ЛАБОРАТОРНАЯ РАБОТА

Архитектура набора команд на примере RISC-V

Никита Поляков

18.09.2020
ChipExpo-2020

ВВЕДЕНИЕ

МАТЕРИАЛЫ

1.

ПОДГОТОВКА К РАБОТЕ

Работа будет выполняться на компьютере с операционной системой Windows (XP и новее) или Linux Ubuntu. Выполнение на компьютерах с другими операционными системами также возможно, но не проверялось (при желании можете выполнить проверку самостоятельно).

Основным инструментом будет симулятор архитектуры RISC-V. Для его запуска понадобится поддержка Java (инструкции по установке см. ниже).

1. Установка Java

1. для Ubuntu: https://www.digitalocean.com/community/tutorials/java-ubuntu-apt-get-ru
2. для Windows: https://www.java.com/ru/download/help/download_options.xml#windows

2. Запуск эмулятора

1. Скачиваем репозиторий https://github.com/DigitalDesignSchool/ce2020labs либо через браузер, либо через Git утилиту.
    a. через браузер:
        i. переходим по адресу https://github.com/DigitalDesignSchool/ce2020labs
        ii. нажимаем на зеленую кнопочку Code
        iii. выбираем Download ZIP
        iv. распаковываем архив в папку, в которой будем работать.
    b. через командную строку Ubuntu:
        i. установка Git, если его нет

```
sudo apt update

sudo apt install git
```

ii. в командной строке выполняем

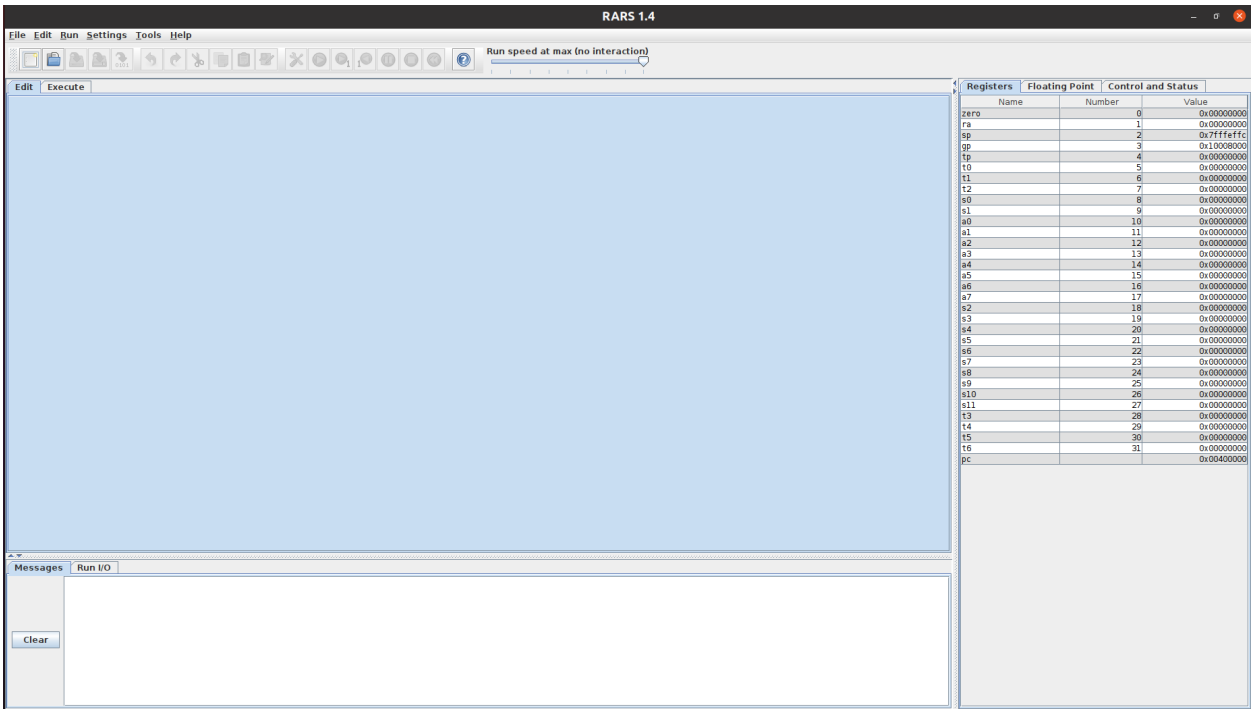
git clone https://github.com/DigitalDesignSchool/ce2020labs

создастся папка ce2020labs

2. Заходим в папку ce2020labs/day_3/arch/risc_v_lab
3. Сам запуск
    a. в Linux в командной строке выполняем

./rars.sh

    b. в Windows запускаем двумя щелчками мыши скрипт rars.bat



ПЛАН РАБОТЫ

1. Архитектура набора команд
2. Работа симулятора
3. Простейшие команды (ADD, ADDI, MV)
4. Условные операции и циклы
5. Работа с памятью

