Руководство по Venus

Данная руководство посвящено использованию CLI и веб-интерфейсов Venus.

§ Расширение Visual Studio Code

Venus доступен и как расширение для `Visual Studio Code`. Найти можно в разделе расширений по названию `Visual Studio Code`. Данное расширение позволяет отлаживать код и отслеживать состояние регистров и памяти.

§ Веб-интерфейс Venus

Веб-интерфейс Venus доступен по ссылке https://venus.cs61c.org/. Исходный код доступен в свободном доступе и этот же сервис можно поднять локально:

```
https://github.com/61c-teach/venus
```

§ Вкладка "Editor"

- Вставьте ваш код во вкладку "Editor" (Редактор)
- Программы начинаются с первой строки ассемблерного кода независимо от метки, если только функция `main` не отмечена `.globl` (см. ниже). Это означает, что функция `main` должна быть помещена первой, если она не объявлена `.globl`.
    - Примечание: Иногда мы хотим предварительно распределить некоторые элементы в памяти до того, как программа начнет выполняться. Поскольку это распределение не является настоящим кодом программы, мы можем поместить его перед функцией `main`.
- Программы заканчиваются функцией `ecall` со значением аргумента 10. Этот сигнал означает завершение программы. Инструкции `ecall` аналогичны "системным вызовам" и позволяют нам делать такие вещи, как вывод в консоль или запрос памяти из кучи.
    - Исключением из правила выхода по `ecall` является случай, когда ваша функция `main` помечена как глобальная, в этом случае вы можете либо выйти по `ecall`, либо вернуть код завершения программы в `a0` (как возвращаемое значение функции main в C).
- Метки заканчиваются двоеточием (`:`).
- Комментарии начинаются со знака решётки (`#`).
- Вы НЕ МОЖЕТЕ вставить более одной инструкции на строку.
- Вы можете использовать комбинацию горячих клавиш для сохранения (`CMD` + `S`/`CTRL` + `S` в зависимости от вашей платформы, данная комбинация сохранит файл). Вы можете проверить имя актитвного файла в нижней части редактора: если он безымянный, то появится запрос на загрузку файла, а если вы быстро будете нажимать на кнопку сохранения, то так же получите этот запрос.
- Когда вы закончите редактирование, перейдите на вкладку "Симулятор" (Simulator), чтобы подготовиться к выполнению программы.

§ Вкладка "Simulator"

- Когда вы впервые перейдете на вкладку "Simulator" из вкладки "Editor", вы увидите кнопку "Assemble and Simulate from Editor" (Сборка и симуляция кода из редактора), которая сообщит об ошибках в вашем коде или повторно соберет его.
- Точки останова можно установить, щелкнув на нужной строке кода перед тем, как нажать кнопку "Run". Вы также можете вставить в код инструкцию `ebreak`, чтобы автоматически заставить Venus сделать паузу.
    - Вы можете имитировать условные точки останова, поместив `ebreak` внутри ветви, как показано ниже:

```
# ...code

li t0, 3 # Stops execution if a0 == 3
bne a0, t0, after_breakpoint
ebreak

after_breakpoint:
# ...code
```

- Кнопка "Run" запускает вашу программу до тех пор, пока вы не достигнете точки останова, подобно команде "run" в GDB.
- Кнопка "Step" приведет к переходу к следующей ассемблерной инструкции.
- Кнопка "Prev" возвращает одну ассемблерную инструкцию.
- Кнопка "Reset" завершает текущее выполнение, очищает все точки останова и возвращает к началу выполнения.
- Кнопка "Dump" создает дамп шестнадцатеричного представления каждой инструкции в вашей программе.
- Вы можете увидеть содержимое регистров, памяти и кэша в боковой панели справа.
    - Вы можете вручную указывать значения в регистрах, чтобы повлиять на выполнение программы.
    - На вкладке "Memory" используйте поля "Jump to" или "Address" для быстрого перехода к определенному адресу в вашей программе.
    - Для всех меню используйте "Display Settings" (Настройки отображения) в нижней части панели для переключения между шестнадцатеричными, ASCII, десятичными дополнительного кода и беззнаковыми десятичными интерпретациями ваших значений.
- Кнопка "Trace" приведет к сбросу значений в регистрах и памяти на основе заданного формата, более подробно описанного в разделе о трассировки.

