

Национальный исследовательский университет «Высшая школа экономики»  
Факультет компьютерных наук  
Программная инженерия

# Архитектура вычислительных систем

Отчёт по индивидуальному домашнему заданию №1  
Вариант 37

**Работу**  
**выполнила:**  
М. В. Царахова  
Группа: БПИ-213  
**Преподаватель:**  
С. А. Виденин

Москва  
2022

# Содержание

|                             |          |
|-----------------------------|----------|
| <b>Постановка задачи</b>    | <b>3</b> |
| <b>1. 4 балла</b>           | <b>4</b> |
| 1.1. Тестирование . . . . . | 4        |
| <b>2. 5 баллов</b>          | <b>5</b> |
| <b>3. 6 баллов</b>          | <b>6</b> |
| 3.1. Тестирование . . . . . | 8        |

## Постановка задачи

Разработать программу, которая получает одномерный массив  $AN$ , после чего формирует из элементов массива  $A$  новый массив  $B$  по правилам, указанным в варианте, и выводит его. Память под массивы может выделяться статически, на стеке, автоматически по выбору разработчика. При решении задачи необходимо использовать подпрограммы для реализации ввода, вывода и формирования нового массива.

Сформировать массив  $B$  из элементов массива  $A$  в следующем порядке: элементы с индексами  $i \leq \frac{N+1}{2}$  переместить на позиции с четными индексами массива  $B$  с сохранением их исходного порядка относительно друг друга, а оставшиеся элементы ( $i > \frac{N+1}{2}$ ) разместить на позициях с нечетными индексами массива  $B$  также с сохранением их исходного порядка.

## 1. 4 балла

Была получена ассемблерная программа с помощью команды

```
1 gcc -O0 -S -masm=intel -fno-asynchronous-unwind-tables task.c
```

-O0 убирает любую оптимизацию, -S останавливает компилятор после порождения ассемблерного кода, -masm=intel устанавливает синтаксиса Intel, -fno-asynchronous-unwind-tables отвечает за удаление макросов. Далее были написаны комментарии к ассемблированному коду.

С помощью следующей команды был получен исполняемый файл из программы на си

```
1 gcc task.c -o c_out
```

А с помощью этой команды был получен исполняемый файл из ассемблерной программы

```
1 gcc -c task.s -o task.o && gcc task.o -o asm_out
```

### 1.1. Тестирование

#### 1. Минимальный размер массива

Ввод: 1 1

Вывод ./c\_out: 1

Вывод ./asm\_out: 1

#### 2. Размер массива = 2

Ввод: 2 1 2

Вывод ./c\_out: 1 2

Вывод ./asm\_out: 1 2

#### 3. Размер массива = 3

Ввод: 3 1 2 3

Вывод ./c\_out: 1 3 2

Вывод ./asm\_out: 1 3 2

#### 4. Размер массива нечетный = 7

Ввод: 7 1 2 3 4 5 6 7

Вывод ./c\_out: 1 5 2 6 3 7 4

Вывод ./asm\_out: 1 5 2 6 3 7 4

#### 5. Размер массива четный = 8

Ввод: 8 1 2 3 4 5 6 7 8

Вывод ./c\_out: 1 5 2 6 3 7 4 8

Вывод ./asm\_out: 1 5 2 6 3 7 4 8

Вывод - программы идентичны, тесты полностью соответствуют запрашиваемой логике.

