

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчет по лабораторной работе №3**

**Дисциплина:** Низкоуровневое программирование.

**Тема:** Программирование RISC-V.

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_ Бехтольд Ек.В.  
(подпись)

Принял

преподаватель \_\_\_\_\_ Алексюк А.О.  
(подпись)

«\_\_\_» \_\_\_\_\_ 2021 г.

Санкт-Петербург  
2021

## ОГЛАВЛЕНИЕ

|   |    |
|---|----|
| 1. Техническое задание .....                                      | 3  |
| 2. Метод решения .....  | 3  |
| 3. Код программы на языке ассемблера RISC-V.....                  | 4  |
| 4. Подпрограмма с вызывающей программой в соответствии с ABI..... | 8  |
| 5. Выводы.....  | 12 |

## 1. Техническое задание.

1.1 Разработать программу для RISC-V, реализующую циклический сдвиг массива чисел на заданное количество разрядов влево, отладить программу в симуляторе Jupiter. Массив (массивы) данных и другие параметры (преобразуемое число, длина массива, параметр статистики и пр.) располагаются в памяти по фиксированным адресам.

1.2 Выделить определенную вариантом задания функциональность в подпрограмму, организованную в соответствии с ABI, разработать использующую ее тестовую программу. Адрес обрабатываемого массива данных и другие значения передавать через параметры подпрограммы в соответствии с ABI. Тестовая программа должна состоять из инициализирующего кода, кода завершения, подпрограммы main и тестируемой подпрограммы.

## 2. Метод решения.

Пример: пусть в смежных ячейках памяти машины, начиная, например, с адреса 100, размещены следующие значения:  $0$ ,  $1 \cdot 2^{-16}$ ,  $2 \cdot 2^{-16}$ ,  $3 \cdot 2^{-16}$ ,  $4 \cdot 2^{-16}$ ,  $5 \cdot 2^{-16}$ . В двоичном представлении:

100: 0000000000000000

101: 0000000000000001

102: 0000000000000010

103: 0000000000000011

104: 0000000000000100

105: 0000000000000101

В целочисленной интерпретации, эти комбинации разрядов соответствуют числам 0, 1, 2, 3, 4, 5. После выполнения программы в тех же ячейках памяти

должны располагаться числа: 2, 3, 4, 5, 0, 1 (осуществлен циклический сдвиг на 2 разряда влево).

Для реализации данной задачи мы будем брать элемент массива из ячейки и записывать его в предыдущую ячейку, предварительно первый элемент поместив в рабочую ячейку. На место последнего элемента положим элемент хранящийся в рабочей ячейки. Эта процедура будет продолжаться  $k$ - раз, где  $k$  — количество сдвигов.

### 3. Код программы на языке ассемблера RISC-V

```
1 .text
2 __start:
3 .globl __start          #start programm
4     li a2, 0             #current shift
5     lw a3, shift         #a3 = 3 number of shifts required
6     la a4, array         #a4 = address 0 element of array
7     la a7, array         #a7 = address 0 element of array
8     lw a5, length        #a5 = 11 length of array
9     addi t3, a5, -1
10    slli t3, t3, 2
11    add s1, a7, t3        #s1 = address of the last array element
12
13    li a6, 1              #current shift count
14    beq x0, a3, loop_exit #if shift == 0 go to loop_exit
15    lw t2, 0(a7)         #t2 = array[0]
16
17    li a0, 4
18    la a1, msg3
19    ecall
20    li a0, 11             # print new line
21    li a1, '\n'
22    ecall
23    li a0, 4
24    la a1, msg1
25    ecall
26    li a0, 1              # print to console
27    mv a1, a2
28    ecall
29    li a0, 4
30    la a1, msg2
31    ecall
32
```