§ Вкладка "Chocopy"

Вы можете не обращать внимания на эту вкладку. Ее суть в том, что вы можете скомпилировать подмножество Python в ассемблер RISC-V, который затем можно выполнить.

§ Вкладка "Venus"

Большинство основных функций, необходимых для написания ассемблерных программ , находятся на вкладках "Editor" и "Simulator". Вкладка "Venus" представляет собой терминалоподобный интерфейс, позволяющий работать с несколькими файлами для создания более сложных программ.

§ Команды терминала

- Набрав `help` в терминале, вы увидите список команд (вам может понадобиться прокрутить терминал вправо, чтобы увидеть их все), а `help [КОМАНДА]` даст вам информацию об определённой команде.
- Здесь перечислены несколько распространенных команд:
  - Манипуляции с файлами:
    - `ls`: Показывает содержимое указанного каталога (или текущего каталога, если аргументы не указаны).
    - `cd`: Изменяет каталог, в котором находится ваш терминал. Как и в обычном терминале, вы можете использовать `Tab` для автозаполнения имен папок.
    - `touch`: Создает новый пустой файл, который мы можем изменить позже.
    - `xxd`: Выводит содержимое файла в шестнадцатеричном виде.
    - Другие полезные команды (аналогичные тем, которые вы найдете в стандартной оболочке): `cat`, `clear`, `cp`, `mkdir`, `mv`, `rm`, `pwd`, `zip`, `unzip`.
  - Редактор и симулятор Venus:
    - `edit`: Открывает указанный файл на вкладке "Editor" (например, `edit file.s`). Обратите внимание, что сочетания клавиш `CMD` + `S` и `CTRL` + `S` будут работать для сохранения файла в виртуальной файловой системе, если вы решите отредактировать другой файл позже. Но чтобы убедиться, что ваша работа сохранена, мы рекомендуем периодически сохранять локальные копии каждого файла. Если вы смонтировали свою локальную файловую систему, то сохранение автоматически отразит изменения в вашей локальной файловой системе.
    - `run`: Запускает предоставленный(-ые) ассемблерный(-ые) файл(-ы) до завершения в терминале. Выводные данные также будут отображаться в терминале.
    - `vdb`: Связывает и собирает предоставленный(-ые) файл(-ы) сборки и открывает результат на вкладке "Simulator" для отладки.
  - Подключение к локальной файловой системе:
    - `upload`: Открывает запрос на загрузку файлов с локального компьютера в виртуальную файловую систему. Вы можете удерживать клавиши `CMD` или `CTRL`, чтобы выбрать несколько файлов для загрузки. После загрузки файла мы можем редактировать его в редакторе Venus с помощью команды edit.
    - `download`: Загружает указанные файлы локально. Вы можете указать несколько файлов для загрузки, например, `download file1.s file2.s`, и для каждого из них будет выведено соответствующее приглашение с вопросом, куда загрузить файлы.
    - `mount`: "Монтирует" вашу локальную файловую систему в виртуальную файловую систему Venus. Эта команда принимает URL, по которому запущен jar веб-сервер Venus, а затем имя, которое вы хотите присвоить смонтированной папке в виртуальной файловой системе. Например, если вы запустили jar Venus с

помощью `java -jar tools/venus.jar . -dm --port 6162` на *локальной машине* и хотели, чтобы смонтированная папка была доступна по адресу `vmfs` в виртуальной файловой системе, вы бы выполнили команду `mount http://localhost:6162 vmfs` в терминале *Venus*.

- Если вы опустите флаг `--port`, порт по умолчанию будет равен 6161. При запуске `mount local` по умолчанию будет использоваться этот порт.
- Если вы опустите аргумент path в команде `mount` в терминале Venus, по умолчанию ,будет использоваться `drive`.
- В браузере вам может быть предложено ввести ключ шифрования. Ключ должен быть указан в последней строке вывода терминала Java-сервера. Вы должны скопировать и вставить его. Вы также можете указать ключ в веб-терминале Venus, используя команду `mount local vmfs <ключ шифрования>`.

- `umount`: "Размонтирует" папку, которая была смонтирована с помощью команды `mount`.

§ Передача аргументов в вашу программу

Используйте поле "Simulator Default Args" для задания аргументов вашей программы. Они будут отражены при запуске программы на вкладке "Simulaor": `a0` будет инициализировано эквивалентом `argc`, а `a1` - эквивалентом `argv`.

