САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №4

«ISA. Ассемблер, дизассемблер»

Выполнил(а): Сентемов Лев Александрович

Номер ИСУ: 334863

студ. гр. М3135

Санкт-Петербург

2021

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

Инструментарий и требования к работе: работа была выполнена на Python 3.7.7.

Теоретическая часть

RISC-V – открытая и свободная ISA, состоящая из 53 основных команд (RV32I), с возможностью расширения дополнительными командами: умножение и деление (RV32M), сжатые команды (RVC) и др.

Instr	Название	Функция	Описание	Формат	Opcode	Func3	Func7	Пример использования
add	ADDition	Сложение	rd = rs1 + rs2	R	0110011	0x0	0x00	op rd, rs1, rs2 xor x2, x5, x6 sll x7, x11, x12
sub	SUBtraction	Вычитание	rd = rs1 - rs2			0x0	0x20	
xor	eXclusive OR	Исключающее ИЛИ	rd = rs1 ^ rs2			0x4	0x00	
or	OR	Логическое ИЛИ	rd = rs1 | rs2			0x6	0x00	
and	AND	Логическое И	rd = rs1 & rs2			0x7	0x00	
sll	Shift Left Logical	Логический сдвиг влево	rd = rs1 << rs2			0x1	0x00	
srl	Shift Right Logical	Логический сдвиг вправо	rd = rs1 >> rs2			0x5	0x00	
sra	Shift Right Arithmetic	Арифметический сдвиг вправо	rd = rs1 >>> rs2			0x5	0x20	
slt	Set Less Then	Результат сравнения A < B	rd = (rs1 < rs2) ? 1 : 0			0x2	0x00	
sltu	Set Less Then Unsigned	Беззнаковое сравнение A < B	rd = (rs1 < rs2) ? 1 : 0			0x3	0x00	
addi	ADDition Immediate	Сложение с константой	rd = rs1 + imm	I	0010011	0x0	-	op rd, rs1, imm addi x6, x3, -12 ori x3, x1, 0x8F
xori	eXclusive OR Immediate	Исключающее ИЛИ с константой	rd = rs1 ^ imm			0x4		
ori	OR Immediate	Логическое ИЛИ с константой	rd = rs1 | imm			0x6		
andi	AND Immediate	Логическое И с константой	rd = rs1 & imm			0x7		
slli	Shift Left Logical Immediate	Логический сдвиг влево	rd = rs1 << imm			0x1	0x00	
srli	Shift Right Logical Immediate	Логический сдвиг вправо	rd = rs1 >> imm			0x5	0x00	
srai	Shift Right Arithmetic Immediate	Арифметический сдвиг вправо	rd = rs1 >>> imm			0x5	0x20	
slti	Set Less Then Immediate	Результат сравнения A < B	rd = (rs1 < imm) ? 1 : 0			0x2	-	
sltiu	Set Less Then Immediate Unsigned	Беззнаковое сравнение A < B	rd = (rs1 < imm) ? 1 : 0			0x3		
lb	Load Byte	Загрузить байт из памяти	rd = SE(Mem[rs1 + imm][7:0])	I	0000011	0x0	-	op rd, imm(rs1) lh x1, 8(x5)
lh	Load Half	Загрузить полуслово из памяти	rd = SE(Mem[rs1 + imm][15:0])			0x1		
lw	Load Word	Загрузить слово из памяти	rd = SE(Mem[rs1 + imm][31:0])			0x2		
lbu	Load Byte Unsigned	Загрузить беззнаковый байт из памяти	rd = Mem[rs1 + imm][7:0]			0x4		
lbh	Load Half Unsigned	Загрузить беззнаковое полуслово из памяти	rd = Mem[rs1 + imm][15:0]			0x5		
sb	Store Byte	Сохранить байт в память	Mem[rs1 + imm][7:0] = rs2[7:0]	S	0100011	0x0	-	op rs2, imm(rs1) sw x1, 0xCF(x12)
sh	Store Half	Сохранить полуслово в память	Mem[rs1 + imm][15:0] = rs2[15:0]			0x1		
sw	Store Word	Сохранить слово в память	Mem[rs1 + imm][31:0] = rs2[31:0]			0x2		
beq	Branch if Equal	Перейти, если A == B	if (rs1 == rs2) PC += imm	B	1100011	0x0	-	comp rs1, rs2, imm beq x8, x9, offset bltu x20, x21, 0xFC
bne	Branch if Not Equal	Перейти, если A != B	if (rs1 != rs2) PC += imm			0x1		
blt	Branch if Less Then	Перейти, если A < B	if (rs1 < rs2) PC += imm			0x4		
bge	Branch if Greater or Equal	Перейти, если A >= B	if (rs1 >= rs2) PC += imm			0x5		
bltu	Branch if Less Then Unsigned	Перейти, если A < B беззнаковое	if (rs1 < rs2) PC += imm			0x6		
bgeu	Branch if Greater or Equal Unsigned	Перейти, если A >= B беззнаковое	if (rs1 >= rs2) PC += imm			0x7		
jal	Jamp And Link	Переход с сохранением адреса возврата	rd = PC + 4; PC += imm	J	1101111	-	-	jal x1, offset
jalr	Jamp And Link Register	Переход по регистру с сохранением адреса возврата	rd = PC + 4; PC = rs1	I	1100111	0x0		jalr x1, 0(x5)
lui	Load Upper Immediate	Загрузить константу в сдвинутую на 12	rd = imm << 12	U	0110111	-	-	lui x3, 0xFFFFF
auipc	Add Upper Immediate to PC	Сохранить счетчик команд в сумме с константой << 12	rd = PC + (imm << 12)		0010111			auipc x2, 0x000FF
ecall	Environment CALL	Передача управления операционной системе	Воспринимать как **nop**	I	1110011	-	-	-
ebreak	Environment BREAK	Передача управления отладчику						