Рис. 1. Листинг программы.

```

33 sub_loop:                                # shifting the array
34     lw t0, 4(a4)                          # t0 = array[i+1]
35     sw t0, 0(a4)                          # array[i] = array[i+1]
36
37     li a0, 1                              # print to console
38     mv a1, t0
39     ecall
40
41     addi a4, a4, 4                          # change address of the next element in t
42     addi a6, a6, 1                          # increment current loop iteration number
43
44     bltu a6, a5, sub_loop                  # loop if current iteration < number of i
45

```

Рис. 1.2. Листинг программы(продолжение).

```

46 main_loop:
47     li a0, 1                              #print to console
48     mv a1, t2
49     ecall
50
51     sw t2, 0(s1)                          # array[n-1] = array[0]
52
53     addi a2, a2, 1                          # a2 += 1
54     beq a2, a3, loop_exit                  # a2 == a3 go to loop_exit
55     mv a4, a7                              # a4 = a7
56     lw t2, 0(a4)                          # t1 = array[0]
57     li a6, 1                              # a6 = 1
58
59     li a0, 11                             # print new line
60     li a1, '\n'
61     ecall
62     li a0, 4
63     la a1, msg1
64     ecall
65     li a0, 1                              # print to console
66     mv a1, a2
67     ecall
68     li a0, 4
69     la a1, msg2
70     ecall
71
72     jal zero, sub_loop                     # jump to sub_loop
73

```

Рис. 1.3. Листинг программы(продолжение).

```

73
74 loop_exit:
75     li a0, 10
76     ecall
77
78 .rodata                                # directive read only memory
79 length:
80     .word 6
81 shift:
82     .word 4
83 msg1: .string "Shift "
84 msg2: .string ": "
85 msg3: .string "Array: 123456 "
86
87 .data
88 array:
89     .word 1, 2, 3, 4, 5, 6

```

Рис. 1.4. Листинг программы(продолжение).

Сперва идет обозначение директивы `.text`, которая обозначает, что последующие инструкции будут размещаться в секции кода “text”, то есть начиная с адреса `0x00010008`. Метка `start` является меткой начала программы.

Дальше следует директива `globl`, которая указывает, что последующая последовательность символов является экспортируемой.

- С помощью псевдоинструкции `lw` (записать данные из одного адреса памяти в другой) записываем в регистр `a5` длину массива.
- Вызывая инструкцию `la` (записать адрес одной ячейки в другой адрес памяти), мы записываем в `a4` адрес 0 по счету элемента массива.
- `a2` - текущее количество сдвигов.
- `a3` – необходимое количество сдвигов.
- `a4` – адрес нулевого элемента массива.
- `a7` - адрес нулевого элемента массива для восстановления адреса в регистре `a4`.
- `s1` - адрес последнего элемента массива.

