Лабораторная работа №3: Ассемблер RISC-V

§ Цели

- Ознакомиться с использованием симулятора Venus
- Получите представление о том, как перевести код на языке Си в RISC-V.
- Научиться писать функции RISC-V, по правилам о соглашении вызова процедур.

§ Введение в Ассемблер

В нескольких лабораторных работах мы будем работать с файлами ассемблера RISC-V, каждый из которых имеет расширение файла `.s`. Для их запуска мы будем использовать Venus, образовательный ассемблер и симулятор RISC-V. Вы можете запустить Venus локально из своего терминала или из браузера Venus, а следующие инструкции помогут вам выполнить все шаги по его настройке. Однако, для наших лабораторных вам может быть удобнее использовать веб-редактор. Для этой лабораторной также можно использовать расширение `RISC-V Venus Simulator` для `Visual Studio Code`.

§ Ассемблер/Основы Venus

Чтобы начать работу с Venus, ознакомьтесь с разделами "Вкладка "Editor"" и "Вкладка "Simualtor"" в руководстве Venus. Можете прочитать руководство целиком, но этих разделов уже должно быть достаточно для начала вашей работы.

> **Предупреждение**: Для следующих упражнений, пожалуйста, убедитесь, что ваш готовый код сохранен в файле на вашей локальной машине и добавлен в репозиторий. В противном случае у нас не будет доказательств того, что вы выполнили лабораторные задания.

§ Упражнение 1: Подключение ваших файлов к Venus

Вы можете "примонтировать" папку с вашего локального устройства к веб-фронтенду Venus, чтобы правки, которые вы делаете в редакторе браузера Venus, отражались в вашей локальной файловой системе, и наоборот. Если вы этого не сделаете, то файлы, созданные и отредактированные в Venus, будут теряться каждый раз, когда вы закрываете вкладку, если вы не скопируете и вставите их в локальный файл.

В этом упражнении вы выполните процесс подключения вашей файловой системы к Venus, что избавит вас от необходимости копировать и вставлять файлы между локальным диском и редактором Venus.

Если по какой-то причине эта функция не работает у вас (она относительно новая, и есть вероятность, что в ней есть баги), то до конца этого задания везде, где говорится об открытии файла в Venus, вы должны скопировать и вставить его содержимое в веб-редактор Venus, а затем вручную скопировать и вставить эти изменения обратно на локальную машину.

Вот что вам нужно сделать:

- Если вы еще не склонировали репозиторий Labs на локальную машину, откройте терминал на локальной машине и склонируйте его при помощи `git clone`.
    - Пользователи Windows должны клонировать репозиторий вне WSL (рекомендуется Git Bash). Обратите внимание, что пути Windows также доступны из WSL (например, `C:/Users/oski/csa-2022/` в Windows - это `/mnt/c/Users/oski/csa-2022/` в WSL).
- В терминале пропишите `cd <папка с репозиторием лабораторной>`, после чего выполните команду `java -jar lab03/venus.jar . -dm`. Это откроет каталог вашей лабораторной для Venus через сетевой порт.
    - Вы должны увидеть большой логотип "Javalin".
    - Если вы увидите сообщение типа "port unable to be bound" (не удалось привязать порт), то вы можете указать другой номер порта, добавив к команде аргумент `--port PORT_NUM` (например, `java -jar lab03/venus.jar . -dm --port 6162` откроет файловую систему на порту 6162).
- Откройте https://venus.cs61c.org в вашем веб-браузере (рекомендуется использовать Chrome или Firefox, Safari не работает). В веб-терминале Venus выполните команду `mount local labs` (если вы выбрали другой порт, замените "local" на полный URL, например, `http://localhost:6162`). Это подключит Venus к вашей файловой системе.
    - В вашем браузере может появиться сообщение `Key has been shown in the Venus mount server! Please copy and paste it into here.` ("Ключ показан в сервере монитирования Venus! Пожалуйста, скопируйте и вставьте его сюда."). Вы должны увидеть ключ в последней строке вывода вашего локального терминала; просто скопируйте и вставьте его в диалог.
- Перейдите на вкладку "Files". Теперь вы должны увидеть каталог лабораторной работы в папке `labs`.
- Перейдите к `lab03` и убедитесь, что он работает, нажав кнопку `Edit` рядом с `fib.s`. Должна открыться вкладка `Editor`.
    - Если вы внесли какие-либо изменения в файл на вкладке `Editor`, нажатие `CMD` + `D` на Mac, либо `CTRL` + `D` на Windows/Linux обновит вашу локальную копию файла. Чтобы проверить, успешно ли прошло сохранение, откройте файл на локальной машине и посмотрите, совпадает ли он с тем, что у вас в веб-редакторе (к сожалению, сообщение обратной связи пока не реализовано).
    - **Примечание**: Если вы внесете какие-либо изменения в файл на локальной машине, и если тот же файл был открыт в редакторе Venus, вам придется снова открыть его из меню "Files", чтобы обновить его и получить новые изменения.
- Чтобы сделать так, чтобы файловая система пыталась перемонтироваться автоматически при каждом закрытии и повторном открытии Venus, включите "Save on Close" в панели настроек (на вкладке Venus). Это заставит веб-клиент Venus попытаться найти файловую систему, открытую при локальном запуске Venus, и выдаст ошибку о том, что он не смог подключиться к серверу, если не увидит его запущенным. Если это произойдет, просто выполните описанные выше действия, чтобы вручную перемонтировать файловую систему.

