явно", file=fout)

# print("; примеры:", file=fout)

# print("00010078 \_start: addi a0, zero, 0", file=fout)

# print("0001007a lui a1, 65536", file=fout)

# print("00010080 lw a0, -24(s0)", file=fout)

# print("00010088 c.addi4spn a0, sp, 8", file=fout)

for instruction in instructions:

instruction.print(file=fout)

print(file=fout)

# print("; между секциями text и symtab одна пустая строка", file=fout)

print(".symtab", file=fout)

# print("; заголовок таблицы", file=fout)

# print("; \"%s %-15s %7s %-8s %-8s %-8s %6s %s\\n\"", file=fout)

# print("; строки таблицы", file=fout)

# print("; \"[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\\n\"", file=fout)

print(

"%s %-15s %7s %-8s %-8s %-8s %6s %s" %

("Symbol", "Value", "Size", "Type", "Bind", "Vis", "Index", "Name"),

file=fout

)

for symbol\_index, symbol in enumerate(symbols):

print(

"[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s" %

(

symbol\_index, symbol.st\_value, symbol.st\_size, st\_type2string[symbol.st\_type],

st\_bind2string[symbol.st\_bind], st\_visibility2string[symbol.st\_visibility],

shndx2string[symbol.st\_shndx] if symbol.st\_shndx in shndx2string else symbol.st\_shndx,

symbol.st\_name

),

file=fout

)

# ==================================================== CLEANUP =====================================================

# To cleanup, make variables not unused and linter happy

del data, e\_version, e\_entry, e\_phoff, e\_shoff, e\_ehsize,

e\_phentsize, e\_phnum, e\_shentsize, e\_shnum, e\_shstrndx

def main():

print(MSG["welcome"])

if len(sys.argv) > 1:

input\_path = mkpath(sys.argv[1].strip())

output\_path = mkpath(sys.argv[2].strip())

print("Detected command line arguments, running parse(\"{}\", \"{}\")...".format(input\_path, output\_path), end="\n\n")

parse(input\_path, output\_path)

else:

print()

parse(mkpath("elfs", "test1.elf"), "result.txt")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**utils.py**

from json import load as json\_load

import os

def mkpath(\*paths):

return os.path.normpath(os.path.join(\*paths))

def int2byte(a):

return bytes([a])

def int2hex(a):

return hex(a)[2:]

def hex2int(a):

return int(a, 16)

def bytes2int(a):

return int.from\_bytes(a, "little")

def bytes2hex(a):

return int2hex(bytes2int(a))

def bytes2string(a):

return a.decode("utf-8")

def int2bits(a):

return bin(a)[2:]

def bytes2bits(a):

return int2bits(bytes2int(a))

def bits2int(a):

return int(a, 2)

def skip(\*args, \*\*kwargs):

pass

with open("msg.json" if os.path.isfile("msg.json") else mkpath("src", "msg.json"), 'r', encoding="utf-8") as file:

MSG = json\_load(file)

**symbol\_constants.py**

STT\_NOTYPE = 0

STT\_OBJECT = 1

STT\_FUNC = 2

STT\_SECTION = 3

STT\_FILE = 4

STT\_COMMON = 5

STT\_TLS = 6

STT\_LOOS = 10

STT\_HIOS = 12

STT\_LOPROC = 13

STT\_SPARC\_REGISTER = 13

STT\_HIPROC = 15

st\_type2string = {

STT\_NOTYPE: "NOTYPE",

STT\_OBJECT: "OBJECT",

STT\_FUNC: "FUNC",

STT\_SECTION: "SECTION",

STT\_FILE: "FILE",

STT\_COMMON: "COMMON",

STT\_TLS: "TLS",

STT\_LOOS: "LOOS",

STT\_HIOS: "HIOS",

STT\_LOPROC: "LOPROC",

STT\_SPARC\_REGISTER: "SPARC\_REGISTER",

STT\_HIPROC: "HIPROC"