- `ab` – текущее сдвинутое количество элементов, отсчет начинается с 1 по причине того, что за цикл должны будут сдвинуться на 1 разряд вправо все элементы, кроме последнего.
- `Beq` - проверка: если указано сдвинуть массив на 0, то мы не выполняем работу, иначе выполняем первый цикл.
- Первый цикл направлен на выполнение сдвига на 1 разряд массива. Берем элемент из ячейки `i+1` и помещаем его в ячейку `i`.
- Далее прибавляется 1 к количеству сдвинутых разрядов, и проверка: если количество сдвинутых разрядов еще не равно длине массива, то заново выполняем цикл `sub_loop`, иначе в последнюю ячейку массива кладем нулевой элемент.
- После следует цикл `main_loop`, нужный для того, чтобы контролировать, необходимо ли еще раз сдвинуть массив на один элемент или же массив сдвинут на заданное количество разрядов.
- Увеличиваем на 1 количество выполненных сдвигов и сравниваем, достигло ли оно необходимого значения, если да, выходим из цикла и заканчиваем работу, в противном случае снова переходим к циклу `sub_loop`.
- В директиве `.rodata` располагаются неизменяемые данные, предназначенные только для чтения.
- В директиве `.data` располагается исходный массив, а после выполнения работы программы конечный результат.
- Результат работы программы выводится в консоль в удобочитаемом для пользователя виде.

```

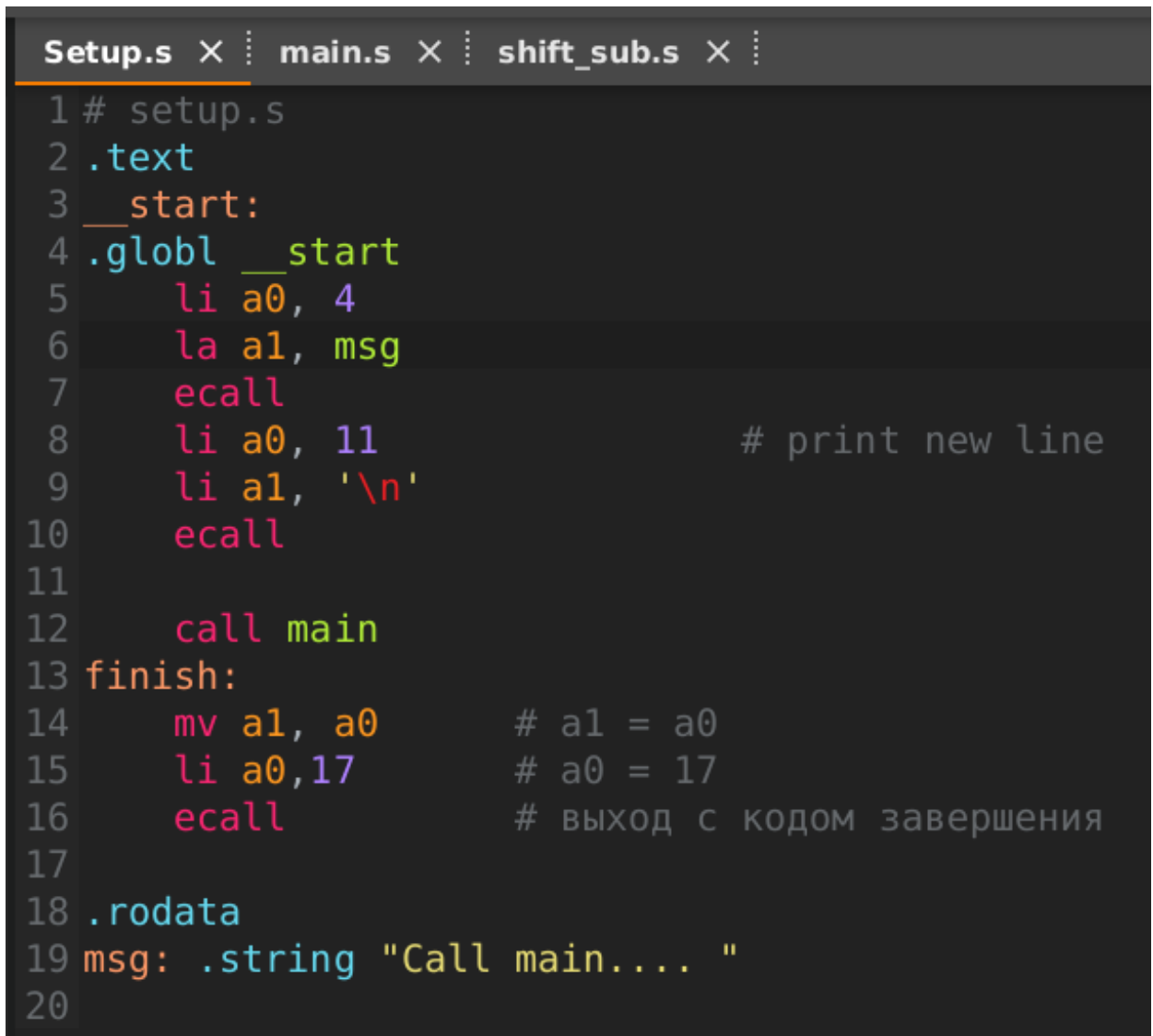
Array: 123456
Shift 0: 234561
Shift 1: 345612
Shift 2: 456123
Shift 3: 561234

```

Рис. 2. Результат работы программы.

