Лабораторная работа №4	22.Б05	2023
ISA	Назаров Матвей Антонович	

Инструментарий и требования к работе: Работа выполняется на Python 3.11.5

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-riscv-lomalovo

Описание работы: начну с описания структуры elf файла. В начале лежит заголовок файла, в нем находится общее описание структуры файла и его характеристики. Из этого заголовка нам нужны только поля:
e_shoff – смещение таблицы заголовков секций относительно начала файла
e_shentsize – размер одного заголовка секции
e_shnum – число заголовков секций
e_shstrndx - индекс заголовка с названиями секций среди всех заголовков
Далее мы обращаемся к Таблице заголовков секций и к секции с именами секций .shstrtab и получаем полную информацию о секциях:
1) Имя
2) sh_name – смещение строки, содержащее название данной секции, относительно начала таблицы названий секций
3) sh_addr – адрес начиная с которого будет загружена секция
4) sh_offset – смещение секции от начала файла
5) sh_size – размер этой секции

Кроме того, нам понадобится секция “.symtab”, в которой все строки имеют длину 16, из нее мы получаем все необходимое нам, а именно st_name, st_value, st_size, st_info, st_other из которых мы позже будем получать нужные нам данные, такие как binding, visibility, index

Теперь приступим к описанию работы кода

Я создаю класс ElfParser в котором и будет производится вся работа

Все “координаты переменных” я брал из статьи википедии про elf (https://ru.wikipedia.org/wiki/Executable_and_Linkable_Format)

```
def __init__(self, elf_path):
    self.out = ""
    with open(elf_path, 'rb') as elf_file:
        self.data = elf_file.read()
    self.sections = dict()
    e_shoff = self.get_from_data(32, 4)
    e_shentsize = self.get_from_data(46, 2)
    e_shnum = self.get_from_data(48, 2)
    e_shstrndx = self.get_from_data(50, 2)
```

Сначала я считаю переменные из заголовка файла, для этого использую функцию get_from_data, которая принимает два аргумента, первый с какого байта читать, а второй сколько читать в формате little endian:

```
def get_from_data(self, start, size):
    return int.from_bytes(self.data[start: start + size], byteorder="little")
```

После этого я парсю секции и записываю их в лист:

```
for i in range(e_shnum):
    sh_name = self.get_from_data(e_shoff + e_shentsize * i, 4)
    sh_addr = self.get_from_data(e_shoff + e_shentsize * i + 12, 4)
    sh_offset = self.get_from_data(e_shoff + e_shentsize * i + 16, 4)
    sh_size = self.get_from_data(e_shoff + e_shentsize * i + 20, 4)
    section_name = ""
    pos_for_name = self.get_from_data(e_shoff + e_shstrndx * e_shentsize
+ 16,
                                      4) + sh_name
    while self.get_from_data(pos_for_name, 1) != 0:
        section_name += chr(self.get_from_data(pos_for_name, 1))
        pos_for_name += 1
    self.sections[section_name] = {
        "sh_name": sh_name,
        "sh_addr": sh_addr,
        "sh_offset": sh_offset,
        "sh_size": sh_size}
```

Далее я парсю “.symtab” и тоже записываю в лист. Про symtab смотрел тут (https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html)

```
symtab = self.sections.get(".symtab")
symtabList = []
symtabSymbols = {}
for i in range(symtab.get("sh_size") // 16):
    addr = symtab.get("sh_offset") + i * 16
    name = ""
    pos_for_name = self.get_from_data(addr, 4) +
self.sections.get(".strtab").get("sh_offset")
    while self.get_from_data(pos_for_name, 1) != 0:
        name += chr(self.get_from_data(pos_for_name, 1))
        pos_for_name += 1
    value = self.get_from_data(addr + 4, 4)
    size = self.get_from_data(addr + 8, 4)
    info = self.get_from_data(addr + 12, 1)
```

```
    other = self.get_from_data(addr + 13, 3)
    item = {
        "symbol": i,
        "name": name,
        "value": value,
        "size": size,
        "info": info,
        "other": other}
    symtabList.append(item)
    symtabSymbols[value] = item
```

Теперь делаю создаю строку для вывода symtab:

```
symtab_format = "[{:4d}] 0x{:<15X} {:5d} {:<8s} {:<8s} {:<8s} {:>6s} {:s}\n"
symtab_out = ""
symtab_out += ".symtab\n\n"
symtab_out += "Symbol Value              Size Type     Bind     Vis      Index Name\n"
for item in symtabList:
    symtab_out += symtab_format.format(
        item.get("symbol"),
        item.get("value"),
        item.get("size"),
        get_type(item.get("info") & 0b1111),
        get_binding(item.get("info") >> 4),
        get_visibility(item.get("other") & 0b11),
        get_index(item.get("other") >> 8),
        item.get("name")
    )
```