## 2. 5 баллов

Моя программа изначально была написана с использованием локальных переменных и была разбита на функции. Также в моей ассемблерной программе изначально было написано много комментариев, включая моменты передачи параметров в функции. Параметры передаются через регистры rdi, rsi, edx. Внутри функций эти регистры переносятся на стек, начиная с QWORD PTR -24[rbp].

```
1 input:
2     endbr64
3     push    rbp
4     mov     rbp, rsp
5     sub     rsp, 32                # пролог
6     mov     QWORD PTR -24[rbp], rdi # array = rdi
7     mov     DWORD PTR -28[rbp], esi # size = esi
8     mov     DWORD PTR -4[rbp], 0   # i = 0
9     jmp     .L2                    # переход в проверку условия цикла
```

```
1 create_new_array:
2     endbr64
3     push    rbp
4     mov     rbp, rsp                # пролог
5     mov     QWORD PTR -24[rbp], rdi # array = rdi
6     mov     QWORD PTR -32[rbp], rsi # new_array = rsi
7     mov     DWORD PTR -36[rbp], edx # size = edx
8     mov     DWORD PTR -16[rbp], 0   # i = 0
9     mov     DWORD PTR -12[rbp], 0   # j = 0
10    jmp     .L5
```

```
1 output:
2     endbr64
3     push    rbp
4     mov     rbp, rsp
5     sub     rsp, 32                # пролог
6     mov     QWORD PTR -24[rbp], rdi # array = rdi
7     mov     DWORD PTR -28[rbp], esi # size = esi
8     mov     DWORD PTR -4[rbp], 0   # i = 0
9     jmp     .L13                   # переход к проверке условия цикла
```

Сверху приведены три функции, и есть описание того что записывается из регистра на стек. Например в input из rdi, в который передают array, данные записываются в QWORD PTR -24[rbp].

Снизу приведен кусочек из .L21, в который мы переходим из функции main, в котором вызываются все описанные выше функции. Везде написаны комментарии что это именно и как оно передается в вызов через регистры.

```
1     mov     QWORD PTR -64[rbp], rax # new_array[size]
2     mov     edx, DWORD PTR -92[rbp] # edx = size
3     mov     rax, QWORD PTR -80[rbp] # rax = array
4     mov     esi, edx                # esi = size
5     mov     rdi, rax                # rdi = array
6     call    input                    # call input with rdi = array, rsi =
size
7     mov     edx, DWORD PTR -92[rbp] # edx = size
8     mov     rcx, QWORD PTR -64[rbp] # rcx = new_array
9     mov     rax, QWORD PTR -80[rbp] # rax = array
10    mov     rsi, rcx                # rsi = new_array
11    mov     rdi, rax                # rdi = array
12    call    create_new_array         # call create_new_array with rdi = array
, rsi = new_array, edx = size
```

```

13     mov     edx, DWORD PTR -92[rbp] # edx = size
14     mov     rax, QWORD PTR -64[rbp] # rax = new_array
15     mov     esi, edx               # esi = size
16     mov     rdi, rax               # rdi = new_array
17     call    output                 # call output with rdi = new_array, rsi
    = size
18     mov     eax, 0

```

### 3. 6 баллов

Для того чтобы уменьшить рефакторинг кода было принято решение заменить массивы на стеке на динамическое выделение массива.

Было:

```

1 int main() {
2     int size;
3     scanf("%d", &size);
4     int array[size];
5     int new_array[size];
6
7     input(array, size);
8     create_new_array(array, new_array, size);
9     output(new_array, size);
10    return 0;
11 }

```

Стало:

```

1 int main() {
2     int size;
3     scanf("%d", &size);
4     int *array = malloc(size * sizeof(int));
5     int *new_array = malloc(size * sizeof(int));
6
7     input(array, size);
8     create_new_array(array, new_array, size);
9     output(new_array, size);
10    free(array);
11    free(new_array);
12    return 0;
13 }

```

Была получена ассемблерная программа с помощью команды

```

1 gcc -O0 -S -masm=intel -fno-asynchronous-unwind-tables task2.c

```

Сначала наверх были вынесены все переменные

```

1 .section .rodata
2 .input:
3     .string "%d"
4 output1:
5     .string "%d "
6 output2:
7     .string "%d\n"

```

Функция input, вместо сохранения параметров на стеке были использованы регистры r13, r12, вместо локальной переменной i на стеке теперь используется rbx. Такие же манипуляции были проведены с output и create\_new\_array.

```

1     input:
2     push    rbp