}

# ======================================================================================================================

STB\_LOCAL = 0

STB\_GLOBAL = 1

STB\_WEAK = 2

STB\_LOOS = 10

STB\_HIOS = 12

STB\_LOPROC = 13

STB\_HIPROC = 15

st\_bind2string = {

STB\_LOCAL: "LOCAL",

STB\_GLOBAL: "GLOBAL",

STB\_WEAK: "WEAK",

STB\_LOOS: "LOOS",

STB\_HIOS: "HIOS",

STB\_LOPROC: "LOPROC",

STB\_HIPROC: "HIPROC"

}

# ======================================================================================================================

STV\_DEFAULT = 0

STV\_INTERNAL = 1

STV\_HIDDEN = 2

STV\_PROTECTED = 3

STV\_EXPORTED = 4

STV\_SINGLETON = 5

STV\_ELIMINATE = 6

st\_visibility2string = {

STV\_DEFAULT: "DEFAULT",

STV\_INTERNAL: "INTERNAL",

STV\_HIDDEN: "HIDDEN",

STV\_PROTECTED: "PROTECTED",

STV\_EXPORTED: "EXPORTED",

STV\_SINGLETON: "SINGLETON",

STV\_ELIMINATE: "ELIMINATE"

}

**section\_constants.py**

SHT\_NULL = 0x0

SHT\_PROGBITS = 0x1

SHT\_SYMTAB = 0x2

SHT\_STRTAB = 0x3

SHT\_RELA = 0x4

SHT\_HASH = 0x5

SHT\_DYNAMIC = 0x6

SHT\_NOTE = 0x7

SHT\_NOBITS = 0x8

SHT\_REL = 0x9

SHT\_SHLIB = 0x0A

SHT\_DYNSYM = 0x0B

SHT\_INIT\_ARRAY = 0x0E

SHT\_FINI\_ARRAY = 0x0F

SHT\_PREINIT\_ARRAY = 0x10

SHT\_GROUP = 0x11

SHT\_SYMTAB\_SHNDX = 0x12

SHT\_NUM = 0x13

# SHT\_LOOS = 0x60000000

# ======================================================================================================================

SHN\_UNDEF = 0

SHN\_LORESERVE = 0xff00

SHN\_LOPROC = 0xff00

SHN\_BEFORE = 0xff00

SHN\_AFTER = 0xff01

SHN\_AMD64\_LCOMMON = 0xff02

SHN\_HIPROC = 0xff1f

SHN\_LOOS = 0xff20

SHN\_LOSUNW = 0xff3f

SHN\_SUNW\_IGNORE = 0xff3f

SHN\_HISUNW = 0xff3f

SHN\_HIOS = 0xff3f

SHN\_ABS = 0xfff1

SHN\_COMMON = 0xfff2

SHN\_XINDEX = 0xffff

SHN\_HIRESERVE = 0xffff

shndx2string = {

SHN\_UNDEF: "UNDEF",

SHN\_LORESERVE: "LORESERVE",

SHN\_LOPROC: "LOPROC",

SHN\_BEFORE: "BEFORE",

SHN\_AFTER: "AFTER",

SHN\_AMD64\_LCOMMON: "AMD64\_LCOMMON",

SHN\_HIPROC: "HIPROC",

SHN\_LOOS: "LOOS",

SHN\_LOSUNW: "LOSUNW",

SHN\_SUNW\_IGNORE: "SUNW\_IGNORE",

SHN\_HISUNW: "HISUNW",

SHN\_HIOS: "HIOS",

SHN\_ABS: "ABS",

SHN\_COMMON: "COMMON",

SHN\_XINDEX: "XINDEX",

SHN\_HIRESERVE: "HIRESERVE"

}

**registors.py**

REGS\_BIT4\_HUMAN = {

0: "zero",

1: "ra",

2: "sp",

3: "gp",

4: "tp",

5: "t0",

6: "t1",

7: "t2",

8: "s0",

9: "s1",

10: "a0",

11: "a1",

12: "a2",

13: "a3",

14: "a4",

15: "a5",

16: "a6",

17: "a7",

18: "s2",

19: "s3",

20: "s4",

21: "s5",

22: "s6",

23: "s7",

24: "s8",

25: "s9",

26: "s10",

27: "s11",

28: "t3",

29: "t4",

30: "t5",

31: "t6"

}

REGS\_BIT2\_HUMAN = {

0: "s0",

1: "s1",

2: "a0",

3: "a1",

4: "a2",

5: "a3",

6: "a4",

7: "a5"

