

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №6**  
**по дисциплине «Организация ЭВМ и систем»**  
**Тема: Изучение режимов адресации в RISC-V.**

Студент гр. 3382

Копасова К. А.

Преподаватель

Куршев Е. О.

Санкт-Петербург

2024

**Цель работы:** практическое освоение различных режимов адресации в ассемблерных программах RISC-V для организации работы с памятью.

**Задание:**

Требуется написать программу, которая использует разные режимы адресации для вычислений по массиву данных в памяти. Результатом выполнения вашей программы будет измененный массив в памяти.

В качестве исходных данных дается стартовый адрес в памяти для хранения массива, количество элементов в массиве и формула для требуемых вычислений (Вычисления включают изменения каждого элемента массива в зависимости от условия).

При автоматической проверке вашей программы исходные данные располагаются в регистрах следующим образом:

- a1 - адрес памяти, где расположен массив
- a2 - количество элементов в массиве

Считайте, что массив уже инициализирован и заполнен данными.

Ваша программа должна иметь следующую структуру:

```
.globl solution
solution:
    # при старте данной метки ваша программа должна выполнить
    # необходимые вычисления и изменить элементы массива согласно ветке
    условия и формуле в ней
    ret
```

Доступ к массиву (чтение, изменение) должен выполняться из памяти.

Формула для вычислений будет выведена ниже (`arr[i]` - элемент массива, считаем что `arr[-1] == 0`):

ЕСЛИ  $((arr[4] \mid arr[0] - arr[0]) > 2)$

ТО  $(arr[i] = arr[i - 1] \mid 90)$

ИНАЧЕ  $(arr[i] = arr[i] \& 47)$

Ваш seed = 4745065824

## Теоретический материал

### 1. Режимы адресации для RISC-V:

Режим адресации определяет, как процессор интерпретирует операнды команды для доступа к данным. В RISC-V основные режимы адресации включают:

#### 1) **Регистровая адресация:**

Операнд находится в регистре. Например, команда `add a0, a1, a2` складывает значения из регистров `a1` и `a2` и сохраняет результат в `a0`.

#### 2) **Непосредственная адресация (Immediate):**

Операнд указывается прямо в команде в виде константы. Например, `addi a0, a1, 5` добавляет к значению в регистре `a1` число 5 и сохраняет результат в `a0`.

#### 3) **Базовая адресация (Base + Offset):**

Используется для доступа к данным в памяти. Указывается базовый адрес (в регистре) и смещение (в байтах). Например, команда `lw a0, 4(a1)` загружает слово из памяти по адресу `a1 + 4`.

#### 4) **Относительная адресация (PC-relative):**

Используется для переходов и вызовов функций. Смещение прибавляется к текущему значению счётчика команд (PC). Например, `jal ra, label` передаёт управление на метку `label`.

### 2. Работа с массивами в памяти

Массивы хранятся в последовательных ячейках памяти. Для работы с элементами массива нужно:

1) **Знать адрес начала массива:** этот адрес передаётся в программе через регистр `a1`.

2) **Использовать смещения для доступа к элементам.**

3) **Чтение и запись данных:** используются команды загрузки (`lw`, `ld`) и сохранения (`sw`, `sd`). Например:

`ld t0, 16(a1)` загрузит 3-й элемент массива, если его размер — 8 байт.

`sd t0, 16(a1)` запишет данные в тот же элемент.

### 3. Операторы для условного перехода

RISC-V предоставляет набор операторов для переходов на основе сравнения значений:

1) **Безусловный переход:** Команда `j label` передаёт управление на указанную метку.

2) **Условные переходы на основе сравнения регистров:** Сравнение выполняется непосредственно в команде. Например:

`beq a0, a1, label` — переход на метку `label`, если значения в регистрах `a0` и `a1` равны.

`bne a0, a1, label` — если значения не равны.

3) **Переходы на основе величин (меньше, больше, и т.д.):**

`blt a0, a1, label` — переход, если  $a0 < a1$ .

`bge a0, a1, label` — переход, если  $a0 \geq a1$ .

Для работы с беззнаковыми числами используются команды `bltu` и `bgeu`.

### 4. Организация циклов на языке ассемблера

Циклы в ассемблере реализуются через комбинацию меток, условных и безусловных переходов. Использование меток и операторов перехода позволяет гибко управлять выполнением циклов.

## Ход работы

1. Выполним начальную настройку docker согласно инструкциям из раздела “Предварительная настройка среды и самостоятельная отладка решений”.