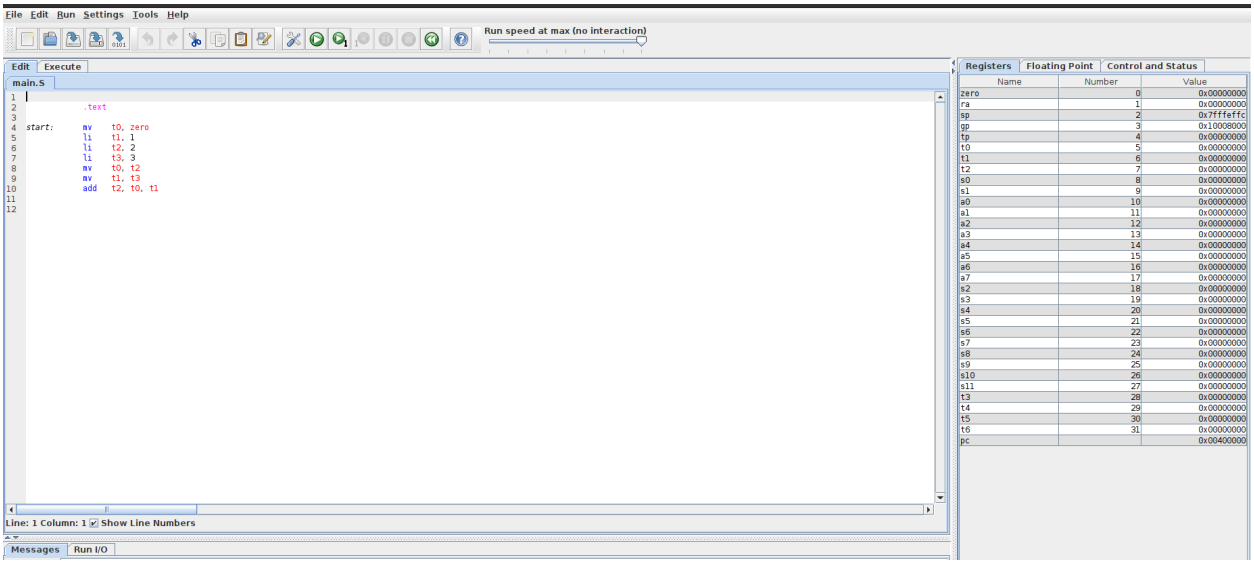
АРХИТЕКТУРА НАБОРА КОМАНД

РАБОТА СИМУЛЯТОРА

Симулятор позволяет выполнить трансляцию программы, написанной на языке ассемблера, в машинный код и эмулировать исполнение программы.

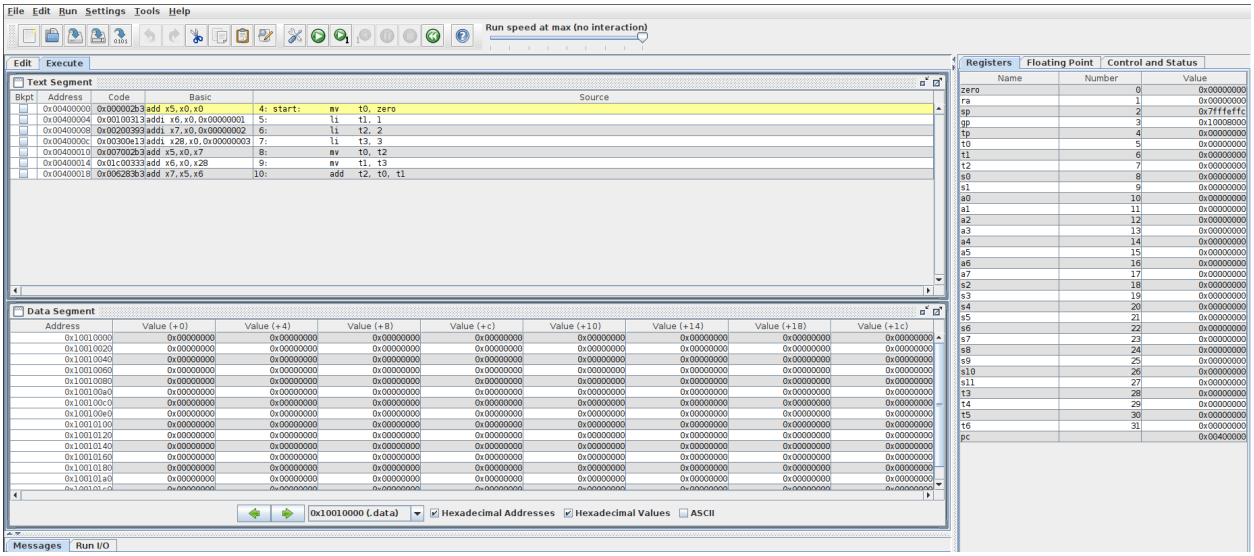
После запуска симулятора (инструкции по запуску см. выше) необходимо загрузить программу, выбрав в меню

File -> Open -> выбрать файл с расширением .S (например, ce2020labs/day_3/arch/risc_v_lab/prog/simple/main.S)



В основном окне появится код открываемой программы. В правой части отображаются все архитектурные регистры и их содержимое (значение в текущий момент).