Тут я пользую функциями get_type, get_binding, get_visibility, get_index, которые из данных info и other достают нужную нам информацию

Пример одной из таких функций:

```
def get_binding(info: int):
    bindings = {
        0: "LOCAL",
        1: "GLOBAL",
        2: "WEAK",
        10: "LOOS",
        12: "HIOS",
        13: "LOPROC",
        15: "HIPROC",
    }
    return bindings.get(info, 'UNKNOWN')
```

Все данные, я беру из табличек с указанного выше сайта:

Figure 4-17: Symbol Binding

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOOS	10
STB_HIOS	12
STB_LOPROC	13
STB_HIPROC	15

Figure 4-18: Symbol Types

Name	Value
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_HIPROC	15

И т.д.

Теперь самое интересное: Парсинг самих комманд:

Как мы знаем из документации, указанной в ТЗ, в RISC-V есть несколько основных типов комманд:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Для каждого из классов в файле commands.py я создал отдельный dataclass, характеризующий его, пример:

```
class RType:
    funct7: str = ""  # 7 bytes
    rs2: str = ""     # 5 bytes
    rs1: str = ""     # 5 bytes
    funct3: str = ""  # 3 bytes
    rd: str = ""      # 5 bytes
    opcode: str = ""  # 7 bytes
    name: str = ""
```

