

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ №5

по дисциплине «Системное программирование»

«Комбинирование разноязыковых модулей»

Вариант 24

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 20 февраля 2023 г.

Преподаватель:

Морозов В. И.

«___» _____ 2023 г.

Москва, 2023

Содержание

1	Цель работы	3
2	Ход работы	3
2.1	Описание алгоритма	3
2.2	Компилятор gcc	3
2.3	Компилятор MSVC	6
3	Выводы о проделанной работе	10
	Список использованных источников	11
	Приложение А. Код main.c	12
	Приложение Б. Код atnt.s	13
	Приложение В. Код intel.s	15

1 Цель работы

Написать программу, в которой создается одномерный числовой массив. После заполнения значениями (случайными числами) массива в нем нужно выполнить циклическую перестановку элементов. Количество позиций для циклической перестановки вводится пользователем с клавиатуры. Собрать программу с помощью компилятора Visual Studio и синтаксисом Intel.

2 Ход работы

2.1 Описание алгоритма

Программа должна осуществить циклический сдвиг массива из n элементов на $shift$ позиций вправо. $0 < shift < n$, $n > 1$.

Сдвиг состоит из трёх этапов:

1. Сначала в стек складываются последние $shift$ элементов;
2. Затем первые $n - shift$ сдвигаются на $shift$ вправо;
3. В конце первые $shift$ элементов заполняются из стека.

Алгоритм сдвига реализован в ассемблерной вставке, создание и заполнение массива, а также ввод пользовательских данных — на языке C [1].

2.2 Компилятор gcc

Код файла на C приведён в приложении А. Компилятор gcc использует транслятор gas, который использует синтаксис AT&T [2]. Приведём ассемблерную вставку в листинге 1.

```
1 .text
2 .global f
3 .type f, @function
4
5 f:
6     pushq %rdi    # array
7     pushq %rsi    # n
8     pushq %rdx    # shift
9     pushq %rcx
10    pushq %rbx
11
12
13    # store last (shift) items of array to stack
14
15    # init for loop
16    movq $0, %rcx
17    movl %edx, %ecx
18
19    array_to_stack:
20        # find position (number of elements - iterations left)
21        pushq %rsi
22        sub %ecx, %esi    # esi = n - items left
23        movl (%rdi, %rsi, 4), %ebx    # ebx = array[n - items left]
24        popq %rsi
25
26        pushq %rbx    # store element array[n - items left] to stack
27        loop array_to_stack
28
```

```

29
30 # move first (n - shift) elements to the end of array
31
32 # init for loop
33 pushq %rsi
34 sub %edx, %esi # esi = n - shift
35 movq $0, %rcx
36 movl %esi, %ecx
37 popq %rsi
38
39 move_elements:
40     pushq %rcx
41
42     # source element
43     sub $1, %rcx # items left - 1
44     movl (%rdi, %rcx, 4), %ebx # array[items left - 1]
45
46     # destination element
47     add %rdx, %rcx # rcx = shift + items left
48     movl %ebx, (%rdi, %rcx, 4)
49
50     popq %rcx
51     loop move_elements
52
53
54 # fill in the head of array with elements from stack
55
56 # init for loop
57 movq $0, %rcx
58 movl %edx, %ecx
59
60 return_from_stack:
61     # get element
62     popq %rbx
63
64     # find position (iterations left)
65     pushq %rcx
66     sub $1, %ecx
67     movl %ebx, (%rdi, %rcx, 4) # ebx = array[items left - 1]
68     popq %rcx
69
70     loop return_from_stack
71
72
73 popq %rbx
74 popq %rcx
75 popq %rdx # shift
76 popq %rsi # n
77 popq %rdi # array
78
79 ret

```

Листинг 1 – Ассемблерная вставка на синтаксисе AT&T

Для вызова функции f из программы на C используется следующее объявление (листинг 2):

```
extern void f(int32_t *array, int32_t n, int32_t shift);
```

Листинг 2 – Объявление функции в C

Для сборки используется команда: `gcc atnt.s main.c`

Далее продемонстрируем работу программы (рисунки 1-2).

```
alex@alex-nb ~/D/y/h/mix (master)> ./a.out
Enter array length (n > 1): 10
Enter shift (0 < shift < n): 1
Array before:
62 83 9 74 69 27 28 8 87 92
Array after:
92 62 83 9 74 69 27 28 8 87
```

Рисунок 1 – Работа программы

```
alex@alex-nb ~/D/y/h/mix (master)> ./a.out
Enter array length (n > 1): 5
Enter shift (0 < shift < n): 3
Array before:
89 52 26 82 14
Array after:
26 82 14 89 52
```

Рисунок 2 – Работа программы

Исполняемый файл также можно открыть в отладчике edb (рисунки 3).