}

REGS\_BIT4 = list(REGS\_BIT4\_HUMAN.values())

REGS\_BIT2 = list(REGS\_BIT2\_HUMAN.values())

**instructions.py**

from instruction\_formatter import format\_instruction

from traceback import print\_exc

from os.path import isfile

from asserts import \*

from utils import \*

from registors import REGS\_BIT4, REGS\_BIT2

from CSR import csr2string

RV32\_PATH = "RV32.txt" if isfile("RV32.txt") else mkpath("src", "RV32.txt")

RVC\_PATH = "RVC.txt" if isfile("RVC.txt") else mkpath("src", "RVC.txt")

with open(RV32\_PATH, 'r', encoding="utf-8") as file:

RV32 = [line.strip().split() for line in file if not line.startswith('#') and line.strip()]

with open(RVC\_PATH, 'r', encoding="utf-8") as file:

RVC = [line.strip().split() for line in file if not line.startswith('#') and line.strip()]

def remove\_comments(data):

for line\_index, line in enumerate(data):

for i in range(len(line)):

if line[i].startswith('#'):

del line[i:]

break

def gen\_reg\_checks(source):

checks = []

if "!=" in source:

i = source.find("!=") + 2

while i < len(source) and ('0' <= source[i] <= '9' or source[i] in "{, }"):

i += 1

right\_operand = source[source.find('!=') + 2:i]

if right\_operand.startswith('{'):

right\_operand = tuple(map(int, right\_operand.lstrip('{').rstrip('}').split(',')))

checks.append(lambda x, right\_operand=right\_operand: x not in right\_operand)

else:

right\_operand = int(right\_operand)

checks.append(lambda x, right\_operand=right\_operand: x != int(right\_operand))

return tuple(checks)

def preprocess\_templates(templates):

for template\_instruction in templates:

for i, item in enumerate(template\_instruction):

if any(item.startswith(x) for x in ("imm", "uimm", "nzimm", "nzuimm")):

imm\_type, item = item.split('[', 1)

imm = tuple(

tuple(map(int, x.split(':'))) if ':' in x else int(x)

for x in item.rstrip("]").split('|')

)

imm\_length = 0

for x in imm:

if isinstance(x, int):

imm\_length += 1

else:

imm\_length += abs(x[0] - x[1]) + 1

template\_instruction[i] = (imm, imm\_length, imm\_type)

elif item.startswith("rd") or item.startswith("rs"):

reg\_size = 3 if "′" in item else 5

checks = gen\_reg\_checks(item)

template\_instruction[i] = ("REG", reg\_size, checks)

elif item.startswith("int"):

num, size = map(int, item.lstrip("int").lstrip('(').rstrip(')').split(','))

template\_instruction[i] = (

"INT",

int(size),

tuple([lambda x, num=num: x == int(num)])

)

remove\_comments(RV32)

preprocess\_templates(RV32)

remove\_comments(RVC)

preprocess\_templates(RVC)

class Instruction:

def \_\_init\_\_(self, addr, source, labels):

'''

Params:

addr, source, labels (lables should be passed by-reference)

Attributes:

addr - [int] instruction address

source - [str] source bits

labels

uknown - [bool] if command is unknown

data

name - [str] command name

type - [str] command type

'''

self.addr = addr

self.labels = labels

self.source = source

if source.endswith("11"):

instructions = RV32

self.command\_size = 4

else:

instructions = RVC

self.command\_size = 2

self.unknown = True

last\_match\_template = None

for template\_instruction in instructions:

try:

match = self.parse(source, template\_instruction)

except Exception as e:

print\_exc()

skip(e)

continue

if match is not None:

assert\_cond(

self.unknown,

"One instruction (#{}) ({}) refers to multiple templates: {} and {}".format(

int2hex(self.addr).zfill(8), source, last\_match\_template, template\_instruction

)

)

self.unknown = False

last\_match\_template = template\_instruction

self.data, self.name, self.type = match

if self.unknown:

print("Instruction(#{:08x}) does not match any template: {}".format(self.addr, source))

def parse(self, source, template):

type = template[-1]

name = template[-2]

data = []

imm\_parts = []

imm\_size = 0

imm\_type = None

cur\_pos = 0

for item in template[:-2]:

# REG

if isinstance(item, tuple) and item[0] == "REG":

\_, reg\_size, checks = item

reg = bits2int(source[cur\_pos:cur\_pos + reg\_size])

if not all(check(reg) for check in checks):

return None

data.append(REGS\_BIT2[reg] if reg\_size == 3 else REGS\_BIT4[reg])

cur\_pos += reg\_size

# REG

elif isinstance(item, tuple) and item[0] == "INT":

\_, int\_size, checks = item

num = bits2int(source[cur\_pos:cur\_pos + int\_size])

if not all(check(num) for check in checks):

return None

data.append(num)

cur\_pos += int\_size

# Imm

elif isinstance(item, tuple):

imm, cur\_imm\_length, imm\_type = item

for x in imm:

if isinstance(x, int):

imm\_size = max(imm\_size, x)

else:

imm\_size = max(imm\_size, \*map(int, x))

imm\_parts.append((imm, source[cur\_pos:cur\_pos + cur\_imm\_length]))

data.append("imm")

cur\_pos += cur\_imm\_length

# Const

elif all('0' <= i <= '9' for i in item):

if item != source[cur\_pos:cur\_pos + len(item)]:

return None

data.append(item)

cur\_pos += len(item)

# Unsigned imm

elif item == "zimm":

data.append(bits2int(source[cur\_pos:cur\_pos + 5]))

cur\_pos += 5

# Const

elif item == "shamt":

data.append(bits2int(source[cur\_pos:cur\_pos + 5]))

cur\_pos += 5

# CSR const

elif item == "csr":

if bits2int(source[cur\_pos:cur\_pos + 12]) not in csr2string:

return None

data.append(csr2string[bits2int(source[cur\_pos:cur\_pos + 12])])

cur\_pos += 12

if imm\_parts:

imm\_size += 1

imm = [0] \* imm\_size

for imm\_template, imm\_source in imm\_parts:

cur\_pos = 0

for x in imm\_template:

if isinstance(x, int):

imm[-x - 1] = imm\_source[cur\_pos]

cur\_pos += 1

else:

for i in range(x[0], x[1] - 1, -1):

imm[-i - 1] = imm\_source[cur\_pos]

cur\_pos += 1

imm = "".join(map(str, imm))

if 'u' not in imm\_type and imm[0] == '1': # Negative integers

imm = int(imm, 2) - (1 << imm\_size)

else:

imm = int(imm, 2)

if type in ("J", "CJ", "B", "CB"):

imm += self.addr

for i, item in enumerate(data):

if item == "imm":

data[i] = imm

if type in ("J", "B"):

if data[0] not in self.labels[1]:

self.labels[1][data[0]] = "LOC\_%05x" % (self.labels[0])

data[0] = "LOC\_%05x" % (self.labels[0])

self.labels[0] += 1

elif type in ("CB", "CJ"):

if data[1] not in self.labels[1]:

self.labels[1][data[1]] = "LOC\_%05x" % (self.labels[0])

data[1] = "LOC\_%05x" % (self.labels[0])

self.labels[0] += 1

return data, name, type

def \_\_str\_\_(self):

return format\_instruction(self)

def print(self, \*args, \*\*kwargs):

return print(self, \*args, \*\*kwargs)

**instruction\_formatter.py**

unmatched\_label\_count = 0

def format\_instruction(instruction):

label = find\_label(instruction, instruction.addr)

if instruction.unknown:

return "%08x %11s %s" % (

instruction.addr,

label + ':' if label else '',

"unknown\_command"

)

if instruction.type == "I":

return "%08x %11s %s %s, %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[3],

instruction.data[1],

instruction.data[0]

)

if instruction.type == "J":

target\_label = find\_label(instruction, instruction.data[0])

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[1],

instruction.data[0] if target\_label is None else target\_label

)

if instruction.type in ("JR", "I-load/store"):

return "%08x %11s %s %s, %s(%s)" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[3],

instruction.data[0],

instruction.data[1]

)

if instruction.type == "U":

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[1],

instruction.data[0]