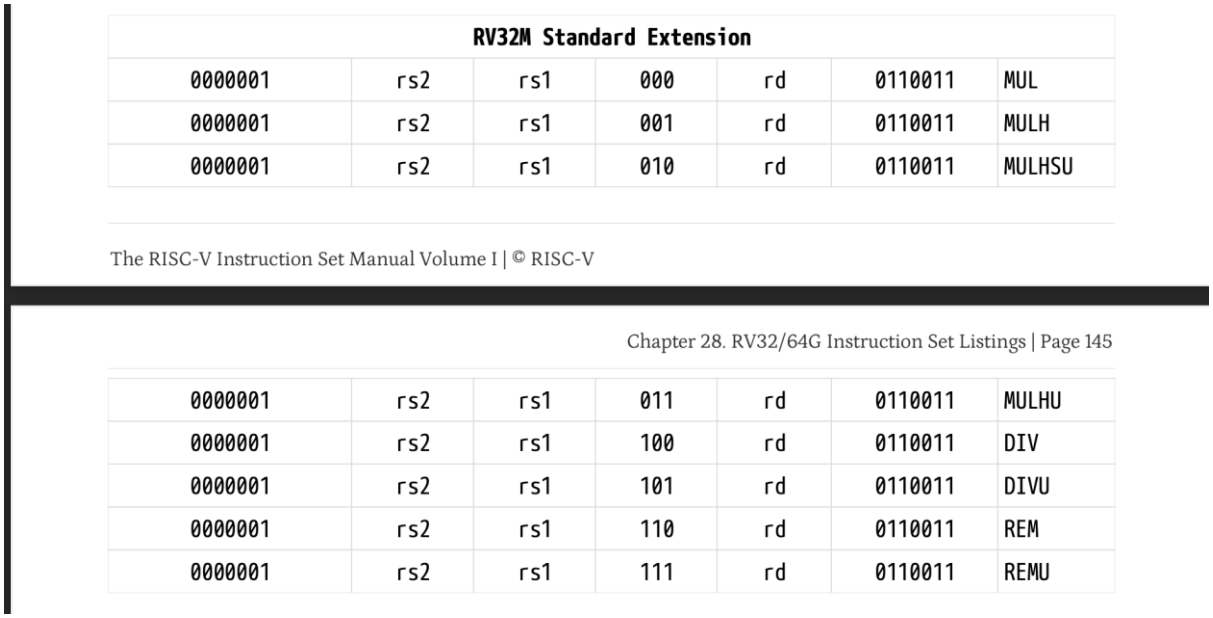
На основе этого я занес в массив все команды из RV32I и RV32M:

```
RV32I = [
    UType(opcode="0110111", name="LUI"),
    UType(opcode="0010111", name="AUIPC"),
    JType(opcode="1101111", name="JAL"),
    IType(funct3="000", opcode="1100111", name="JALR"),
    BType(funct3="000", opcode="1100011", name="BEQ"),
    BType(funct3="001", opcode="1100011", name="BNE"),
    BType(funct3="100", opcode="1100011", name="BLT"),
    BType(funct3="101", opcode="1100011", name="BGE"),
    BType(funct3="110", opcode="1100011", name="BLTU"),
    BType(funct3="111", opcode="1100011", name="BGEU"),
    IType(funct3="000", opcode="0000011", name="LB"),
    IType(funct3="001", opcode="0000011", name="LH"),
    IType(funct3="010", opcode="0000011", name="LW"),
```

```
    IType(funct3="100", opcode="0000011", name="LBU"),
    IType(funct3="101", opcode="0000011", name="LHU"),
    SType(funct3="000", opcode="0100011", name="SB"),
    SType(funct3="001", opcode="0100011", name="SH"),
    SType(funct3="010", opcode="0100011", name="SW"),
    IType(funct3="000", opcode="0010011", name="ADDI"),
    IType(funct3="010", opcode="0010011", name="SLTI"),
    IType(funct3="011", opcode="0010011", name="SLTIU"),
    IType(funct3="100", opcode="0010011", name="XORI"),
    IType(funct3="110", opcode="0010011", name="ORI"),
    IType(funct3="111", opcode="0010011", name="ANDI"),
    IType(imm_11_0="0000000", funct3="001", opcode="0010011", name="SLLI"),
    IType(imm_11_0="0000000", funct3="101", opcode="0010011", name="SRLI"),
    IType(imm_11_0="0100000", funct3="101", opcode="0010011", name="SRAI"),
    RType(funct7="0000000", funct3="000", opcode="0110011", name="ADD"),
    RType(funct7="0100000", funct3="000", opcode="0110011", name="SUB"),
    RType(funct7="0000000", funct3="001", opcode="0110011", name="SLL"),
    RType(funct7="0000000", funct3="010", opcode="0110011", name="SLT"),
    RType(funct7="0000000", funct3="011", opcode="0110011", name="SLTU"),
    RType(funct7="0000000", funct3="100", opcode="0110011", name="XOR"),
    RType(funct7="0000000", funct3="101", opcode="0110011", name="SRL"),
    RType(funct7="0100000", funct3="101", opcode="0110011", name="SRA"),
    RType(funct7="0000000", funct3="110", opcode="0110011", name="OR"),
    RType(funct7="0000000", funct3="111", opcode="0110011", name="AND"),
    FENCEType(funct3="000", opcode="0001111", name="FENCE"),
    FENCEType("1000", "0011", "0011", "00000", "000", "00000", "0001111",
"FENCE.TSO"),
    FENCEType("0000", "0001", "0000", "00000", "000", "00000", "0001111",
"FENCE.TSO"),
    IType("000000000000", "00000", "000", "00000", "1110011", "ECALL"),
    IType("000000000001", "00000", "000", "00000", "1110011", "EBREAK")
]

RV32M = [
    RType(funct7="0000001", funct3="000", opcode="0110011", name="MUL"),
    RType(funct7="0000001", funct3="001", opcode="0110011", name="MULH"),
    RType(funct7="0000001", funct3="010", opcode="0110011", name="MULHSU"),
    RType(funct7="0000001", funct3="011", opcode="0110011", name="MULHU"),
    RType(funct7="0000001", funct3="100", opcode="0110011", name="DIV"),
    RType(funct7="0000001", funct3="101", opcode="0110011", name="DIVU"),
    RType(funct7="0000001", funct3="110", opcode="0110011", name="REM"),
    RType(funct7="0000001", funct3="111", opcode="0110011", name="REMU"),
]
```

Все данные для комманд я брал из документации из ТЗ



RV32M Standard Extension						
0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU

The RISC-V Instruction Set Manual Volume I | © RISC-V

Chapter 28. RV32/64G Instruction Set Listings | Page 145

0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

Начнем парсить

```
text = self.sections.get(".text")
addr = text.get("sh_addr")
cmds = []
labels = {}
lLabels = {}
for i in range(0, text.get("sh_size"), 4):  # каждая команда 4 байта
    args = []
    name = "unknown command"
    bin_command = bin(self.get_from_data(text.get("sh_offset") + i,
4))[2:].rjust(32, "0")
    opcode = bin_command[25:]  # под opcode выделено 7 бит
    possible_commands = list(filter(lambda s: s.opcode == opcode,
commands.RV32I + commands.RV32M))
    # Тут учитываем, что у команд, с совпавшим opcode одинаковый
тип
    type_of_command = None
    if not possible_commands:
        name = "unknown command"
    else:
        type_of_command = type(possible_commands[0])
```

Я учитываю, что у комманд с одинвковым opcode всегда один и тот

же тип, поэтому изначально я из всего списка комманд выбираю те, которые подходят по opcode, а затем определяю тип аозможной команды и делаю перебор по типам.

