Лабораторная работа №4

ISA

Цель работы: знакомство с архитектурой набора команд RISC-V (unpriv-isa-asciidoc.pdf).

Инструментарий и требования к работе: работа выполняется на C/C++ (C11 и новее / C++20), Python (3.11.5) или Java (Temurin-17.0.8.1+1). Требования для всех работ: Правила оформления и написания работ.

Описание работы

Необходимо написать программу-транслятор (дизассемблер), с помощью которой можно преобразовывать машинный код (извлеченный из elf-файла) в текст программы на языке ассемблера.

Должен поддерживаться следующий набор команд RISC-V: RV32I, RV32M. Обратите внимание, расширения (Zifence, Zicsr) поддерживать не нужно. Подробнее (volume 1): https://riscv.org/technical/specifications/

Кодирование: little endian.

Вывод регистров: ABI. Регистр `x8` выводится как `s0`.

Псевдонимы команд: псевдонимы команд парсить не нужно.

Обрабатывать нужно только секции `.text`, `.symtab`.

Для каждой строки кода указывается её адрес в hex формате.

Обозначение меток нужно найти в Symbol Table (.symtab). Если же название метки там не найдено, то используется следующее обозначение: L%i, например, `L2`, `L34`. Нумерация начинается с 0. Для каждой метки перед названием указывается адрес (пример ниже).

Если встреченная инструкция не известна, то нужно вывести вместо инструкции “invalid_instruction”.

Шаблон файла дизассемблера

Файл должен состоять из двух частей: `.text` и `.symtab`, отделенных друг от друга одной пустой строкой. Сначала идет `.text`, затем `.symtab`.

Ниже приведены комментарии (строки, начинающиеся с ;) и форматы оформления. Формат строк указан по правилам printf (Си).

```
.text
; с меткой: "\n%08x \t<%s>\n", аргументы: адрес, метка
; без метки: адрес, hex код инструкции, инструкция, аргументы (ниже)
;     инструкции с 3 аргументами: "   %05x:\t%08x\t%7s\t%s, %s, %s\n"
;     инструкции с 2 аргументами: "   %05x:\t%08x\t%7s\t%s, %s\n"
;     load/store/jalr инструкции:  "   %05x:\t%08x\t%7s\t%s, %d(%s)\n"
;     J* инструкции с меткой: "   %05x:\t%08x\t%7s\t%s, 0x%x <%s>\n"
;     B* инструкции с меткой: "   %05x:\t%08x\t%7s\t%s, 0x%x, <%s>\n"
;     fence инструкции:  "   %05x:\t%08x\t%7s\t%s, %s\n"
;     без аргументов:  "   %05x:\t%08x\t%7s\n"
;     неизвестная:  "   %05x:\t%08x\t%-7s\n"
; immediate (константы): dec формат
; offset (в переходах J*, B*, в lui и auipc): hex формат
; пример ниже (отображение в ТЗ передаёт суть)
; пример в соответствии с форматом лежит в репозитории
;  если формат в файле расходится с текущим условием, то приоритетнее ; вывод в файле репозитория

00010074          <main>
   10074:         00000013       addi zero, zero, 0
   10078:         00100137        lui sp, 0x11
   100a8:         fcf42e23         sw a5, -36(s0)

; между секциями text и symtab 2 пустых строки
.symtab
; заголовок таблицы
; "\nSymbol Value          Size Type  Bind  Vis        Index Name\n"
; строки таблицы
; "[%4i] 0x%-15X %5i %-8s %-8s %-8s %6s %s\n"
; пример ниже (отображение в ТЗ передаёт суть), полный в репозитории

Symbol Value    Size Type     Bind      Vis       Index Name
[   0] 0x0         0 NOTYPE   LOCAL     DEFAULT   UNDEF
[   1] 0x10074     0 SECTION  LOCAL     DEFAULT       1
[   2] 0x112F8     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] 0x11AF8     0 NOTYPE   GLOBAL    DEFAULT     ABS __global_pointer$
```

Для вывода результата настоятельно рекомендуется использовать printf в C и C++, System.out.printf в Java и print в Python (https://stackoverflow.com/a/37848366). Использовать другие варианты не запрещается, но результат должен быть эквивалентным.

Про L метки: Когда в коде кто-то захочет перейти на определённый адрес, у которого нет метки, то тогда ставим метку L%i. Рассмотрим следующий случай (вывод утилиты objdump):

```
100fc:  fea794e3                bne  a5,a0,100e4 <mmul+0x38>
```

В bne задан offset на адрес, для которого явно не определена метка. Значит на 100e4 назначается метка (например, L0) и вашем коде дизасм может выглядеть одним из следующих случаев (приведена часть строки):

```
bne a5, a0, 0x100e4, <L0>
```

Числа выводятся в следующем виде:

- отрицательные (пример для -3): dec: -3, hex (хранение в дополнении до 2): `0xfffffffd`
- положительные (пример для 11): dec: 11, hex (хранение в дополнении до 2): `0xb`

Fence: predecessor и successor выводится как буквы установленных битов. Пример: `fence rw, orw`.

Аргументы программе передаются через командную строку:

<имя_входного_elf_файла> <имя_выходного_файла>

Модификация ППА

Должен поддерживаться следующий набор команд RISC-V: RV32I, RV32M, а также RV32A и расширения Zifence, Zihintpause.