```

```

3  mov     rbp, rsp
4  push    r13                # сохраняем r13, r12, rbx
5  push    r12
6  push    rbx
7  sub     rsp, 8             # выравниваем стек
8  mov     r13, rdi           # r13 = array
9  mov     r12d, esi          # r12d = size
10 mov     ebx, 0             # ebx = i = 0
11 jmp     .L2
12 .L3:
13 mov     rsi, r13
14 lea     rdi, .input[rip]
15 mov     eax, 0
16 call    __isoc99_scanf@PLT # call scanf with rdi = "%d", rsi = array + 4
17 * i
18 add     ebx, 1             # i++
19 add     r13, 4             # сдвигаем array на 4
20 .L2:
21 cmp     ebx, r12d          # if i < size -> continue
22 jl      .L3
23 add     rsp, 8
24 pop     rbx
25 pop     r12
26 pop     r13
27 pop     rbp
28 ret
.size input, .-input

```

Функция main, пушим регистры r13, r12, rbx чтобы записать туда r13 = size, r12 = array, rbx = new\_array.

Переменную size необходимо предварительно всё таки положить на стек, т.к. необходимо проводить считывание по адресу, после этого она переносится в регистр, чтобы не обращаться часто к стеку. А затем проводятся вызовы функций передавая данные через регистры.

```

1  .globl  main
2  .type   main, @function
3  main:
4  push    rbp
5  mov     rbp, rsp
6  push    r13
7  push    r12
8  push    rbx
9  sub     rsp, 24
10 mov     rax, QWORD PTR fs:40
11 mov     QWORD PTR -40[rbp], rax
12 xor     eax, eax
13
14 lea     rax, -44[rbp]
15 mov     rsi, rax           # rax = n - хранится на стеке по адресу rbp
16 - 44
17 lea     rdi, .LC0[rip]    # rdi = "%d"
18 mov     eax, 0
19 call    __isoc99_scanf@PLT # call scanf with rdi = "%d", rsi = n
20
21 mov     r13d, DWORD PTR -44[rbp] # r13d = size
22 movsx   rax, r13d
23 sal     rax, 2
24 mov     rdi, rax          # rdi = size * 4
25 call    malloc@PLT       # call malloc with rdi = size * 4
26 mov     r12, rax          # r12 = array = result of malloc

```

```

27  movsx    rax, r13d
28  sal      rax, 2
29  mov      rdi, rax
30  call     malloc@PLT          # call malloc with rdi = size * 4
31  mov      rbx, rax           # rbx = array = result of malloc
32
33  mov      esi, r13d
34  mov      rdi, r12
35  call     input              # call input with rdi = array, esi = size
36
37  mov      edx, r13d
38  mov      rsi, rbx
39  mov      rdi, r12
40  call     create_new_array    # call input with rdi = array, rsi =
new_array, edx = size
41
42  mov      esi, r13d
43  mov      rdi, rbx
44  call     output              # call output with rdi = new_array, esi =
size
45
46  mov      rdi, r12
47  call     free@PLT           # free array
48
49  mov      rdi, rbx
50  call     free@PLT           # free new_array
51
52  mov      eax, 0
53  mov      rcx, QWORD PTR -40[rbp]
54  xor      rcx, QWORD PTR fs:40
55  je       .L17
56  call     __stack_chk_fail@PLT
57 .L17:
58  add      rsp, 24
59  pop      rbx
60  pop      r12
61  pop      r13
62  pop      rbp
63  ret
64  .size    main, .-main

```

### 3.1. Тестирование

#### 1. Минимальный размер массива

Ввод: 1 1

Вывод ./c\_out: 1

Вывод ./asm\_out: 1

#### 2. Размер массива = 2

Ввод: 2 1 2

Вывод ./c\_out: 1 2

Вывод ./asm\_out: 1 2

#### 3. Размер массива = 3

Ввод: 3 1 2 3



Вывод ./c\_out: 1 3 2

Вывод ./asm\_out: 1 3 2

4. Размер массива нечетный = 7

Ввод: 7 1 2 3 4 5 6 7

Вывод ./c\_out: 1 5 2 6 3 7 4

Вывод ./asm\_out: 1 5 2 6 3 7 4

5. Размер массива четный = 8

Ввод: 8 1 2 3 4 5 6 7 8

Вывод ./c\_out: 1 5 2 6 3 7 4 8

Вывод ./asm\_out: 1 5 2 6 3 7 4 8

Все тесты совпадают результатом с предыдущими программами, значит наш рефакторинг был корректным.