Среди наших комманд (RV32I и RV32M) есть простые случаи: это комманды SType, BType, UType, JType, RType и так названная мной: FENCEType и сложная: IType

Просты случаи разбираются легко, ведь в них нет пересечения возможных комманд: все комманды с разным opcode делают разное, достаточно лишь внутри распарсить комманду на части, пример:

```
elif type(possible_commands[0]) is commands.SType:
    funct3 = bin_command[17:20]
    possible_commands = list(filter(lambda s: s.funct3 == funct3,
possible_commands))
    if possible_commands:
        rd = bin_command[20:25]
        rs2 = bin_command[7:12]
        rs1 = bin_command[12:17]
        funct7 = bin_command[0:7]

        imm = int(funct7 + rd, 2)
        imm = imm if imm < 2 ** 11 else imm - 2 ** 12

        name = possible_commands[0].name
        args = [register_name(int(rs2, 2)), register_name(int(rs1, 2)),
str(imm)]
    else:
        name = "unknown command"
```

imm помогает опрделять табличка из начала документации:

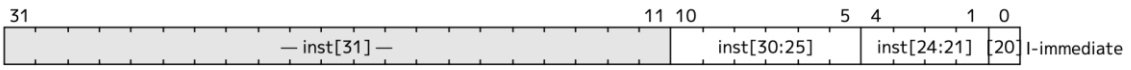
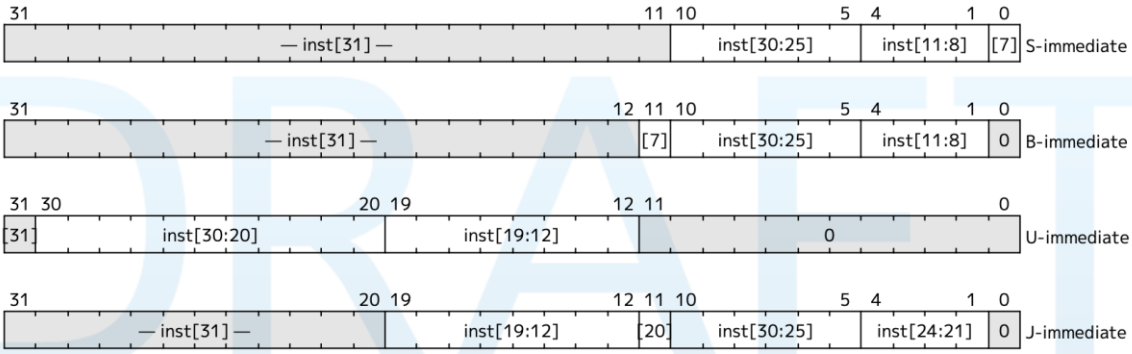
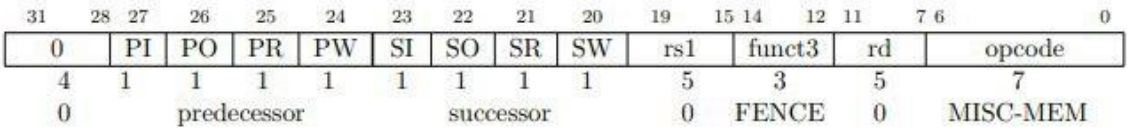


Figure 1. Types of immediate produced by RISC-V instructions.



Fence я парсил исходя из этой таблички:



31–28	27	26	25	24	23	22	21	20	19–15	14–12	11–7	6–0
0	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
4	1	1	1	1	1	1	1	1	5	3	5	7
0	predecessor				successor				0	FENCE	0	MISC-MEM

```
elif type(possible_commands[0]) is commands.FENCEType:
    funct3 = bin_command[17:20]
    possible_commands = list(filter(lambda s: s.funct3 == funct3,
possible_commands))
    if bin_command == "10000011001100000000000000001111":
        name = "FENCE.TSO"
    elif bin_command == "00000001000000000000000000001111":
        name = "PAUSE"
    elif possible_commands:
        name = possible_commands[0].name
        first_code = bin_command[4:8]
        second_code = bin_command[8:12]
        first_word = ""
        second_word = ""
        for k, item in enumerate(["i", "o", "r", "w"]):
            if first_code[k] == "1":
                first_word += item
            if second_code[k] == "1":
```

```
            second_word += item
        args = [first_word, second_word]
```