Как только вы открыли `fib.s`, вы готовы перейти к Упражнению 2!

§ Упражнение 2: Знакомство с Venus

В этом упражнении мы даем вам реализацию fib как на языке C, так и на ассемблере. Взгляните на оба файла.

В верхней части `fib.s` можно увидеть директивы `.data`, `.word`, `.text`.

- `.data`: Обозначает место объявления глобальных переменных
- `.word`: Выделяет и инициализирует место для 4-байтовой переменной в сегменте данных.
- `.text`: Указывает на начало кода.

Мы также добавили комментарии к `fib.s`, чтобы помочь вам лучше понять программу. Есть две новые инструкции:

- `la n`: загружает адрес метки, в которой находится n. Это псевдоинструкция, которая разбивается на инструкции `auipc` и `addi`. Обратитесь к справочникам чтобы узнать, что делает `auipc` (Харрис и Харрис Таблица B1 и 6.3.8).
- `ecall`: Инструкция `ecall` используется для выполнения системных вызовов или запросы на другие привилегированные операции, такие как доступ к файловой системе или запись вывода на консоль. В этом задании мы будем использовать `ecall` в основном для выхода из системы или для вывода целых чисел. Чтобы указать, какое действие должен выполнить `ecall`, вы должны передать код к `ecall` через `a0`. Чтобы завершить программу, вы должны установить `a0` в `10`. Чтобы вывести целое число, вы должны установить `a0` в 1 и `a1` в целое число, которое вы хотите вывести.

Запустить `fib.s` в Venus:

1. Откройте файл `fib.s` в редакторе Venus. Если вы не смогли смонтировать файловую систему в упражнении 1, то вы можете скопировать/вставить `fib.s` с локальной машины в редактор Venus напрямую.
2. Перейдите на вкладку "Simulator" и нажмите кнопку "Assemble & Simulate from Editor" (Собрать и просимулировать код из редактора). Это подготовит написанный вами код к выполнению. Если вы вернетесь на вкладку "Editor", ваша симуляция будет сброшена.
3. В симуляторе, чтобы выполнить следующую инструкцию, нажмите кнопку "step".
4. Чтобы отменить инструкцию, нажмите кнопку "prev". Обратите внимание, что это может отменять или не отменять операции, выполняемые `ecall`, такие как выход из программы или вывод в консоль.
5. Чтобы запустить программу до конца, нажмите кнопку "run".
6. Чтобы сбросить состояние машины в начальное, нажмите кнопку "reset".
7. Содержимое всех 32 регистров находится справа, а вывод консоли - внизу.
8. Чтобы просмотреть содержимое памяти, нажмите на вкладку "Memory" справа. Вы можете переходить к различным частям памяти с помощью выпадающего меню внизу.

§ Ход работы

Откройте файл `fib.s` в Venus и ответьте на следующие вопросы. Некоторые из вопросов потребуют запуска кода RISC-V с помощью вкладки симулятора Venus.