Рисунок № 1 – Базовые команды RISC-V

Базовое RISC-V содержит 32 регистра, для кодировки каждого из которых требуется 5 бит. Регистр x0 всегда равен 0. В наборе соглашений ABI каждый регистр от x0 до x31 имеет определенное имя.

Регистр	Имя в ABI	Описание	Тип
32 целочисленных регистра			
x0	Zero	Always zero	
x1	ra	Return address	Вызывающий
x2	sp	Stack pointer	Вызываемый
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Вызывающий
x6–7	t1–2	Temporary	Вызывающий
x8	s0/fp	Saved register / frame pointer	Вызываемый
x9	s1	Saved register	Вызываемый
x10–11	a0–1	Function argument / return value	Вызывающий
x12–17	a2–7	Function argument	Вызывающий
x18–27	s2–11	Saved register	Вызываемый
x28–31	t3–6	Temporary	Вызывающий

Таблица №1 – Регистры RISC-V

Каждая инструкция базового RISC-V кодируется 32 битами. Команды бывают нескольких типов. Кодировка разных типов отличается.

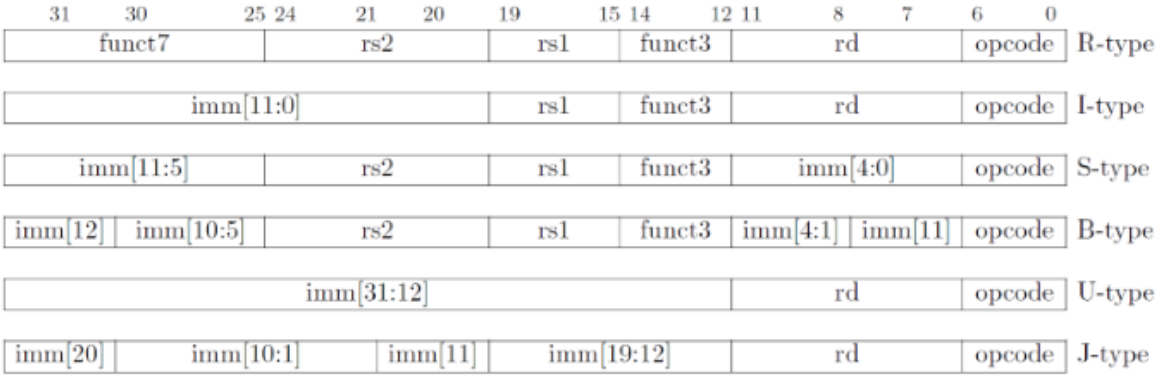

Рисунок № 2 – Кодировка команд

Практическая часть

В аргументах командной строки передаются названия файлов для ввода и вывода.

Программа обрабатывает входной elf файл и выдаёт результат.