§ Настройки

Параметры конфигурации для Venus можно найти в панели "Settings" на вкладке "Venus".

§ Общие настройки

- "Simulator Default Args" - передает аргументы вашей программе
- "Text Start" - определяет начало текстового сегмента
- "Max History" - указывает максимальное количество шагов выполнения, которые могут быть отменены; вы можете оставить отрицательным для отсутствия ограничения.
- "Aligned Addressing" - если включено, заставляет все обращения к памяти выравниваться по размеру типа данных, и будет выдавать ошибку во время выполнения, если обращение к памяти не является таковым.
- "Mutable Text" - если включено, позволяет перезаписывать текстовый сегмент (где хранятся инструкции) во время выполнения вашей программы.
- "Only Ecall Exit" - если включено, требует, чтобы `main` завершал программу через `ecall`, в отличие от завершения, когда ПК превысит конец текстового раздела.
- "Default Reg States" - если включен, инициализирует `a0` на argc, `a1` на argv, `gp` на начало статической секции памяти, `sp` на вершину стека, и `ra` на адрес возврата функции `main`, если найдена глобальная метка `main`; если выключен, все регистры инициализируются как 0
- " Allow Access" - если включено, разрешает доступ к памяти по адресам между стеком и кучей.
- " Max number of steps" - устанавливает ограничение на количество шагов, которые может выполнить ваша программа; оставьте отрицательное значение для отсутствия ограничения
- "Dark Mode " - активирует темный режим

§ Соглашение о вызовах

Включает проверку соглашения о вызовах функций (более подробную информацию см. в разделе Инструменты). При включении проверки, ее результаты будут выведены в консоль симулятора.

§ Трассировка

Позволяет выводить значения регистров и памяти во время выполнения программы. Более подробную информацию см. в разделе Трассировка.

§ CLI Venus

Последнюю версию Venus CLI можно загрузить по адресу https://venus.cs61c.org/jvm/venus-jvm-latest.jar. Для ее запуска требуется установленная Java.

§ Передача аргументов в вашу программу

Все, что написано после имени файла, будет передано вашей программе в качестве аргументов командной строки. Например, если вы хотите передать аргументы `arg1`, `arg2` и `arg3` в `test.s`, вы должны выполнить следующую команду:

```
$ java -jar venus.jar test.s arg1 arg2 arg3
```

`a0` будет инициализирован эквивалентом `argv`, а `a1` будет инициализирован эквивалентом `argc`. Чтобы предоставить Venus опции симулятора или ассемблера, поместите ваши флаги в команде между JAR и именем файла сборки. Например, если вы хотите запустить `test.s` с неизменяемым текстом (`-it`) и программой проверки соглашения о вызове (`-cc`), выполните следующую команду:

```
$ java -jar venus.jar -it -cc test.s arg1 arg2 arg3
```

Примечание: После имени файла могут быть вы можете написать флаги для настройки поведения сборки или симуляции Venus. Эти аргументы будут потребляться симулятором и не передаваться программе; это поведение может измениться в будущем с усовершенствованием анализатора аргументов Venus. Если вы хотите передать в программу аргумент, имя которого совпадает с именем флага Venus, добавьте `--` перед аргументами программы. Например, чтобы передать опцию "-it" в качестве первого аргумента `test.s`, вы должны выполнить команду:

```
$ java -jar venus.jar test.s -- -it
```

§ Распространенные ошибки Venus

- `Ran for more than max allowed steps!` Venus автоматически завершит работу, если ваша программа проработает слишком много шагов. Это ожидаемо для больших программ, и

вы можете обойти это с помощью флага `-ms`. Если вы получаете эту ошибку при малых входных данных, возможно, у вас бесконечный цикл.

- `Attempting to access uninitialized memory between the stack and heap.`: Ваш код пытается прочитать или записать адрес памяти между указателями стека и кучи, что вызывает ошибку сегментации. Проверьте, достаточно ли памяти вы выделяете, и обращаетесь ли вы к правильным адресам.
- `The magic value for this malloc node is incorrect! This means you are overriding malloc metadata OR have specified the address of an incorrect malloc node!` Ваш код изменяет метаданные узла malloc. Эта ошибка может возникнуть при выполнении любой из команд `alloc` или `free`, поскольку venus реализует форму malloc в виде связанного списка. Метаданные находятся прямо под указателем на это место (нижний адрес), поэтому если вы записываете в неправильное место, вы можете повредить эти данные. В Venus есть метод обнаружения некоторых повреждений данных, поэтому вы и получаете эту ошибку. Проверьте, правильно ли вы индексируете все выделенные (malloc) данные и не выходите ли вы за границы выделенных данных.