#### 4. Подпрограмма с вызывающей программой в соответствии с ABI

Теперь нам потребуется модифицировать подпрограмму, реализовав в ней функциональность тестовой программы. В то же время, код, обеспечивающий вызов `main` и завершение работы, может использоваться «как есть» в самых разных программах. Учитывая это, мы разобьем текст программы на 3 файла: `setup.s`, `main.s` и `shift_sub.s`.



```
1 # setup.s
2 .text
3 __start:
4 .globl __start
5     li a0, 4
6     la a1, msg
7     ecall
8     li a0, 11                # print new line
9     li a1, '\n'
10    ecall
11
12    call main
13 finish:
14    mv a1, a0                # a1 = a0
15    li a0, 17                # a0 = 17
16    ecall                   # выход с кодом завершения
17
18 .rodata
19 msg: .string "Call main.... "
20
```

Рис. 3. Листинг программы `setup.s`.



```

Setup.s × | main.s × | shift_sub.s × |
1 # main.s
2 .text
3 main:
4 .globl main
5     li a0, 4                # print to console "Array: "
6     la a1, msg3
7     ecall
8     li a0, 11               # print new line
9     li a1, '\n'
10    ecall
11    li a0, 4                # print to console "Array: "
12    la a1, msg4
13    ecall
14    li a0, 11               # print new line
15    li a1, '\n'
16    ecall
17
18    lw a3, shift             # a2 = 2 number of shift
19    la a4, array             # a4 = adress 0 element of ar
20    la a7, array             # a4 = adress 0 element o
21    lw a5, array_length      # a5 = 8
22    addi t3, a5, -1
23    slli t3, t3, 2
24    add s1, a7, t3           #s1 = address of the last a
25

```

Рис.4. Листинг программы main.s.