Листинг кода

/disassembler.py

```
import sys
from struct import unpack, pack



def read(file, offset, size):
    word = file[offset: offset + size]
    if size == 1:
        return unpack('<B', word)[0]
    elif size == 2:
        return unpack('<H', word)[0]
    elif size == 4:
```

```
        return unpack('<I', word)[0]
    elif size == 8:
        return unpack('<Q', word)
    return unpack('<N', word)[0]



inputName = sys.argv[1]
outputName = sys.argv[2]



inp = open(inputName, 'rb')
out = open(outputName, 'w')

elf = b''
for i in inp.readlines():
    elf += i

inp.close()

e_ident = ['MAG0', 'MAG1', 'MAG2', 'MAG3', 'CLASS', 'DATA', 'VERSION', 'OSABI',
'ABIVERSION', 'EI_PAD', 'EI_PAD1', 'EI_PAD2', 'EI_PAD3', 'EI_PAD4', 'EI_PAD5',
'EI_PAD6']
for i in range(len(e_ident)):
    exec(f'EI_{e_ident[i]} = elf[{i}]')

e_type = unpack('<H', elf[16:18])[0]

e_machine = unpack('<H', elf[18:20])[0]

e_version = unpack('<I', elf[20:24])[0]
```

```
e_entry = unpack('<I', elf[24:28])[0]

e_phoff = unpack('<I', elf[28:32])[0]

e_shoff = unpack('<I', elf[32:36])[0]

e_flags = unpack('<I', elf[36:40])[0]

e_ehsize = unpack('<H', elf[40:42])[0]

e_phentsize = unpack('<H', elf[42:44])[0]

e_phnum = unpack('<H', elf[44:46])[0]

e_shentsize = unpack('<H', elf[46:48])[0]

e_shnum = unpack('<H', elf[48:50])[0]

e_shstrndx = unpack('<H', elf[50:52])[0]

sections = []
sections_table_variables = ['sh_name', 'sh_type', 'sh_flags', 'sh_addr',
                 'sh_offset', 'sh_size', 'sh_link', 'sh_info',
                 'sh_addralign', 'sh_entsize']



for i in range(e_shnum):
    offset = e_shoff + i * e_shentsize
```

```
    sect = {}
    for j in sections_table_variables:
        sect[j] = read(elf, offset, 4)
        offset += 4
    if sect['sh_type'] == 0:
        offset -= 32
    sections.append(sect)



shstrtab_offset = sections[e_shstrndx]['sh_offset']
shstrtab_size = sections[e_shstrndx]['sh_size']
sections_names = elf[shstrtab_offset:shstrtab_offset + shstrtab_size]
for i in range(e_shnum):
    sections[i]['name']  =  sections_names[sections[i]['sh_name']:].split(b'\x00',
1)[0].decode('utf-8')


text_table = None
symtab_table = None
strtab_table = None


for i in sections:
    if i['name'] == '.text':
        text_table = i
    elif i['name'] == '.symtab':
        symtab_table = i
    elif i['name'] == '.strtab':
        strtab_table = i
```

```
text_section      =      elf[text_table['sh_offset']:text_table['sh_offset']      +
text_table['sh_size']]



symtab_section    =    elf[symtab_table['sh_offset']:symtab_table['sh_offset']    +
symtab_table['sh_size']]



strtab_section    =    elf[strtab_table['sh_offset']:strtab_table['sh_offset']    +
strtab_table['sh_size'] - 1]




entries_number = symtab_table['sh_size'] // symtab_table['sh_entsize']

symtab = []



for i in range(entries_number):

    line = symtab_section[symtab_table['sh_entsize'] * i:

                          symtab_table['sh_entsize'] * (i + 1)]



    entry = dict()

    entry['name']    =    strtab_section[read(line,    0,    4):].split(b'\x00',
1)[0].decode('utf-8')

    entry['value'] = read(line, 4, 4)

    entry['size'] = read(line, 8, 4)

    entry['info'] = read(line, 12, 1)

    entry['other'] = read(line, 13, 1)

    entry['shndx'] = read(line, 14, 2)

    symtab.append(entry)




def bi(x):

    f = lambda y: bin(x[y])[2:].rjust(8, '0')

    if len(x) == 2:
```

```
        return f(1) + f(0)
    return f(3) + f(2) + f(1) + f(0)


reg_names = ['zero', 'ra', 'sp', 'gp', 'tp', 't0', 't1', 't2',
             's0', 's1', 'a0', 'a1', 'a2', 'a3', 'a4', 'a5',
             'a6', 'a7', 's2', 's3', 's4', 's5', 's6', 's7',
             's8', 's9', 's10', 's11', 't3', 't4', 't5', 't6']


result_text = []


offset = 0
while offset < len(text_section):
    line = []


    command = bi(text_section[offset:min(offset + 4, len(text_section))])
    if command[30:] != '11':
        continue
    else:
        opcode = command[25:]
        rd = command[20:25]
        funct3 = command[17:20]
        rs1 = command[12:17]
        rs2 = command[7:12]
        funct7 = command[:7]


        if opcode == '0110011':  # R-type
            if funct7 == '0000000':
                line.append(['add',  'sll',  'slt',  'sltu',  'xor',  'srl',  'or',
'and'][int(funct3, 2)])
            elif funct7 == '0100000':
```

```
                line.append(['sub',      'unknown_command',      'unknown_command',
'unknown_command',       'unknown_command',       'sra',       'unknown_command',
'unknown_command'][int(funct3, 2)])

            elif funct7 == '0000001':