Содержание отчета

1. Минититульник (таблица с ФИО, группой и названием работы из шаблона).
2. Ссылка на репозиторий.
3. Инструментарий (язык и версия компилятора/интерпретатора).
4. *Результат работы написанной программы* (то, что выводится в стандартный поток вывода на тесте из репозитория или вашего теста с большим числом команд).
5. *Описание работы написанного кода*: ЧТО БЫЛО РЕАЛИЗОВАНО (32i, 32m, или всё из этого), и как вы реализовали (разбор elf файла, парсинг команд). При описании работы написанного кода может быть полезно приложить небольшие рисунки для иллюстрации пояснений и ссылки на соответствующие документы (а не просто фразу “взял из спецификации).
6. Ссылки на используемые источники: по risc-v и elf. Фраза “взял(а) из спецификации” не будет принята.

Порядок сдачи работы

1. Выполнить работу.
2. Оформить миниотчёт в формате pdf.
3. Загрузить файл отчета и файлы с исходным кодом (расширения *.c/*.h/*.cpp/*.hpp или .py) в выданный вам репозиторий в корень.
4. Запустить автотесты (проверяется полное совпадение с реф. файлом). Подробнее: Автотесты на GitHub - comp-arch-course

(gitbook.io) ВАЖНО: закрытый тест (в отличие от теста на GH содержит все команды) но если не проходит автотест на GH, то запуска на закрытом тесте не будет.

5. Отправить на проверку работу (в открытом PR отмечаем Assignee Викторию и ставим label submit) и ждём ответа.

В репозиторий необходимо загружать файл с отчётом в формате pdf, названным в формате **“Фамилия_Имя_Группа_НомерРаботы.pdf”**. Номер группы – только 2 последние цифры группы, номер работы – порядковый номер лабораторной работы: 1, 2 и т.д.

В репозитории в директории “test_data”:

- `test_elf` – файл, который используется в автотесте на GH как входной.
- `test.cc` – файл, из которого получен `test_elf` (просто для ознакомления)
- `ref_disasm.txt` – ожидаемый результат
- `dump_disasm.txt` – результат отработки утилиты `riscv64-unknown-elf-objdump` по `test_elf` (просто для ознакомления)
- `read_disasm.txt` – результат отработки утилиты `riscv64-unknown-elf-readelf` по `test_elf` (просто для ознакомления)
- (могут быть изначально в репозитории) `disasm_ubuntu-22.04.txt` и `disasm_windows-latest.txt` – файлы, в которых будут результаты работы вашей программы после отработки на серверах GH, в коммите будет зафиксирован commit hash, код из которого запускался в

автотесте. Если программа на очередной запуске автотестов не сгенерит файл, то этот файл будет удалён из репозитория.

При запуске автотеста есть возможность убрать отправку сгенерированных файлов в удалённый репозиторий с сервера GH, если вам хватает логов в summary.

Доп. примеры для тестирования: elf_tests_2023.

Полезное

В этом разделе приведены программы, работа с которыми не обязательна для выполнения лабораторной, но может быть полезна для осознания лекционного материала и отладки лабораторной.

Кросс-компилятор RISC-V C и C++

Вы можете самостоятельно поэкспериментировать с использованием riscv-gnu-toolchain:

[Windows]:

- Prebuilt Windows Toolchain for RISC-V (10.1.0)

[Linux/MacOS]:

- Ubuntu – Package Search Results -- gcc-riscv64-linux-gnu
- GNU toolchain for RISC-V, including GCC (isa-spec 20191213)
- Prebuilt RISC-V GCC toolchains for x64 Linux.
- Releases · sifive/freedom-tools

Дизассемблер (без псевдоинструкций): `riscv64-unknown-elf-objdump --disassemble --target=elf32-littleriscv --architecture=riscv:rv32 --disassembler-options=no-aliases --disassembler-options=numeric test.elf`

Дизассемблер (предыдущее + symtab): `riscv64-unknown-elf-objdump --disassemble --target=elf32-littleriscv --architecture=riscv:rv32 --disassembler-options=no-aliases --disassembler-options=numeric --syms test.elf`

Таблица символов в формате из задания: `riscv64-unknown-elf-readelf --symbols --wide test.elf`

Компиляция: `riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 -O2 -x c test.s -o test.elf -static -lm -nostdlib`

Компиляция с сохранением “дизассемблера”: `riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 -O2 -S -x c test.s -o test.txt -static -lm -nostdlib`

Сохранить результат вывода на консоль в файл:

```
<command> > file.txt
```

Например сохранение сообщений компиляции в файл f.txt:

```
riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -O2 -x c test.s -o test.elf -static -lm -nostdlib > f.txt
```

Симулятор RISC-V [Linux, MacOS]

michaeljclark/rv8: RISC-V simulator for x86-64 (github.com)

Для его работы необходимо установить riscv-gnu-toolchain.

Графический симулятор RISC-V

GitHub - mortbopet/Ripes: A graphical processor simulator and assembly editor for the RISC-V ISA

Визуальный симулятор архитектуры и редактор ассемблерного кода, созданный для RISC-V.

Вы можете загрузить туда код на C/C++ и при помощи gnu toolchain скопилировать его или сразу загрузить elf файл. Загруженный код можно исполнить. На выбор доступны разные конфигурации конвейера и кэша.

Скачать: https://github.com/mortbopet/Ripes/releases/tag/v2.2.5