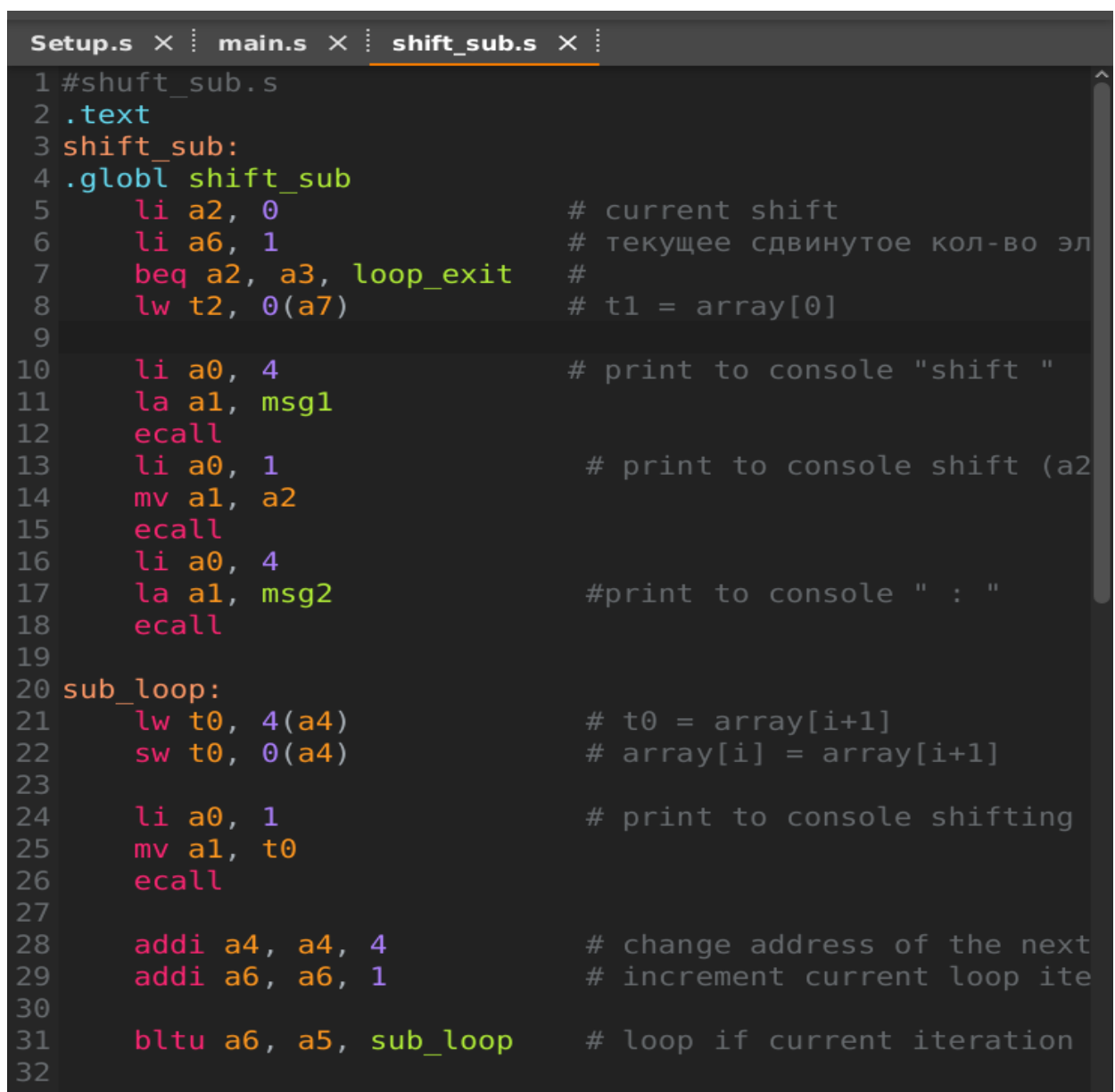
```

25
26     addi sp, sp, -16        # выделение памяти в стеке
27     sw ra, 12(sp)          # save adras for return
28     call shift_sub
29     lw ra, 12(sp)
30     addi sp, sp, 16
31     ret                     # jalr zero, ra, 0
32 .rodata
33 array_length:
34     .word 6
35 msg3: .string "Array: 123456 "
36 msg4: .string "Call shift_sub.... "
37 shift:
38     .word 2
39 .data
40 array:
41     .word 1, 2, 3, 4, 5, 6

```

Рис.4.1. Листинг программы main.s.(продолжение)

В нашем случае длина массива и адрес  $i$ -ого элемента хранятся в регистрах  $a1$ ,  $a0$  соответственно. Также до перехода на метку `shift_sub` нужно сохранить значение регистра  $ra$  (адреса, куда требуется вернуться после выполнения подпрограммы), если этого не сделать, то возвращаемый адрес будет адресом инструкции `ret` и программа просто заиклится. Чтобы этого избежать мы сохраняем значение регистра  $ra$  в стек (регистр  $sp$ ), вместимость стека устанавливаем-16 байт, в инструкции указано отрицательное число, так как стек растет вниз. После возвращения из подпрограммы мы достаем значение регистра  $ra$  и удаляем стек.



```

Setup.s x main.s x shift_sub.s x
1 #shuft_sub.s
2 .text
3 shift_sub:
4 .globl shift_sub
5     li a2, 0                # current shift
6     li a6, 1                # текущее сдвинутое кол-во эл
7     beq a2, a3, loop_exit   #
8     lw t2, 0(a7)            # t1 = array[0]
9
10    li a0, 4                 # print to console "shift "
11    la a1, msg1
12    ecall
13    li a0, 1                 # print to console shift (a2
14    mv a1, a2
15    ecall
16    li a0, 4
17    la a1, msg2              #print to console " : "
18    ecall
19
20 sub_loop:
21    lw t0, 4(a4)              # t0 = array[i+1]
22    sw t0, 0(a4)              # array[i] = array[i+1]
23
24    li a0, 1                  # print to console shifting
25    mv a1, t0
26    ecall
27
28    addi a4, a4, 4             # change address of the next
29    addi a6, a6, 1             # increment current loop ite
30
31    bltu a6, a5, sub_loop     # loop if current iteration
32