1. Запустите программу до конца. Какое число вывела программа? Что представляет собой это число?

2. По какому адресу `n` хранится в памяти? Подсказка: Пройдитесь по коду и посмотрите на содержимое регистра

3. Как вычислить 20-е число Фибоначчи?

§ Упражнение 3: Перевод с языка C на RISC-V

Откройте файлы `ex3.c` и `ex3.s`. Представленный ассемблерный код (файл .s) является переводом данной программы на языке C на язык RISC-V.

Помимо открытия файла на вкладке "Editor" и последующего запуска на вкладке "Simulator", как описано выше, вы также можете запустить `ex3.s` непосредственно в терминале Venus, зайдя в соответствующую папку, а затем запустив `run ex3.s` или `./ex3.s`. Набрав `vdb ex3.s`, вы также соберете файл, а затем перейдете на вкладку "Simulator".

§ Ход работы

Найдите и определите следующие компоненты этого ассемблерного файла и объясните, как они работают.

1. Регистр, представляющий переменную `k`.

2. Регистр, представляющий переменную `sum`.

3. Регистры, действующие как указатели на массивы `source` и `dest`.

4. Ассемблерный код для цикла, найденного в коде C.

5. Как указатели манипулируются в ассемблерном коде.

§ Упражнение 4: Задание по вариантам.

В этом задании вам предстоит написать RISC-V код, который вычисляет заданную математическую функцию. Вы должны написать две функции решающие предложенную задачу, одна должна работать итеративно (через цикла), другая рекурсивно (вызываю саму себя). Напишите проверяющую функцию, которая вызывает обе функции и проверяет результат. Параметры для функций задавайте Напишите свой код в `ex4.s`.

Вариант 1. Факториал.

Реализуйте функцию вычисления факториала. Эта функция принимает один целочисленный параметр `n` и возвращает `n!`.

Вариант 2. Сумма арифметической прогрессии

Реализуйте функцию вычисления суммы арифметической прогресии. Эта функция принимает один целочисленный параметр `n` и возвращает сумму чисел от `1` до `n`.

Вариант 3. Сумма геометрической прогрессии

Реализуйте функцию вычисления суммы геометрической прогресии с шагом 2. Эта функция принимает один целочисленный параметр `n` и возвращает произведение чисел от `1` до `2^n`. `n` считать не больше 31.

Вариант 4. Член арифметической прогрессии с шагом 3

Реализуйте функцию вычисления члена арифметической прогрессии с шагом 3 под номером n. Эта функция принимает один целочисленный параметр `n` и возвращает элемент прогрессии под этим номером. Операцию умножения использовать запрещено.

Вариант 5. Точная степень двойки

Дано натуральное число `n`. Функция должна вернуть `1`, если число `n` является точной степенью двойки, или `0` в противном случае.

Вариант 6. Сумма разрядов числа

Дано целое число `n`. Вычислите количество 1 в его двоичном дополнительном коде.

Вариант 7. Умножение через сумму

Даны два целых неотрицательных числа `m` и `n`. Вычислите произведение `n*m` не используя операцию умножения.

Вариант 8. Деление через разность

Даны два целых неотрицательных числа `m` и `n`. Вычислите результат деления `n / m` с округлением в меньшую сторону, не используя операцию деления.

Вариант 9. Остаток через разность

Даны два целых неотрицательных числа `m` и `n`. Найдите остаток от деления `n % m`, не используя операцию деления и деления по модулю.

Вариант 10*. Функция Аккермана

Данная задача решается только рекурсивно. Даны два целых неотрицательных числа `m` и `n`. Напишите код, вычисляющий A(m,n). Подберите значения `m` и `n`, при которых значение еще вычесляется

```
A(m, n) = n + 1, iff m == 0
A(m, n) = A(m-1, 1), iff m > 0 and n == 0,
A(m, n) = A(m-1, A(m, n-1)), iff m > 0 and n > 0,
```

§ Тестирование

Протестируйте свой код в симуляторе.

```
$ java -jar lab03/venus.jar lab03/factorial.s
```