§ Инструменты

§ Проверка соглашения о вызове

Соглашение о вызове (calling convention) RISC-V определяет, какие регистры сохраняются *вызывающей* функцией, а какие - *вызываемой*. Ошибки, возникающие из-за несоблюдения соглашения о вызове, бывает трудно отследить самостоятельно, поэтому Venus предоставляет флаг `-cc` (или `--callingConvention`) для автоматического обнаружения определенных видов ошибок соглашения о вызове. Прежде чем использовать программу проверки соглашения о вызове, вы должны знать две вещи:

1. Venus не может обнаружить *все* нарушения соглашения о вызовах - тот факт, что вызовы функций в ассемблере являются, по сути, просто переходами к меткам, означает, что Venus должен быть очень консервативным в выборе того, что следует определить как вызов функции. Программа проверки CC служит отличной проверкой на вменяемость (подобно тому, как Valgrind проверяет ошибки памяти в языке C), но есть некоторые ошибки, которые в конечном итоге придется искать вручную.
2. В связи с вышесказанным, программа проверки CC будет надежно проверять нарушения соглашений о вызовах только в теле функций, экспортированных с помощью директивы `.globl`. Иногда можно увидеть сообщения об ошибках в функциях, которые не экспортируются, но обычно это происходит потому, что они вызываются функцией, объявленной `.globl`.

§ Сообщения о нарушениях

В настоящее время существует три типа сообщений об ошибках, которые может выдать проверка соглашения о вызове, каждый из которых объясняется ниже. В каждом примере `func` относится к функции, в которой было выдано сообщение об ошибке. Вы можете самостоятельно запустить эти примеры в Venus, изменив следующий фрагмент кода:

```
.globl func
main:
    jal func
    li a0, 10 # Code for exit ecall
    ecall

func:
    # Paste the example definition of func
```

"Setting of a saved register (s0) which has not been saved!"

`func` перезаписала сохраненённый вызываемый регистр (любой из s0 - s11). Чтобы исправить это, убедитесь, что `func` сохраняет значения всех регистров, которые она перезаписывает, освобождая место на стеке и сохраняя их в памяти в прологе кода, а затем восстанавливая их значения и восстанавливая указатель стека в эпилоге кода.

Пример с ошибкой:

```
func:
    li s0, 100 # === Сообщение об ошибке здесь ===
    li s1, 128 # === Сообщение об ошибке здесь ===
    ret
```

Исправленный пример:

```
func:
    # НАЧАЛО ПРОЛОГА
    # Каждый скомбинированный регистр (4 байта каждый) должен быть
сохранен
    addi sp, sp, -8
    sw s0, 0(sp)
    sw s1, 4(sp)
    # КОНЕЦ ПРОЛОГА
    li s0, 100
    li s1, 128
    # НАЧАЛО ЭПИЛОГА
    lw s1, 4(sp)
    lw s0, 0(sp)
    addi sp, sp, 8
    # КОНЕЦ ЭПИЛОГА
    ret
```

"Save register s0 not correctly restored before return! Expected , Actual "

Значение `s0` в функции, вызвавшей `func`, было разным до и после вызова `func`. Это сообщение об ошибке дополняет предыдущее, но вместо него будет сообщено на строке инструкции `ret` (или

`jalr ra`).

Пример с ошибкой:

```
func:
    # НАЧАЛО ПРОЛОГА
    addi sp, sp, -8
    sw s0, 0(sp)
    sw s1, 4(sp)
    # КОНЕЦ ПРОЛОГА
    li s0, 100
    li s1, 128
    # НАЧАЛО ЭПИЛОГА
    # Забыли восстановить значения s0 и s1!
    addi sp, sp, 8
    # КОНЕЦ ЭПИЛОГА
    ret # === Здесь дважды сообщается об ошибке ===
```

Исправленный пример: Тот же, что и для предыдущей ошибки. Просто не забудьте включить и пролог и эпилог.