```

Рис. 5. Листинг программы `shift_sub.s`.

```

33 main_loop:
34     li a0, 1                # print to console
35     mv a1, t2
36     ecall
37
38     sw t2, 0(s1)            # array[n-1] = array[0]
39
40     addi a2, a2, 1          # a2 += 1
41     beq a2, a3, loop_exit   # a2 == a3 go to loop_exit
42     mv a4, a7               # a4 = a7
43     lw t2, 0(a4)            # t1 = array[0]
44     li a6, 1                # a6 = 1
45
46     li a0, 11               # print new line
47     li a1, '\n'
48     ecall
49     li a0, 4
50     la a1, msg1
51     ecall
52     li a0, 1                # print to console
53     mv a1, a2
54     ecall
55     li a0, 4
56     la a1, msg2
57     ecall
58
59     jal zero, sub_loop      # jump to sub_loop
60
61 loop_exit:
62     ret
63
64 .rodata
65     msg1: .string "Shift "
66     msg2: .string ": "

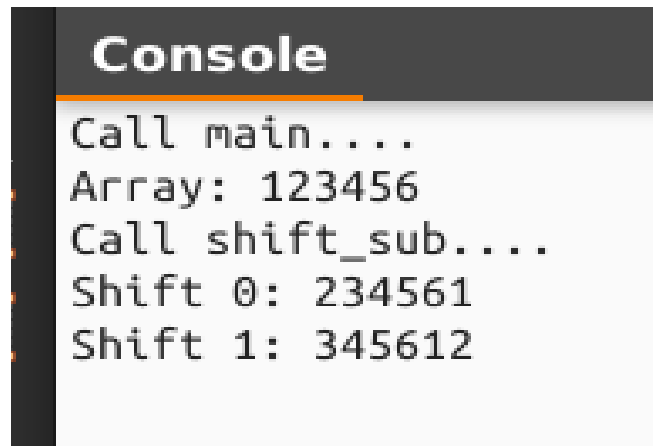
```

Рис. 5.1. Листинг программы shift\_sub.s.(продолжение)

В циклах изменения не вносились.

Результаты работы программы приведены на рис. 7. Исходный массив: 1, 2, 3, 4, 5, 6. Сдвиг выполняется на 2 разряда. Для наглядности, что каждая часть программы работает, на печать были переданы соответствующие указания, например: «Call main...» (вызывается вторая часть программы расположенная

в файле main.s) и «Call shift\_sub...» (вызов третьей части программы расположенной в файле shift\_sub.s.

A screenshot of a terminal window titled "Console". The output shows the execution of a program. It starts with "Call main....", followed by "Array: 123456". Then it shows "Call shift\_sub....", followed by "Shift 0: 234561" and "Shift 1: 345612".

```
Console
Call main....
Array: 123456
Call shift_sub....
Shift 0: 234561
Shift 1: 345612
```

Рис. 7. Результат работы подпрограммы с вызывающей программой в соответствии с ABI.

### **Выводы.**

В ходе выполнения работы были получены навыки работы с языком ассемблера RISC-V. Также была успешно решена поставленная задача циклического сдвига массива чисел на заданное количество разрядов, написана к исходной программе подпрограмма вызываемая программой в соответствии с ABI.