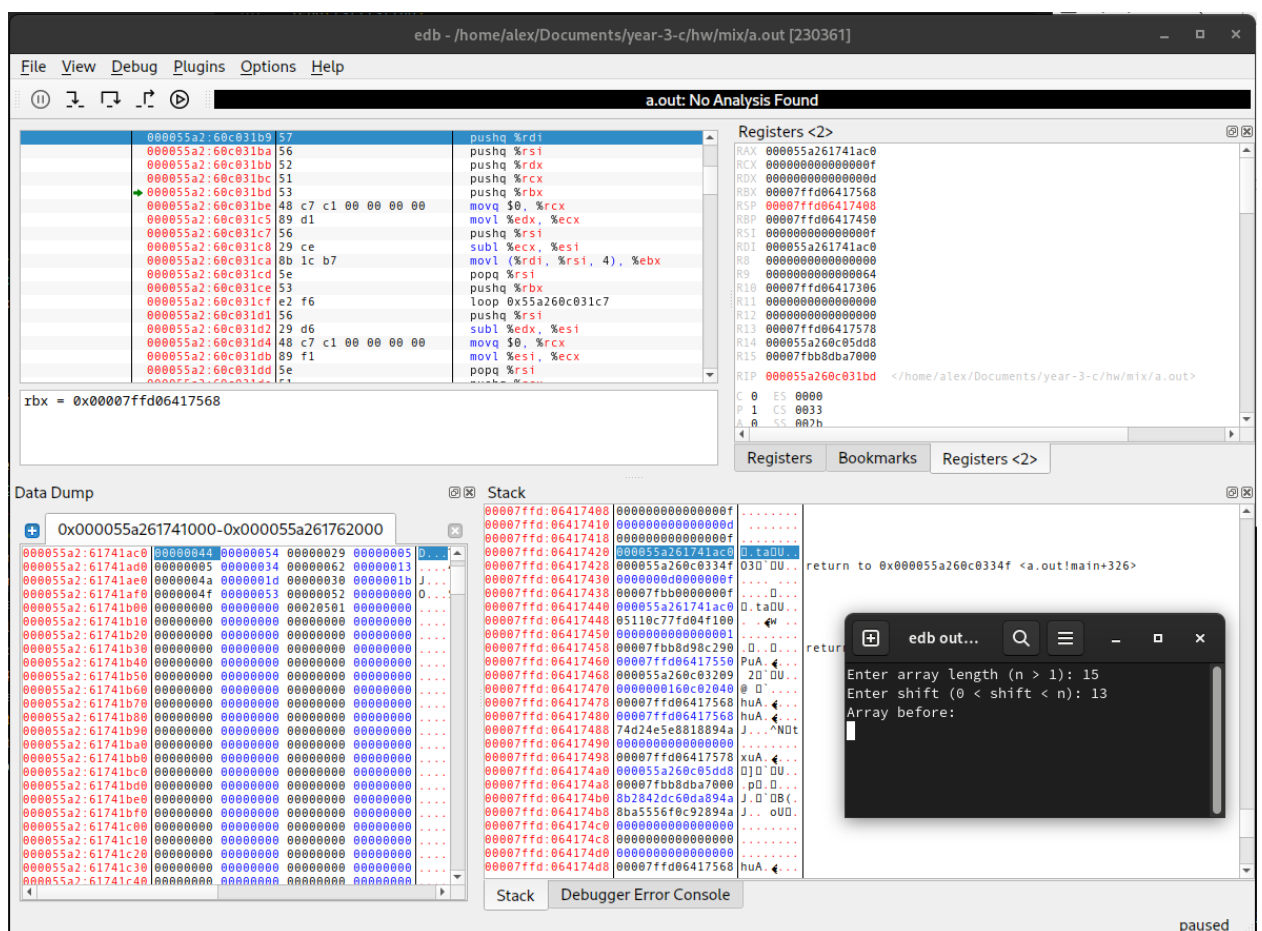


Рисунок 3 – Ассемблерная вставка в отладчике

2.3 Компилятор MSVC

Попробуем скомпилировать программу на Windows с помощью Visual Studio Build Tools. Компилятор MSVC использует транслятор `masm` для синтаксиса Intel, поэтому нам нужно переписать ассемблерную вставку: убрать префикс, поменять местами источник и приёмник, убрать типы из команд, а также учитывать регистры при передаче параметров в соответствии с соглашением о вызовах [3].

Приведём ассемблерную вставку на синтаксисе Intel в листинге 3.

```
1  .CODE
2
3  f PROC
4      push rcx    ; array
5      push rdx    ; n
6      push r8     ; shift
7      push rsi
8      push rbx
9
10     ; move arguments to adequate registers
11     push rcx
12     push rdx
13     push r8
14     pop  rdx
15     pop  rsi
16     pop  rdi
17
18
19     ; store last (shift) items of array to stack
20
21     ; init for loop
22     mov rcx, 0
23     mov ecx, edx
24
25     array_to_stack:
26         ; find position (number of elements - iterations left)
27         push rsi
28         sub esi, ecx ; esi = n - items left
29         mov ebx, [rdi + rsi * 4] ; ebx = array[n - items left]
30         pop rsi
31
32         push rbx ; store element array[n - items left] to stack
33         loop array_to_stack
34
35
36     ; move first (n - shift) elements to the end of array
37
38     ; init for loop
39     push rsi
40     sub esi, edx ; esi = n - shift
41     mov rcx, 0
42     mov ecx, esi
43     pop  rsi
44
45     move_elements:
46         push rcx
47
48         ; source element
49         sub rcx, 1 ; items left - 1
50         mov ebx, [rdi + rcx * 4] ; array[items left - 1]
51
52         ; destination element
53         add rcx, rdx ; rcx = shift + items left
54         mov [rdi + rcx * 4], ebx
55
```

```

56         pop rcx
57         loop move_elements
58
59
60         ; fill in the head of array with elements from stack
61
62         ; init for loop
63         mov rcx, 0
64         mov ecx, edx
65
