Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Задание №3**

**Определение НОД всех элементов массива**

**Дисциплина**: «Низкоуровневое программирование»

Выполнил студент гр. 3530901/90002 Сергиенко Н.И.

Преподаватель Степанов Д.С.

Санкт-Петербург

2021

Содержание

[1 Описание задачи 2](#_Toc70340478)

[2 Алгоритм Евклида 2](#_Toc70340479)

[3 Программа на языке ассемблера RISC-V 2](#_Toc70340480)

[3.1 Инициализация и первая проверка 2](#_Toc70340481)

[3.2 Нахождение НОД 3](#_Toc70340482)

[3.3 Заканчиваем работу 4](#_Toc70340483)

[3.4 Данные 4](#_Toc70340484)

[3.5 Проверка 4](#_Toc70340485)

[4 Подпрограмма и тестовая программа 5](#_Toc70340486)

[4.1 GCD\_main.s 5](#_Toc70340487)

[4.2 GCD\_sub.s 7](#_Toc70340488)

[5 Вывод 8](#_Toc70340489)

# Описание задачи

Реализовать нахождение наибольшего общего делителя (НОД) для массива чисел.

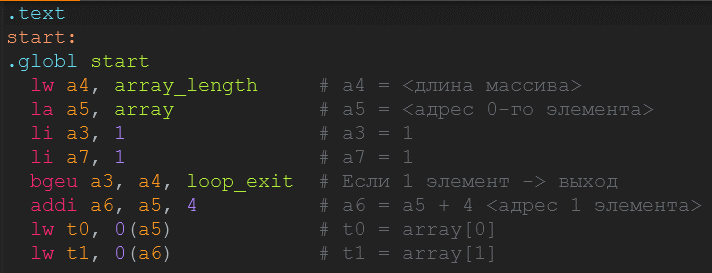
# Алгоритм Евклида

Идея алгоритма заключается в том, что мы вычитаем из большего числа меньшее и заменяем первое на их разность до тех пор, пока их разность не станет равна нулю. В таком случае уменьшаемое и вычитаемое как раз и будут искомым числом.

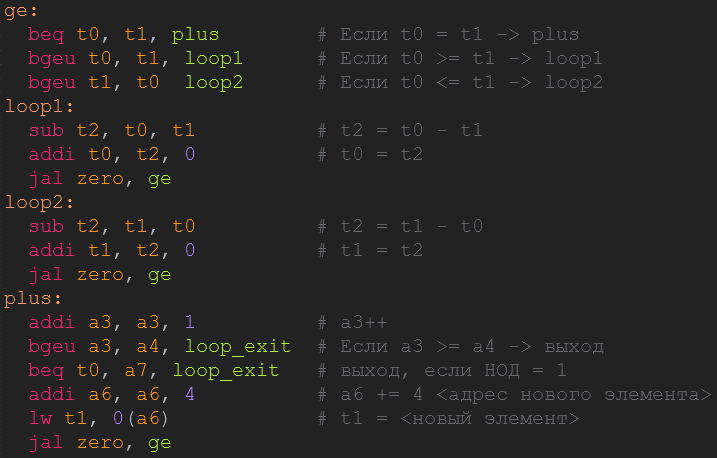
# Программа на языке ассемблера RISC-V

## Инициализация и первая проверка

* Прежде всего, нам понадобится обходить числа в цикле, а, следовательно, нужен счетчик. В качестве него мы будем использовать ячейку .
* Чтобы кол-во итераций не перескочило через длину массива, то запишем её в ячейку .
* В положим адрес 0-го элемента массива.
* В мы положим единицу, чтобы использовать её для сверки (если какое-то из чисел получится равным единице, то будем заканчивать работу).
* Если в массиве всего 1 элемент, то он и является НОДом – выходим.
* В кладем адрес 1-го элемента.
* В кладем значения 0-го и 1-го элементов.



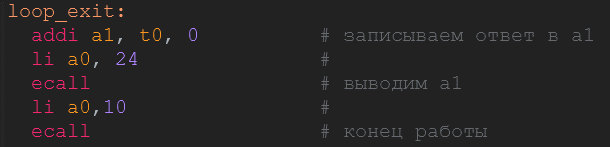
## Нахождение НОД



Рассмотрим, что тут происходит:

1. отвечает за проверки на равенство (если равны, то это НОД), а также смотрит, какое число больше и отсюда переходит в нужный цикл;
2. вычитает из так как больше, также кладет на место большего числа полученную разность, переходит в ;
3. аналогично , только ;
4. переходит на новое число для проверки; смотрит, не перескочили ли мы за длину массива; не равна ли последняя разность 1; обновляет адрес просматриваемого элемента; загружает в следующее значение из массива; переходит в

## Заканчиваем работу



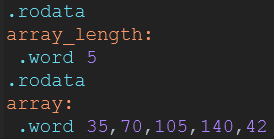
Кладём ответ в ячейку в затем через системный вызов выводим содержимое ячейки в консоль, после чего завершаем программу с кодом 0

**

Мы видим ответ «7», он является верным

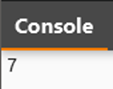
## Данные

Длина массива и элементы массива располагаются в конце программы.



## Проверка

Для проверки попросту возьмем и уменьшим количество итераций на 1, чтобы не брать последнее значение.

**

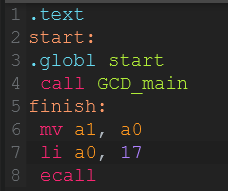


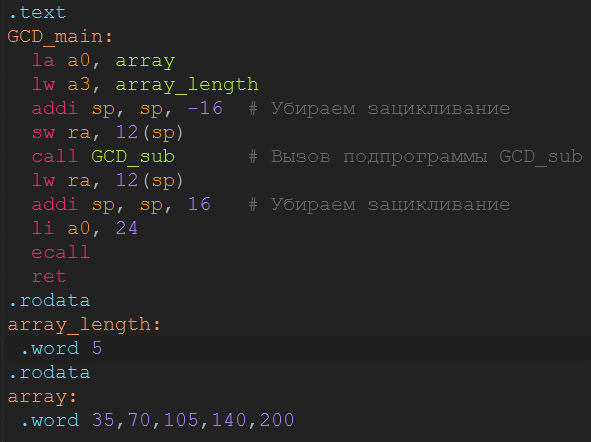
Как можно заметить, для пяти элементов {35, 70, 105, 140, 42} НОДом является число 7, но можно посчитать для первых четырех элементов и он окажется равным 35.

# Подпрограмма и тестовая программа

## GCD\_main.s

Запуск наших подпрограмм.



В данном случае мы должны запустить нашу подпрограмму, передавая ей некоторые начальные параметры, в нашем случае это адрес нулевого элемента массива и длина массива. Также мы должны сохранить значение перед вызовом подпрограммы, чтобы не произошло зацикливания, после чего мы восстанавливаем , чтобы завершился успешно.

## GCD\_sub.s

Как можно заметить, от обычной программы практически ничем и не отличается, разве что вместо чтобы вернуть данное значение в основную функцию. У нас даже ячейки памяти остались практически те же, просто теперь мы стали использовать для адреса и длины, которые получили в качестве параметров при вызове подпрограммы.

# Вывод

RISC-V гораздо удобнее и приятнее в использовании, нежели EDSAC, что позволяет выполнять более сложные задачи в более короткие сроки.