Лабораторная работа № 4	Б06	2023
ISA	Прозорова Варвара Владиславовна	

Инструментарий и требования к работе: Python (3.11.5)

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-riscv-stepashka-21/tree/main

Результат работы написанной программы:

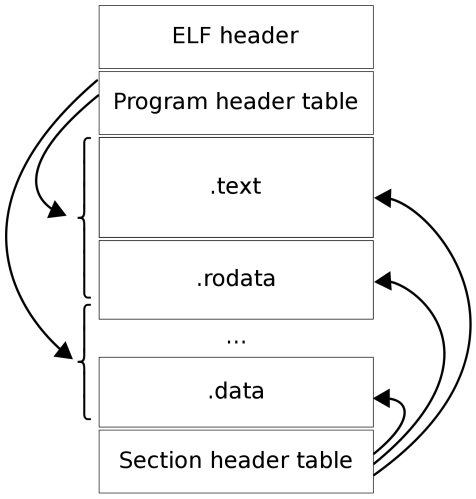
.text

```
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, 0x100
   10098:  fddff0ef    jal     ra, 0x10074 <main>
   1009c:  00050593              addi    a1, a0, 0
   100a0:  00a00893              addi    a7, zero, 10
   100a4:  0ff0000f    fence   1111, 1111
   100a8:  00000073ecall

000100ac <mmul>:
   100ac:  00011f37     lui    t5, 0x11
   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     a0,         a5, 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     t4,         a6, 0x100d8, <L1>

  10114:  050f0f13    addi    t5, t5, 80
  10118:  01478513    addi    a0, a5, 20
  1011c:  fa5f16e3     bne     t0,         t5, 0x100c8, <L2>

  10120:  00008067              jalr      zero, 0(ra)


.symtab
Symbol Value           Size Type     Bind     Vis       Index Name
[   0] 0x0               0 NOTYPE   LOCAL    DEFAULT      0
[   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     0 _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
Process finished with exit code 0
```

Описание работы написанного кода (реализовано: 32i, 32m и symtab)**:**



Структура elf файла: в начале каждого файла есть header или заголовок, содержит общее описание структуры файла и его основные характеристики, таблица заголовков программы (program header) (содержит информацию, необходимую для подготовки программы к выполнению операционной системой), таблица заголовков секций (section header) (хранит информацию о каждой секции), ну и сами секции.

Elh header состоит из `entry`, `phoff`, `shoff`, `phentsize`, `phnum`, `shentsize`, `shnum`, `shstrndx`. Section Header состоит из `sh_name`, `sh_type`, `sh_flags`, `sh_addr`, `sh_offset`, `sh_size`, `sh_link`, `sh_info`, `sh_addralign`, `sh_entsize`.

Словари для “констант”:

- utype, jtype, itype, full_len, stype, btype, rtype, ftype – содержат все rv32m, rv32i.

- regname – названия регистров
- binds, types, indexes, vises – данные для symtab.

Основные функции:

def parse_section_header, def parse_section_header_table – функции для парсинга соответственно section header и section header table.

def parse_elf_header – парсит header, как можно понять из названия (все делалось по таблице из википедии, ссылка указана ниже).

def text_section_content, def symtab_section_content, def strtab_section_content – получение соответсвенно данных для .text, .symtab, .strtab.

def parse_commands – на вход битовая запись команды – определяет opcode, регистры, imm и др. Парсилось все по документу, указанному первым в источниках. Парсинг: есть 32 бита, разбиваем на нужное количество секций для каждого типа, как-то их переворачиваем, преобразовываем, получаем новые 32 бита, которые переводим в инт, получаем имм. Регистры: так же получается из секций 32 бит (все схемы есть в книге).

def disassembler – бьет данные из `text_header` на подстроки длины 4, запускает для каждых из них def parse_commands, получает на выходе список из списков готовых команд.

План действий: парсим хэдер, находим расположение нужных секций, идем в text, далее там парсим команды, получаем итоговый список, выводим. Проблемы могут быть с L метками, поэтому для начала прогоняем весь получившийся список команд, если команда из типа j или b, проверяем есть ли в списке an (имена всех адресов) данный адрес, куда

хотим прыгнуть, если нет, то добавляем и меняем номер следующей L на один. При выводе рассматриваем разные случаи.

Symtab:

Symtab состоит из name (4 bytes) – индекс в секции .strtab, по которому находится название; value (4 bytes) – достается int из байтов, выводится в 16ричной системе; size (4 bytes) – достается int из байтов; info (1 byte) – достается int из байтов, по номеру находится vis, shndx (1 byte) – достается int из байтов, по номеру находится index, other (1 byte) – для вычисления type и bind. Разбиением на байты занимается функция parse_sym_tab, которая получает на вход строку из байтов и выдает на выходе список из частей symtab, перечисленных выше.

План такой: находим symtab, парсим его. Далее для name: берем его индекс, идем в strtab, считаем байты (find_name) (для удобства я заменила все \x00 на #, чтобы реально как один байт), находим нужный элемент и берем срез до первого #. Получаем имя. Остальное получается просто переводом в int, а дальше из нужного словаря вытаскиваем значение.

Промежуточные функции:

всякие to_signed(n) (для перевода числа n в “знаковый” тип), next_int(n) (для “взятия” следующих n байт с переводом в int – парсинг), next_bytes(n) (для “взятия” следующих n байт – парсинг), skip_b(n) (для “пропуска” следующих n байт – парсинг), hex8,

Источники:

RISC-V Assembly Language Programming, John Winans

https://en.wikipedia.org/wiki/RISC-V

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

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html

https://msyksphinz-self.github.io/riscv-isadoc/html/rvm.html