66     return_from_stack:
67         ; get element
68         pop rbx
69
70         ; find position (iterations left)
71         push rcx
72         sub ecx, 1
73         mov [rdi + rcx * 4], ebx ; ebx = array[items left - 1]
74         pop rcx
75
76         loop return_from_stack
77
78
79     pop rbx
80     pop rsi
81     pop r8 ; shift
82     pop rdx ; n
83     pop rcx ; array
84
85     ret
86
87 f ENDP
88 END

```

Листинг 3 – Ассемблерная вставка на синтаксисе Intel

Для компиляции и сборки программы используем Developer Command Prompt и следующие команды [4]:

- ml /c /Cx intel.s (<https://stackoverflow.com/a/4549614>)
- cl intel.obj main.c

Сборка программы показана на рисунке 4. Работа программы — на рисунке 5.

```
x64 Native Tools Command Prompt for VS 2022
*****
** Visual Studio 2022 Developer Command Prompt v17.4.5
** Copyright (c) 2022 Microsoft Corporation
*****
[vcvarsall.bat] Environment initialized for: 'x64'

C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools>cd C:\Users\alex\Documents\mix

C:\Users\alex\Documents\mix>ml64 /c /Cx intel.s
Microsoft (R) Macro Assembler (x64) Version 14.34.31942.0
Copyright (C) Microsoft Corporation. All rights reserved.

Assembling: intel.s

C:\Users\alex\Documents\mix>cl intel.obj main.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.34.31942 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

main.c
Microsoft (R) Incremental Linker Version 14.34.31942.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:intel.exe
intel.obj
main.obj

C:\Users\alex\Documents\mix>_
```

Рисунок 4 – Сборка программы

```
x64 Native Tools Command Prompt for VS 2022

C:\Users\alex\Documents\mix>intel.exe
Enter array length (n > 1): 8
Enter shift (0 < shift < n): 2
Array before:
98 35 6 17 7 91 4 12
Array after:
4 12 98 35 6 17 7 91
C:\Users\alex\Documents\mix>
```

Рисунок 5 – Работа программы

Далее продемонстрируем работу программы в отладчике (рисунки 6-8). На последнем рисунке видно, что последние три элемента массива сохранены в стек.