Проверка корректной установки docker:

```
ksenia@desktop-desktopovich:/mnt/d/ОргЭВМ/мои лаб работы$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
.
 (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Скачаем и запустим образ с помощью программы *docker pull*:

```
ksenia@desktop-desktopovich:/mnt/d/ОргЭВМ/мои лаб работы$ docker pull riscvcourse/workshop_risc-v:latest
```

Проверяем корректность скачивания образа с помощью команды *docker image ls*:

```
ksenia@desktop-desktopovich:/mnt/d/ОргЭВМ/мои лаб работы$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
welcome-to-docker	latest	c494bee906dd	About an hour ago	381MB
riscvcourse/workshop_risc-v	latest	c4cf7308ebd8	8 days ago	4.94GB
docker/welcome-to-docker	latest	eedaff45e3c7	12 months ago	29.5MB
hello-world	latest	305243c73457	19 months ago	24.4kB

Строка с *riscvcourse/workshop\_risc-v* присутствует, значит все установлено правильно.

2. Создадим программу, которая правильно выполнит задание.

Seed = 4745065824.

Условие: ЕСЛИ ((arr[4] | arr[0] - arr[0]) > 2)

ТО (arr[i] = arr[i - 1] | 90)

ИНАЧЕ (arr[i] = arr[i] & 47)

Упростим условие:  $\text{arr}[0] - \text{arr}[0] = 0$  при любых случаях. Следовательно, можем не проводить операцию ИЛИ, т. к. справа в выражении  $((\text{arr}[4] \mid \text{arr}[0] - \text{arr}[0]) > 2)$  всегда будет ноль и результат будет напрямую зависеть от  $\text{arr}[4]$ . Поэтому получаем условие: ЕСЛИ  $\text{arr}[4] > 2$ .

Создадим программу, которая меняет элементы массива в зависимости от условия.

Сначала с помощью инструкции `mv` сохраняем количество элементов массива в регистре `a5` и базовый адрес массива сохраняем в регистр `a6`.

Для каждого элемента массива проверяется условие, зависящее от  $\text{arr}[4]$ . Если условие истинное, то переходим к метке `success_cond` - обрабатываем все элементы массива как  $\text{arr}[i] = \text{arr}[i-1] \mid 90$ . Если же условие ложно, то продолжаем программу, соответствующую обработке по ветке ИНАЧЕ - преобразуем каждый элемент массива в соответствии с формулой  $\text{arr}[i] = \text{arr}[i] \& 47$ . Все элементы массива обрабатываются через цикл с условием, что счетчик оставшихся элементов  $> 0$ . После обработки всех элементов программа завершает выполнение.

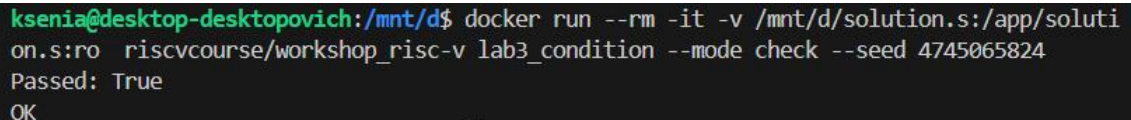
Полный код программы содержится в Приложении.

3. Получим нужный набор операций с массивом, расположенном в памяти, согласно `seed`, с помощью команды:

```
docker run --rm -it riscvcourse/workshop_risc-v lab3_condition --mode
init --seed 4745065824
```

4. Проверим работоспособность программы через локально запущенный `docker` с помощью команды:

```
docker run --rm -it -v /mnt/d/solution.s:/app/solution.s:ro
riscvcourse/workshop_risc-v lab3_condition --mode check --seed 4745065824
```



```
ksenia@desktop-desktopovich:/mnt/d$ docker run --rm -it -v /mnt/d/solution.s:/app/soluti
on.s:ro riscvcourse/workshop_risc-v lab3_condition --mode check --seed 4745065824
Passed: True
OK
```

Проверка пройдена верно, значит программа работает корректно.

## **Вывод**

В ходе лабораторной работы удалось изучить режимы адресации для RISC-V, ознакомиться с подходом по работе с массивов в памяти, операторы для условного перехода и подход к организации циклов на языке ассемблер.

## Приложение

### Файл solution.s:

```
.globl solution
solution:
    # a1 - адрес массива
    # a2 - количество элементов

    # Проверка условия: (arr[4] | arr[0] - arr[0]) > 2
    mv a5, a2 #a5 <- a2
    mv a6, a1 #a6 <- a1
loop:
    ld a4, 32(a1) #a4 <- *(a1 + 4) = arr[4]
    li t0, 2
    bgt a4, t0, success_cond #if a3 > 2 => success_cond

failed_cond:
    ld a7, 0(a6) #a7 <- arr[i]
    andi a7, a7, 47 # a7 <- arr[i] & 47
    sd a7, 0(a6) # a7 -> arr
    addi a6, a6, 8 # next element
    j final

success_cond:
    beq a6, a1, first_element
    ld a7, -8(a6) # a7 <- arr[i-1]
    j success_proc

first_element:
    li a7, 0

success_proc:
    ori a7, a7, 90 # a7 <- a7|90 = arr[i-1]|90
    sd a7, 0(a6)
    addi a6, a6, 8
    j final

final:
    addi a5, a5, -1 # a5 -= 1
    bgtz a5, loop #if a5 > 0 repeat loop

    ret                                # Завершение функции
```