)

if instruction.type == "S-load/store":

return "%08x %11s %s %s, %s(%s)" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[1],

instruction.data[0],

instruction.data[2]

)

if instruction.type == "R":

return "%08x %11s %s %s, %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[4],

instruction.data[2],

instruction.data[1]

)

if instruction.type == "B":

target\_label = find\_label(instruction, instruction.data[0])

return "%08x %11s %s %s, %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[1],

instruction.data[0] if target\_label is None else target\_label

)

if instruction.type in "C":

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[1]

)

if instruction.type in "CB":

target\_label = find\_label(instruction, instruction.data[1])

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[1] if target\_label is None else target\_label

)

if instruction.type == "CSP":

return "%08x %11s %s %s, %s(sp)" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[1]

)

if instruction.type == "CSP2":

return "%08x %11s %s sp, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[1]

)

if instruction.type in ("CSNG", "System"):

return "%08x %11s %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower()

)

if instruction.type == "CJR":

return "%08x %11s %s %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2]

)

if instruction.type == "CI2R":

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[3]

)

if instruction.type == "CS":

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[3],

instruction.data[5]

)

if instruction.type == "CJ":

target\_label = find\_label(instruction, instruction.data[1])

return "%08x %11s %s %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[1] if target\_label is None else target\_label

)

if instruction.type == "CIW":

return "%08x %11s %s %s, sp, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[2],

instruction.data[1]

)

if instruction.type == "CI2":

return "%08x %11s %s %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[3],

instruction.data[1]

)

if instruction.type == "CL":

return "%08x %11s %s %s, %s(%s)" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[4],

instruction.data[1],

instruction.data[2]

)

if instruction.type == "CSR":

return "%08x %11s %s %s, %s, %s" % (

instruction.addr,

label + ':' if label else '',

instruction.name.lower(),

instruction.data[3],

instruction.data[0],

instruction.data[1]

)

return "%08x" % (instruction.addr)

def find\_label(instruction, address):

if address in instruction.labels[1]:

return instruction.labels[1][address]

return None

**CSR.py**

csr2string = {

0x001: "fflags",

0x002: "frm",

0x003: "fcsr",

0xC00: "cycle",

0xC01: "time",

0xC02: "instret",

0xC80: "cycleh",

0xC81: "timeh",

0xC82: "instreth"

}

**asserts.py**

from types import GeneratorType

from utils import MSG

def assert\_cond(value, error\_message=MSG["assert\_cond"]):

if not value:

exit(error\_message.format(value=value))

def assert\_equal(value1, value2, error\_message=MSG["assert\_equal"]):

if isinstance(value2, tuple) or isinstance(value2, list) or isinstance(value2, GeneratorType):

assert\_cond(

all(value1 != check\_value2 for check\_value2 in value2),

error\_message.format(value1=value1, value2=value2)

)

else:

assert\_cond(value1 == value2, error\_message.format(value1=value1, value2=value2))

return value1

def assert\_not\_equal(value1, value2, error\_message=MSG["assert\_equal"]):

if isinstance(value2, tuple) or isinstance(value2, list) or isinstance(value2, GeneratorType):

assert\_cond(

all(value1 == check\_value2 for check\_value2 in value2),

error\_message.format(value1=value1, value2=value2)

)

else:

assert\_cond(value1 != value2, error\_message.format(value1=value1, value2=value2))

return value1

def assert\_all\_equal(value1, value2, error\_message=MSG["assert\_equal"]):

for check\_value1 in value1:

assert\_equal(check\_value1, value2, error\_message)

return value1

**MSG.json**

{

"assert\_equal": "Assertion error: {value1} != {value2}",

"assert\_not\_equal": "Assertion error: {value1} == {value2}",

"assert\_cond": "Assertion error",

"welcome": "Welcome to ELF disassembler. Feel free to disable any warning if needed - just comment them."