Далее выполняем трансляцию программы в машинный код, выбрав в меню Run -> Assemble.



В открывшемся окне появится таблица, содержащая адрес инструкции в памяти, машинный код инструкции, дизассемблер инструкции (обратное представление кода на языке ассемблера) и соответствующая всему этому исходная строка программы.

Далее можно либо исполнить всю программу целиком, выбрав Run -> Go или нажав F5, или выполнить программу по шагам, выбрав Run -> Step или нажав F7. На каждом шаге выполнения программы в правой части будут отображаться текущее содержимое регистров. Желтым выделяется следующая строка, которая будет исполнена. В нижней части отображается текущее содержимое памяти.

ПРОСТЕЙШИЕ КОМАНДЫ (ADD, ADDI, MV, LI)

Команда на языке ассемблера состоит из названия команды, а также, при необходимости, операндов и регистра результата. Рассмотрим команду целочисленного сложения ADD:

```
add rd, r1, r2
```

где rd - регистр результата, r1 - первое слагаемое, r2 - второе слагаемое, например,

```
add t2, t0, t1
```

что эквивалентно a = b + c, где a хранится в регистре t2, b в регистре t0, а c в регистре t1.

Команда ADDI выполняет сложение содержимого регистра с константой (immediate operand, т.е. непосредственное значение или литерал):

```
addi rd, r1, imm
```

где rd - регистр результата, r1 - первое слагаемое, imm - константа (второе слагаемое), например,

```
addi t2, t0, 2
```

что эквивалентно a = b + 2, где a хранится в регистре t2, b в регистре t0.

Команда MV копирует значение одного регистра в другой:

```
mv rd, r1
```

Команда LI загружает непосредственное значение в регистр:

```
li rd, imm
```

Также доступны следующие команды:

`sub rd, rs1, rs2`	вычитание rd = rs1 - rs2
`and rd, rs1, rs2`	побитовое И rd = rs1 & rs2
`or rd, rs1, rs2`	побитовое ИЛИ rd = rs1 | rs2
`xor rd, rs1, rs2`	побитовое исключающее ИЛИ rd = rs1 xor rs2

Задание

Выполните симуляцию выполнения простейшей программы.

1. Запустите симулятор.
2. Откройте программу day_3/arch/risc_v_lab/prog/simple/main.S .
3. Выполните трансляцию программы. Сравните последние 2 столбца (Basis и Source) получившейся таблицы. Во всех ли строках команды совпадают?
4. Выполните пошаговую симуляцию выполнения программы, отслеживая на каждом шагу значения регистров.
5. Модифицируйте программу так, чтобы в конце программы в регистре t2 была сумма чисел 2 и 3, но используя минимальное количество команд. Попробуйте также минимизировать количество используемых регистров.

УСЛОВНЫЕ ОПЕРАЦИИ И ЦИКЛЫ

Для ветвлений и циклов в программе используются команды условных переходов BEQZ, BNEZ, BLEZ, BGEZ, BTLZ, BGTZ. Команды из этой группы выполняют переход на другой участок кода, если выполняется условие, соответствующее команде, если условие не выполняется, выполнение переходит на следующую команду. Пример команды:

```
beqz  rs, offset
```

Данная команда сравнивает значение в регистре rs с нулем. Если rs равен 0, выполняется переход.

В машинном коде переход выполняется по адресу команды в памяти. В данной группе команд адрес перехода указывается относительно адреса текущей команды. На языке ассемблера для упрощения команду, на которую нужно выполнить переход, помечают некоторой меткой, а в команде перехода указывают эту метку. В простейшем случае метка должна быть текстовой:

```
bnez t0, label

.......

label: <команда, на которую нужно выполнить переход>
```

Значение команд:

beqz rs, label - переход, если rs == 0

bnez rs, label - переход, если rs != 0

blez rs, label - переход, если rs <= 0

bgez rs, label - переход, если rs >= 0

bltz rs, label - переход, если rs < 0

bgtz rs, label - переход, если rs > 0

Для организации цикла на заранее известное количество итераций достаточно загрузить в некоторый регистр количество итераций, уменьшать счетчик каждую итерацию и сравнивать с 0 оставшееся количество итераций.

Так как команд прямого сравнения двух чисел нет, для организации сравнения нужно сначала вычислить разность двух чисел при помощи команды SUB.

Задание

1. Запустите симулятор или закройте предыдущую программу (File -> Close).
2. Откройте программу day_3/arch/risc_v_lab/prog/fibonacci/main.S . Программа вычисляет числа Фибоначчи. Количество чисел за вычетом 2 указано в регистре t2.
3. Изучите программу, выполните трансляцию и симуляцию. В каких регистрах хранятся полученные на каждом шаге числа?
4. Модифицируйте программу - измените количество чисел Фибоначчи на 7.
5. Модифицируйте программу так, чтобы числа вычислялись бесконечно.

РАБОТА С ПАМЯТЬЮ