                line.append(['mul',  'mulh',  'mulhsu',  'mulhu',  'div',  'divu',
'rem', 'remu'][int(funct3, 2)])

            else:

                line.append('unknown_command')

            line.append(reg_names[int(rd, 2)] + ',')

            line.append(reg_names[int(rs1, 2)] + ',')

            line.append(reg_names[int(rs2, 2)])

        elif opcode == '0010011':  # I-type

            if funct3 == '101':

                if funct7 == '0000000':

                    line.append('srli')

                    line.append(reg_names[int(rd, 2)] + ',')

                    line.append(reg_names[int(rs1, 2)] + ',')

                    imm = int(command[1:12], 2)

                    if command[0]:

                        imm = -imm

                    line.append(str(imm))

                else:

                    line.append('srai')

                    line.append(reg_names[int(rd, 2)] + ',')

                    line.append(reg_names[int(rs1, 2)] + ',')

                    imm = int(command[7:12], 2)

                    line.append(str(imm))

            elif funct3 == '001':

                line.append('slli')

                line.append(reg_names[int(rd, 2)] + ',')

                line.append(reg_names[int(rs1, 2)] + ',')
```

```
                imm = int(command[:12], 2)

                line.append(str(imm))

            else:

                line.append(['addi', 'unknown_command', 'slti', 'sltiu', 'xori',
'unknown_command', 'ori', 'andi'][int(funct3, 2)])

                line.append(reg_names[int(rd, 2)] + ',')

                line.append(reg_names[int(rs1, 2)] + ',')

                imm = int(command[1:12], 2)

                if line[0] in ('addi', 'slti') and command[0] == '1':

                    imm = -imm

                line.append(str(imm))

        elif opcode == '0000011':  # I-type

            line.append(['lb',  'lh',  'lw',  'unknown_command',  'lbu',  'lhu',
'unknown_command', 'unknown_command'][int(funct3, 2)])

            line.append(reg_names[int(rd, 2)] + ',')

            imm = int(command[1:12], 2)

            if command[0] == '1':

                imm = -imm

            line.append(str(imm) + '(' +

                        reg_names[int(rs1, 2)] + ')')

        elif opcode == '0100011':  # S-type

            line.append(['sb', 'sh', 'sw', 'unknown_command', 'unknown_command',
'unknown_command', 'unknown_command', 'unknown_command'][int(funct3, 2)])

            line.append(reg_names[int(rs2, 2)] + ',')

            imm = int(command[1:12] + rd, 2)

            if command[0] == '1':

                imm = -imm

            line.append(str(imm) + '(' +

                        reg_names[int(rs1, 2)] + ')')

        elif opcode == '1100011':  # B-type
```

```
            line.append(['beq', 'bne', 'unknown_command', 'unknown_command', 'blt',
'bge', 'bltu', 'bgeu'][int(funct3, 2)])

            address = text_table['sh_addr'] + offset + 4

            line.append(reg_names[int(rs1, 2)] + ',')

            line.append(reg_names[int(rs2, 2)] + ',')

            line.append(str(offset))

            ##############################

        elif opcode == '1101111':  # jal

            # address = text_table['sh_addr'] + offset + 4

            line.append('unknown_command')

            ##############################

        elif opcode == '1100111':  # jalr

            line.append('jalr')

            imm = int(command[1:12], 2)

            if command[0] == '1':

                imm = -imm

            line.append(reg_names[int(rd, 2)] + ',')

            line.append(reg_names[int(rs1, 2)] + ',')

            line.append(str(imm))

        elif opcode == '0110111': # lui

            line.append('lui')

            line.append(reg_names[int(rd, 2)] + ',')

            imm = int(command[:8], 2)

            line.append(str(imm))

        elif opcode == '0010111': # auipc

            line.append('auicp')

            line.append(reg_names[int(rd, 2)] + ',')

            imm = int(command[:8], 2)

            line.append(str(imm))

        elif opcode == '1110011': # ecall ebreak
```

```
            if command[:12] == '0' * 12:
                line.append('ecall')
            elif command[:12] == '0' * 11 + '1':
                line.append('ebreak')
            else:
                line.append('unknown_command')
        else:
            line.append('unknown_command')
        offset += 4
        result_text.append(line)

out.write('.text\n')
address = text_table['sh_addr']
for i in result_text:
    out.write(str(hex(address))[2:].rjust(8, '0') + ' ' * 10 + ' '.join(i) + '\n')
    address += 4

out.close()
```