

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
2	Алгоритм Евклида	2
3	Программа на языке ассемблера RISC-V	2
3.1	Инициализация и первая проверка.....	2
3.2	Нахождение НОД	3
3.3	Заканчиваем работу	4
3.4	Данные	4
3.5	Проверка	4
4	Подпрограмма и тестовая программа.....	5
4.1	GCD_main.s	5
4.2	GCD_sub.s	7
5	Вывод	8

1 Описание задачи

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

2 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.

$$a = 35; b = 15$$

$$a) a - b = 35 - 15 = 20; a = 20$$

$$b) a - b = 20 - 15 = 5; a = 5$$

$$c) b - a = 15 - 5 = 10; b = 10$$

$$d) b - a = 10 - 5 = 5; b = 5$$

$$e) a - b = 5 - 5 = 0$$

$$GCD = 5$$

3 Программа на языке ассемблера RISC-V

3.1 Инициализация и первая проверка

- Прежде всего, нам понадобится обходить числа в цикле, а, следовательно, нужен счетчик. В качестве него мы будем использовать ячейку $a3$.
- Чтобы кол-во итераций не перескочило через длину массива, то запишем её в ячейку $a4$.
- В $a5$ положим адрес 0-го элемента массива.
- В $a7$ 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 $a6$ кладем адрес 1-го элемента.
- В $t0$ и $t1$ кладем значения 0-го и 1-го элементов.

```

.text
start:
.globl start
    lw a4, array_length      # a4 = <длина массива>
    la a5, array              # a5 = <адрес 0-го элемента>
    li a3, 1                  # a3 = 1
    li a7, 1                  # a7 = 1
    bgeu a3, a4, loop_exit    # Если 1 элемент -> выход
    addi a6, a5, 4             # a6 = a5 + 4 <адрес 1 элемента>
    lw t0, 0(a5)              # t0 = array[0]
    lw t1, 0(a6)              # t1 = array[1]

```

3.2 Нахождение НОД

```

ge:
    beq t0, t1, plus          # Если t0 = t1 -> plus
    bgeu t0, t1, loop1        # Если t0 >= t1 -> loop1
    bgeu t1, t0, loop2        # Если t0 <= t1 -> loop2
loop1:
    sub t2, t0, t1            # t2 = t0 - t1
    addi t0, t2, 0            # t0 = t2
    jal zero, ge
loop2:
    sub t2, t1, t0            # t2 = t1 - t0
    addi t1, t2, 0            # t1 = t2
    jal zero, ge
plus:
    addi a3, a3, 1            # a3++
    bgeu a3, a4, loop_exit    # Если a3 >= a4 -> выход
    beq t0, a7, loop_exit     # выход, если НОД = 1
    addi a6, a6, 4            # a6 += 4 <адрес нового элемента>
    lw t1, 0(a6)              # t1 = <новый элемент>
    jal zero, ge

```

Рассмотрим, что тут происходит:

1. *ge* отвечает за проверки на равенство (если равны, то это НОД), а также смотрит, какое число больше и отсюда переходит в нужный цикл;
2. *loop1* вычитает из *t0* *t1*, так как *t0* больше, также кладет на место большего числа полученную разность, переходит в *ge*;
3. *loop2* аналогично *loop1*, только $t1 > t0$;

4. *plus* 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 *t1* следующее значение из массива; переходит в *ge*.

3.3 Заканчиваем работу

```
loop_exit:
    addi a1, t0, 0          # записываем ответ в a1
    li a0, 24               #
    ecall                  # выводим a1
    li a0, 10               #
    ecall                   # конец работы
```

Кладём ответ в ячейку в *a1*, затем через системный вызов выводим содержимое ячейки в консоль, после чего завершаем программу с кодом 0



Console

7

Мы видим ответ «7», он является верным

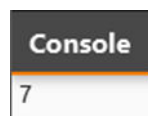
3.4 Данные

Длина массива и элементы массива располагаются в конце программы.

```
.rodata
array_length:
    .word 5
.rodata
array:
    .word 35, 70, 105, 140, 42
```

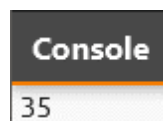
3.5 Проверка

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



Console

7



Console

35

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.

4 Подпрограмма и тестовая программа

4.1 GCD_main.s

Запуск наших подпрограмм.

```
1 .text
2 start:
3 .globl start
4 call GCD_main
5 finish:
6 mv a1, a0
7 li a0, 17
8 ecall
```

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 *ra* перед вызовом подпрограммы, чтобы не произошло заикливания, после чего мы восстанавливаем *ra*, чтобы *ret* завершился успешно.

```
.text
GCD_main:
    la a0, array
    lw a3, array_length
    addi sp, sp, -16 # Убираем заикливание
    sw ra, 12(sp)
    call GCD_sub     # Вызов подпрограммы GCD_sub
    lw ra, 12(sp)
    addi sp, sp, 16  # Убираем заикливание
    li a0, 24
    ecall
    ret
.rodata
array_length:
    .word 5
.rodata
array:
    .word 35, 70, 105, 140, 200
```

4.2 GCD_sub.s

```
.text
GCD_sub:
.globl GCD_sub
    li a2, 1           # a2 = 1
    li gp, 1           # gp = 1
    bgeu a2, a3, loop_exit # если 1 элемент -> выход
    addi a4, a0, 4      # a4 = a0 + 4 <адрес 1 элемента>
    lw t0, 0(a0)        # t0 = array[0]
    lw t1, 0(a4)        # t1 = array[1]
ge:
    beq t0, t1, plus    # если t1==t0 -> plus
    bgeu t0, t1, loop1   # если t0>=t1 -> loop1
    bgeu t1, t0, loop2   # если t0<=t1 -> loop2
loop1:
    sub t2, t0, t1      # t2 = t0 - t1
    addi t0, t2, 0      # t0 = t2
    jal zero, ge
loop2:
    sub t2, t1, t0      # t2 = t1 - t0
    addi t1, t2, 0      # t1 = t2
    jal zero, ge
plus:
    addi a2, a2, 1      # a2++
    bgeu a2, a3, loop_exit # если a2>=a3 -> выход
    beq t0, gp, loop_exit # если НОД=1 -> выход
    addi a4, a4, 4      # a4 +=4 <адрес нового элемента>
    lw t1, 0(a4)        # t1 = <новый элемент>
    jal zero, ge
loop_exit:
    addi a1, t0, 0      # a1 -> ответ
    ret
```

Как можно заметить, от обычной программы практически ничем и не отличается, разве что вместо *ecall* мы написали *ret*, чтобы вернуть данное значение в основную функцию. У нас даже ячейки памяти остались практически те же, просто теперь мы стали использовать *a0* и *a3* для адреса и длины, которые получили в качестве параметров при вызове подпрограммы.

5 Вывод

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