§ Упражнение 5: Практика работы с массивами

Рассмотрим дискретно-значную функцию `f`, определенную на целых ограниченном множестве целых чисел. Вот определение функции (по вариантам, в остальных точках функция не определена и возвращает -1):

```
# V.1          #V.2           #V.3           #V.4
f(-3) = 6    |  f(-5) = 15  |  f(-6) = 1   |  f(-4) = -6
f(-2) = 61   |  f(-4) = 23  |  f(-5) = 26  |  f(-3) = 46
f(-1) = 17   |  f(-3) = 15  |  f(-4) = 41  |  f(-2) = -46
f(0) = -38   |  f(-2) = 48  |  f(-3) = -50 |  f(-1) = 25
f(1) = 19    |  f(-1) = 6   |  f(-2) = 32  |  f(0) = 30
f(2) = 42    |  f(0) = 49   |  f(-1) = -31 |  f(1) = 48
f(3) = 5     |  f(1) = 36   |  f(0) = 23   |  f(2) = 5

# V.5          #V.6           #V.7           #V.8
f(-2) = 23   |  f(-1) = -6  |  f(3) = 14   |  f(-11) = 50
f(-1) = 39   |  f(0) = 20   |  f(4) = 16   |  f(-10) = 2
f(0) = 1     |  f(1) = 7    |  f(5) = 29   |  f(-9) = 48
f(1) = 4     |  f(2) = -38  |  f(6) = 44   |  f(-9) = 17
f(2) = 36    |  f(3) = -32  |  f(7) = 35   |  f(-7) = 13
f(3) = 24    |  f(4) = 49   |  f(8) = 28   |  f(-6) = 35
f(4) = 26    |  f(5) = 21   |  f(9) = -41  |  f(-5) = -5

# V.9          #V.10          #V.11          #V.12
f(-6) = 49   |  f(-6) = 3   |  f(-7) = -19 |  f(-5) = 25
f(-3) = 46   |  f(-4) = 28  |  f(-5) = 28  |  f(-3) = 8
f(-1) = -33  |  f(-2) = -20 |  f(-3) = 47  |  f(-1) = 6
f(0) = -14   |  f(0) = 20   |  f(0) = -39  |  f(0) = 3
f(1) = -27   |  f(2) = 29   |  f(3) = 19   |  f(1) = -12
f(3) = 7     |  f(4) = 32   |  f(5) = 7    |  f(3) = -34
f(6) = 0     |  f(6) = 43   |  f(7) = 27   |  f(5) = 5
```

§ Ход работы

1. Реализуйте функцию в файле `discrete_fn.s` в RISC-V с условием, что ваш код **НЕ** должен использовать инструкции ветвления и/или перехода! Убедитесь, что ваш код сохранен локально. Мы привели несколько подсказок на случай, если у вас что-то не получится. Комментарий `#FIXME` указывает на места в коде, которые стоит исправить.

Обратите внимание, что вы можете сократить `jal ra, label` до `jal label`. Эти две строчки делают одно и то же.

▶ Подсказка 1

Все выходные значения хранятся в выходном массиве, который передается в `f` через регистр a1. Вы можете индексировать этот массив, чтобы получить выход, соответствующий входу.

▶ Подсказка 2

Вы можете получить доступ к значениям массива с помощью `lw`.

▶ Подсказка 3

`lw` требует, чтобы смещение было непосредственным значением. Когда мы вычисляем смещение для этой задачи, оно будет храниться в регистре. Поскольку мы не можем использовать регистр в качестве смещения, мы можем добавить значение, хранящееся в регистре, к базовому адресу, чтобы вычислить адрес индекса, который нас интересует. Затем мы можем выполнить `lw` со смещением `0`.

В следующем примере индекс хранится в `t0`, а указатель на массив - в `t1`. Размер каждого элемента составляет 4 байта. В RISC-V мы должны выполнять собственную арифметику указателей, поэтому (1) нам нужно умножить индекс на размер элементов массива. (2) Затем мы добавляем это смещение к адресу массива, чтобы получить адрес элемента, который мы хотим прочитать, (3) а затем читаем элемент.

```
slli t2, t0, 2 # шаг 1 (см. выше)
add t2, t2, t1  # шаг 2 (см. выше)
lw t3, 0(t2) # шаг 3 (см. выше)
```