}

**RV32.txt**

# RV32I Base Instruction Set TYPE

imm[31:12] rd 0110111 LUI U

imm[31:12] rd 0010111 AUIPC U

imm[20|10:1|11|19:12] rd 1101111 JAL J

imm[11:0] rs1 000 rd 1100111 JALR JR

imm[12|10:5] rs2 rs1 000 imm[4:1|11] 1100011 BEQ B

imm[12|10:5] rs2 rs1 001 imm[4:1|11] 1100011 BNE B

imm[12|10:5] rs2 rs1 100 imm[4:1|11] 1100011 BLT B

imm[12|10:5] rs2 rs1 101 imm[4:1|11] 1100011 BGE B

imm[12|10:5] rs2 rs1 110 imm[4:1|11] 1100011 BLTU B

imm[12|10:5] rs2 rs1 111 imm[4:1|11] 1100011 BGEU B

imm[11:0] rs1 000 rd 0000011 LB I-load/store

imm[11:0] rs1 001 rd 0000011 LH I-load/store

imm[11:0] rs1 010 rd 0000011 LW I-load/store

imm[11:0] rs1 100 rd 0000011 LBU I-load/store

imm[11:0] rs1 101 rd 0000011 LHU I-load/store

imm[11:5] rs2 rs1 000 imm[4:0] 0100011 SB S-load/store

imm[11:5] rs2 rs1 001 imm[4:0] 0100011 SH S-load/store

imm[11:5] rs2 rs1 010 imm[4:0] 0100011 SW S-load/store

imm[11:0] rs1 000 rd 0010011 ADDI I

imm[11:0] rs1 010 rd 0010011 SLTI I

imm[11:0] rs1 011 rd 0010011 SLTIU I

imm[11:0] rs1 100 rd 0010011 XORI I

imm[11:0] rs1 110 rd 0010011 ORI I

imm[11:0] rs1 111 rd 0010011 ANDI I

0000000 shamt rs1 001 rd 0010011 SLLI R

0000000 shamt rs1 101 rd 0010011 SRLI R

0100000 shamt rs1 101 rd 0010011 SRAI R

0000000 rs2 rs1 000 rd 0110011 ADD R

0100000 rs2 rs1 000 rd 0110011 SUB R

0000000 rs2 rs1 001 rd 0110011 SLL R

0000000 rs2 rs1 010 rd 0110011 SLT R

0000000 rs2 rs1 011 rd 0110011 SLTU R

0000000 rs2 rs1 100 rd 0110011 XOR R

0000000 rs2 rs1 101 rd 0110011 SRL R

0100000 rs2 rs1 101 rd 0110011 SRA R

0000000 rs2 rs1 110 rd 0110011 OR R

0000000 rs2 rs1 111 rd 0110011 AND R

0000 pred succ 00000 000 00000 0001111 FENCE FENCE

0000 0000 0000 00000 001 00000 0001111 FENCE.I FENCE

000000000000 00000 000 00000 1110011 ECALL System

000000000001 00000 000 00000 1110011 EBREAK System

csr rs1 001 rd 1110011 CSRRW CSR

csr rs1 010 rd 1110011 CSRRS CSR

csr rs1 011 rd 1110011 CSRRC CSR

csr zimm 101 rd 1110011 CSRRWI CSR

csr zimm 110 rd 1110011 CSRRSI CSR

csr zimm 111 rd 1110011 CSRRCI CSR

# RV32M Standard Extension TYPE

0000001 rs2 rs1 000 rd 0110011 MUL R

0000001 rs2 rs1 001 rd 0110011 MULH R

0000001 rs2 rs1 010 rd 0110011 MULHSU R

0000001 rs2 rs1 011 rd 0110011 MULHU R

0000001 rs2 rs1 100 rd 0110011 DIV R

0000001 rs2 rs1 101 rd 0110011 DIVU R

0000001 rs2 rs1 110 rd 0110011 REM R

0000001 rs2 rs1 111 rd 0110011 REMU R