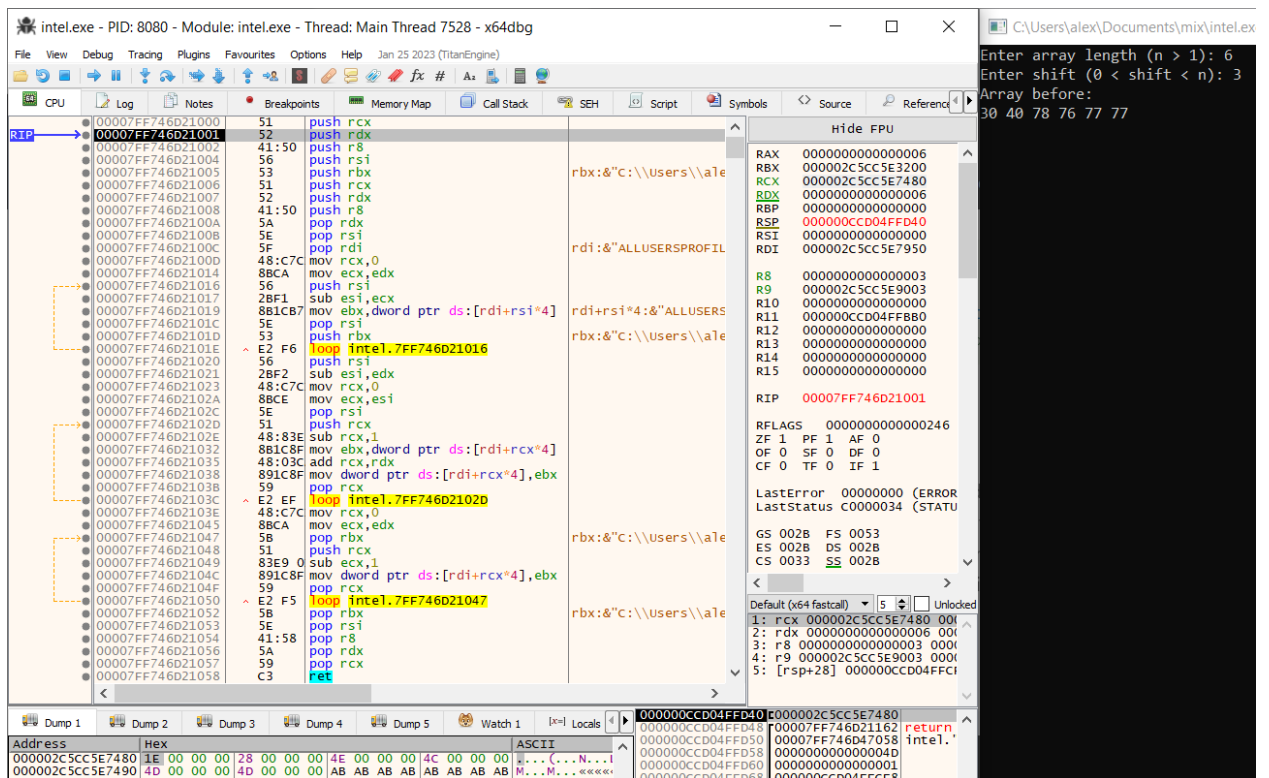


Рисунок 6 – Ассемблерная вставка в отладчике

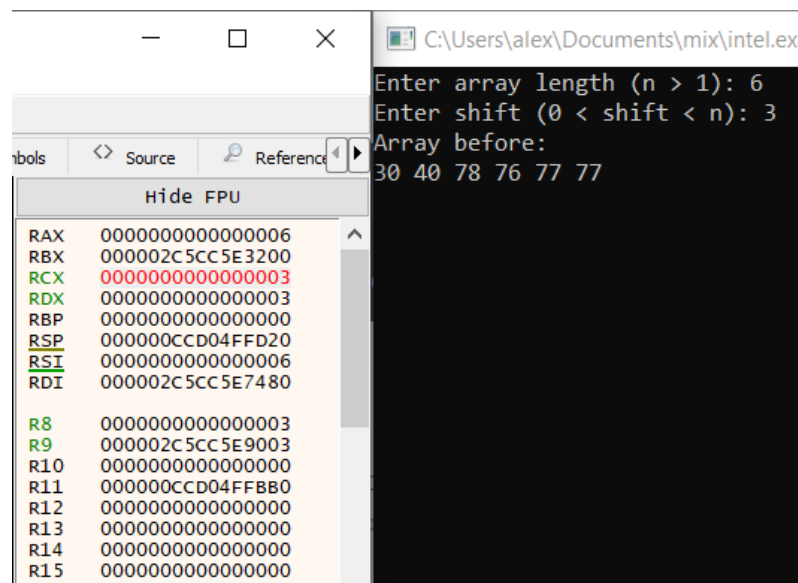


Рисунок 7 – Пользовательские данные в регистрах процессора

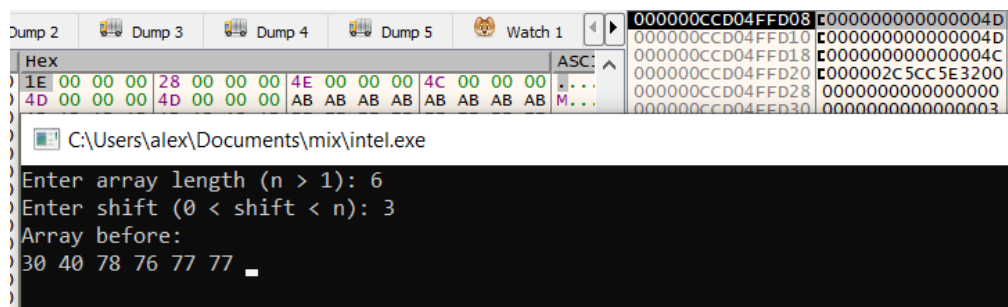


Рисунок 8 – Элементы массива в дампе.

3 Выводы о проделанной работе

В рамках данной работы я написал программу, в которой создается одномерный числовой массив. После заполнения значениями (случайными числами) массива в нем нужно выполнить циклическую перестановку элементов. Количество позиций для циклической перестановки вводится пользователем с клавиатуры. Собрал программу с помощью компилятора Visual Studio с синтаксисом Intel, а также с помощью компилятора gcc с синтаксисом AT&T.

Список использованных источников

- [1] Yale University. x86 assembly guide. <https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html>.
- [2] vivek. At&t assembly syntax. <https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm>.
- [3] Microsoft. x64 calling convention. <https://learn.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-170>.
- [4] Microsoft. Walkthrough: Compile a c program on the command line. <https://learn.microsoft.com/en-us/cpp/build/walkthrough-compile-a-c-program-on-the-command-line?view=msvc-170>.

Приложение А. Код main.c

```
1 #include <time.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdint.h>
5
6 extern void f(int32_t *array, int32_t n, int32_t shift);
7
8 int main()
9 {
10     srand(time(NULL));
11
12     // user input
13     int32_t n, shift;
14
15     printf("Enter array length (n > 1): ");
16     scanf("%d", &n);
17
18     printf("Enter shift (0 < shift < n): ");
19     scanf("%d", &shift);
20
21     // memory allocation for array
22     int32_t *array = malloc(n * sizeof(int32_t));
23
24     if (array)
25     {
26         printf("Array before: \n");
27         for (int i = 0; i < n; ++i)
28         {
29             // populate array
30             array[i] = rand() % 100;
31             printf("%d ", array[i]);
32         }
33
34         f(array, n, shift);
35
36         // print array
37         printf("\nArray after: \n");
38         for (int i = 0; i < n; ++i)
39             printf("%d ", array[i]);
40     }
41
42     free(array);
43     return 0;
44 }
```