С IType было чуть сложнее, для этого у данной комманды я проверял является ли часть imm_11_0 данной нам в таблице комманд

```
elif type(possible_commands[0]) is commands.IType:
    funct3 = bin_command[17:20]
    possible_commands = list(filter(lambda s: s.funct3 == funct3,
possible_commands))
    rd = bin_command[20:25]
    rs1 = bin_command[12:17]
    shamt = bin_command[7:12]
    if not possible_commands:
        name = "unknown command"
    elif len(possible_commands) > 1:
        filtered_commands1 = list(filter(lambda s: s.imm_11_0 ==
bin_command[:7], possible_commands))
        filtered_commands2 = list(filter(lambda s: s.opcode == "1110011",
possible_commands))
        if filtered_commands1:
            imm = int(bin_command[0] * 20 + bin_command[:12], 2)
            imm = imm if imm < 2 ** 31 else imm - 2 ** 32

            name = filtered_commands1[0].name
            args = [register_name(int(rd, 2)), register_name(int(rs1, 2)), imm]
        elif filtered_commands2:
            if bin_command == "00000000000000000000000001110011":
                args = []
                name = "ECALL"
            elif bin_command == "00000000000100000000000001110011":
                args = []
                name = "EBREAK"
            else:
                name = "unknown command"
        else:
```

```
            name = "unknown command"

    else:
        imm = bin_command[0] * 21 + bin_command[1:12]
        imm = int(imm, 2)
        imm = imm if imm < 2 ** 31 else imm - 2 ** 32
        name = possible_commands[0].name
        args = [register_name(int(rd, 2)), register_name(int(rs1, 2)), imm]
```

После этого создавал строку для вывода комманд исходя из форматов заданных в тз и писал в файл:

```
text_out = ""
text_out += ".text\n"s
format_0_args = "    {:05x}:\t{:08x}\t{:>7s}\n"
format_2_args_label = "    {:05x}:\t{:08x}\t{:>7s}\t{:s}, 0x{:x} <{:s}>\n"
format_3_args_label = "    {:05x}:\t{:08x}\t{:>7s}\t{:s}, {:s}, 0x{:x},
<{:s}>\n"
format_load_store_jalr = "    {:05x}:\t{:08x}\t{:>7s}\t{:s}, {:d}({:s})\n"

format_2_args = "    {:05x}:\t{:08x}\t{:>7s}\t{:s}, {:s}\n"
format_3_args = "    {:05x}:\t{:08x}\t{:>7s}\t{:s}, {:s}, {:s}\n"
for i, command in enumerate(cmds):
    command_addr = addr + i * 4

    if command_addr in labels:
        label = labels.get(command_addr)
        label_format = "\n{:08x} \t<{:s}>:\n"
        label_text = label_format.format(command_addr, label)
        text_out += label_text

    int_command = int(command.get("bin_command"), 2)
    command_name = command.get("name").lower()
    command_args = command.get("args")
    args_size = len(command_args)
    type_of_command = command.get("type")

    command_text = ""
```

```
    if command_name == "unknown command" or args_size == 0:
        command_text = format_0_args.format(
            command_addr,
            int_command,
            command_name
        )
    elif (type_of_command == commands.JType) or (type_of_command
== commands.BType):
        addrs = command_addr + command_args[-1]
        if args_size == 2:
            command_text = format_2_args_label.format(
                command_addr,
                int_command,
                command_name,
                command_args[0],
                addrs,
                labels.get(addrs)
            )
        if args_size == 3:
            command_text = format_3_args_label.format(
                command_addr,
                int_command,
                command_name,
                command_args[0],
                command_args[1],
                addrs,
                labels.get(addrs)
            )
    elif type_of_command == commands.SType or
command.get("bin_command")[25:] in ["0000011", "1100111"]:
        command_text = format_load_store_jalr.format(
            command_addr,
            int_command,
            command_name,
            command_args[0],
            int(command_args[2]),
            command_args[1]
        )
    elif args_size == 3:
```

```
            command_text = format_3_args.format(
                command_addr,
                int_command,
                command_name,
                command_args[0],
                command_args[1],
                str(command_args[2])
            )
        elif args_size == 2:
            command_text = format_2_args.format(
                command_addr,
                int_command,
                command_name,
                command_args[0],
                str(command_args[1])
            )
        text_out += command_text
    text_out += "\n"
    self.out = text_out + "\n" + symtab_out
```

Результат работы на тестовых данных из репозитория:

```
.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    iorw, iorw
   100a8:  00000073     ecall

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    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)


.symtab

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
```