**RVC.txt**

# Table 12.4: Instruction listing for RVC, Quadrant 0. TYPE

# 000 0 0 00 Illegal instruction

000 nzuimm[5:4|9:6|2|3] rd′ 00 C.ADDI4SPN CIW

# 001 uimm[5:3] rs1′ uimm[7:6] rd′ 00 C.FLD

# 001 uimm[5:4|8] rs1′ uimm[7:6] rd′ 00 C.LQ

010 uimm[5:3] rs1′ uimm[2|6] rd′ 00 C.LW CL

# 011 uimm[5:3] rs1′ uimm[2|6] rd′ 00 C.FLW

# 011 uimm[5:3] rs1′ uimm[7:6] rd′ 00 C.LD

# 100 — 00 Reserved

# 101 uimm[5:3] rs1′ uimm[7:6] rs2′ 00 C.FSD

# 101 uimm[5:4|8] rs1′ uimm[7:6] rs2′ 00 C.SQ

110 uimm[5:3] rs1′ uimm[2|6] rs2′ 00 C.SW CL

# 111 uimm[5:3] rs1′ uimm[2|6] rs2′ 00 C.FSW

# 111 uimm[5:3] rs1′ uimm[7:6] rs2′ 00 C.SD

# Table 12.5: Instruction listing for RVC, Quadrant 1. TYPE

000 0 int(0,5) int(0,5) 01 C.NOP NOP

000 nzimm[5] rs1/rd!=0 nzimm[4:0] 01 C.ADDI C

001 imm[11|4|9:8|10|6|7|3:1|5] 01 C.JAL CJ

# 001 imm[5] rs1/rd!=0 imm[4:0] 01 C.ADDIW

010 imm[5] rd!=0 imm[4:0] 01 C.LI C

011 nzimm[9] int(2,5) nzimm[4|6|8:7|5] 01 C.ADDI16SP CSP2

011 nzimm[17] rd!={0,2} nzimm[16:12] 01 C.LUI C

100 nzuimm[5] 00 rs1′/rd′ nzuimm[4:0] 01 C.SRLI CI2

# 100 0 00 rs1′/rd′ 0 01 C.SRLI64

100 nzuimm[5] 01 rs1′/rd′ nzuimm[4:0] 01 C.SRAI CI2

# 100 0 01 rs1′/rd′ 0 01 C.SRAI64

100 imm[5] 10 rs1′/rd′ imm[4:0] 01 C.ANDI CI2

100 0 11 rs1′/rd′ 00 rs2′ 01 C.SUB CS

100 0 11 rs1′/rd′ 01 rs2′ 01 C.XOR CS

100 0 11 rs1′/rd′ 10 rs2′ 01 C.OR CS

100 0 11 rs1′/rd′ 11 rs2′ 01 C.AND CS

# 100 1 11 rs1′/rd′ 00 rs2′ 01 C.SUBW

# 100 1 11 rs1′/rd′ 01 rs2′ 01 C.ADDW

# 100 1 11 — 10 — 01 Reserved

# 100 1 11 — 11 — 01 Reserved

101 imm[11|4|9:8|10|6|7|3:1|5] 01 C.J CJ

110 imm[8|4:3] rs1′ imm[7:6|2:1|5] 01 C.BEQZ CB

111 imm[8|4:3] rs1′ imm[7:6|2:1|5] 01 C.BNEZ CB

# Table 12.6: Instruction listing for RVC, Quadrant 2. TYPE

000 nzuimm[5] rs1/rd!=0 nzuimm[4:0] 10 C.SLLI C

# 000 0 rs1/rd!=0 0 10 C.SLLI64

# 001 uimm[5] rd uimm[4:3|8:6] 10 C.FLDSP

# 001 uimm[5] rd!=0 uimm[4|9:6] 10 C.LQSP

010 uimm[5] rd!=0 uimm[4:2|7:6] 10 C.LWSP CSP

# 011 uimm[5] rd uimm[4:2|7:6] 10 C.FLWSP

# 011 uimm[5] rd!=0 uimm[4:3|8:6] 10 C.LDSP

100 0 rs1!=0 int(0,5) 10 C.JR CJR

100 0 rd!=0 rs2!=0 10 C.MV CI2R

100 1 int(0,5) int(0,5) 10 C.EBREAK CSNG

100 1 rs1!=0 int(0,5) 10 C.JALR CJR

100 1 rs1/rd!=0 rs2!=0 10 C.ADD CI2R

# 101 uimm[5:3|8:6] rs2 10 C.FSDSP

# 101 uimm[5:4|9:6] rs2 10 C.SQSP

110 uimm[5:2|7:6] rs2 10 C.SWSP CSP

# 111 uimm[5:2|7:6] rs2 10 C.FSWSP

# 111 uimm[5:3|8:6] rs2 10 C.SDSP