Приложение Б. Код atnt.s

```
1 .text
2     .global f
3     .type f, @function
4
5     f:
6         pushq %rdi    # array
7         pushq %rsi    # n
8         pushq %rdx    # shift
9         pushq %rcx
10        pushq %rbx
11
12
13        # store last (shift) items of array to stack
14
15        # init for loop
16        movq $0, %rcx
17        movl %edx, %ecx
18
19        array_to_stack:
20            # find position (number of elements - iterations left)
21            pushq %rsi
22            sub %ecx, %esi    # esi = n - items left
23            movl (%rdi, %rsi, 4), %ebx    # ebx = array[n - items left]
24            popq %rsi
25
26            pushq %rbx    # store element array[n - items left] to stack
27            loop array_to_stack
28
29
30        # move first (n - shift) elements to the end of array
31
32        # init for loop
33        pushq %rsi
34        sub %edx, %esi    # esi = n - shift
35        movq $0, %rcx
36        movl %esi, %ecx
37        popq %rsi
38
39        move_elements:
40            pushq %rcx
41
42            # source element
43            sub $1, %rcx    # items left - 1
44            movl (%rdi, %rcx, 4), %ebx    # array[items left - 1]
45
46            # destination element
47            add %rdx, %rcx    # rcx = shift + items left
48            movl %ebx, (%rdi, %rcx, 4)
49
50            popq %rcx
51            loop move_elements
52
53
54        # fill in the head of array with elements from stack
55
56        # init for loop
57        movq $0, %rcx
58        movl %edx, %ecx
59
60        return_from_stack:
61            # get element
62            popq %rbx
63
64            # find position (iterations left)
65            pushq %rcx
66            sub $1, %ecx
67            movl %ebx, (%rdi, %rcx, 4)    # ebx = array[items left - 1]
```

```
68         popq %rcx
69
70         loop return_from_stack
71
72
73     popq %rbx
74     popq %rcx
75     popq %rdx # shift
76     popq %rsi # n
77     popq %rdi # array
78
79     ret
```

Приложение В. Код intel.s

```
1 .CODE
2
3 f PROC
4     push rcx    ; array
5     push rdx    ; n
6     push r8     ; shift
7     push rsi
8     push rbx
9
10    ; move arguments to adequate registers
11    push rcx
12    push rdx
13    push r8
14    pop rdx
15    pop rsi
16    pop rdi
17
18
19    ; store last (shift) items of array to stack
20
21    ; init for loop
22    mov rcx, 0
23    mov ecx, edx
24
25    array_to_stack:
26        ; find position (number of elements - iterations left)
27        push rsi
28        sub esi, ecx ; esi = n - items left
29        mov ebx, [rdi + rsi * 4] ; ebx = array[n - items left]
30        pop rsi
31
32        push rbx ; store element array[n - items left] to stack
33        loop array_to_stack
34
35
36    ; move first (n - shift) elements to the end of array
37
38    ; init for loop
39    push rsi
40    sub esi, edx ; esi = n - shift
41    mov rcx, 0
42    mov ecx, esi
43    pop rsi
44
45    move_elements:
46        push rcx
47
48        ; source element
49        sub rcx, 1 ; items left - 1
50        mov ebx, [rdi + rcx * 4] ; array[items left - 1]
51
52        ; destination element
53        add rcx, rdx ; rcx = shift + items left
54        mov [rdi + rcx * 4], ebx
55
56        pop rcx
57        loop move_elements
58
59
60    ; fill in the head of array with elements from stack
61
62    ; init for loop
63    mov rcx, 0
64    mov ecx, edx
65
66    return_from_stack:
67        ; get element
```

```

68         pop rbx
69
70         ; find position (iterations left)
71         push rcx
72         sub ecx, 1
73         mov [rdi + rcx * 4], ebx ; ebx = array[items left - 1]
74         pop rcx
75
76         loop return_from_stack
77
78
79         pop rbx
80         pop rsi
81         pop r8 ; shift
82         pop rdx ; n
83         pop rcx ; array
84
85         ret
86
87 f ENDP
88 END

```