"Usage of unset register t0"

`func` попыталась прочитать из регистра, который не был установлен вызывающей функцией и еще не был инициализирован в теле `func`. Даже если функция, вызывающая `func`, устанавливает значения в таких регистрах, как `t0` или `s0`, попытка использовать их значения внутри `func` будет нарушением барьера абстракции: `func` не должна знать ничего о значениях регистров вызывающей функции, за исключением аргумента, указателя стека и адреса возврата.

Пример с ошибкой:

```
func:
    mv a0, t0 # === Сообщение об ошибке здесь ===
    ret
```

Исправленный пример: Подобная ошибка обычно возникает из-за опечатки (инструкция читается не из того регистра, или порядок аргументов инструкции был случайно изменен), или нарушения барьеров абстракции, как описано выше. Исправления зависят от того, для чего предназначалась функция.

§ Трассировка

Вы можете настроить трассировку как в веб-версии, так и в CLI-версии Venus для вывода значений в регистрах и памяти во время выполнения вашей программы. В веб-интерфейсе вы можете включить трассировку в разделе "Traces" в панели настроек; в командной строке вы можете включить и настроить трассировку с помощью флагов `--trace` и `--tracepattern`.

Шаблоны трассировки поддерживают несколько специальных символов, напоминающих строку формата C. Имеются следующие шаблоны:

- `\t`: отступ вкладки
- `\n`: новая строка
- `%0%` - `%31%`: значение в соответствующем регистре
- `%line%`: номер строки выполняемого кода
- `%pc%`: программный счетчик текущей инструкции
- `%inst%`: код текущей инструкции
- `%output%`: вывод сообщения ecall
- `%decode%`: декодированный машинный код текущей инструкции

§ Написание больших программ для RISC-V

Когда мы начнем писать более сложные программы для RISC-V, возникнет необходимость разделить наш код на несколько файлов и отлаживать/тестировать их по отдельности. Вкладка "Venus" веб-редактора предоставляет доступ к терминалу (и соответствующей виртуальной файловой системе), который позволяет нам редактировать и тестировать программы, собранные из нескольких файлов.

Примечание: Чтобы не потерять ничего из своей работы, не забывайте *почаще* сохранять локальные копии файлов. Чтобы быть уверенным вдвойне, сохраняйте свои локальные работы с помощью git и загружайте эти сохранения!

§ Работа с несколькими файлами

Директива `.globl` определяет функции, которые мы хотим экспортировать в другие файлы, аналогично включению функции в заголовочный файл в C.

Venus поддерживает директиву `.import` (технически не являющуюся частью спецификации RISC-V) для доступа к другим ассемблерным файлам, аналогично директиве `include` в C. Она делает доступными только метки с пометкой `.globl`.

§ Поведение, не определенное в спецификации RISC-V

§ Нулевые указатели

В отличие от большинства реальных программ (и спецификации RISC-V), разыменование нулевого указателя НЕ приводит к ошибке сегментации. Это происходит потому, что по умолчанию начало текстового сегмента установлено в `0x0000_0000`, в то время как в реальной системе оно, скорее всего, будет иметь значение `0x10000_0000`. Это значение можно настроить в панели настроек веб-интерфейса.

§ Системные вызовы

Инструкция `ecall` используется для выполнения системных вызовов или запроса других привилегированных операций, таких как доступ к файловой системе или запись вывода на консоль.

Чтобы выполнить системный вызов, загрузите соответствующий номер системного вызова в регистр `a0`, затем разместите аргументы системного вызова по порядку в остальных регистрах аргументов

(a1, a2 и т.д.) по мере необходимости. Например, следующий фрагмент ассемблера выведет в stdout число 1024:

```
li a0, 1    # номер системного вызова для вывода целого числа
li a1, 1024 # целое число, которое мы выводим
ecall       # выполнить системный вызов
```

Системные вызовы, возвращающие значения, помещают значение в a0, как и любая другая функция.

Примечание: Соглашение о вызове для RISC-V в Linux (описанное в man-руководствах Linux в разделе "Architecture calling conventions") фактически определяет, что номер системного вызова должен передаваться через регистр a7. Venus использует a0 для простоты и следует традициям симулятора SPIM.

§ Список системных вызовов

Полный список доступных вызовов см. в wiki Venus.