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:

Смирнов Д. В.

«___» _____ 2023 г.

Москва, 2023

Содержание

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 A. 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).

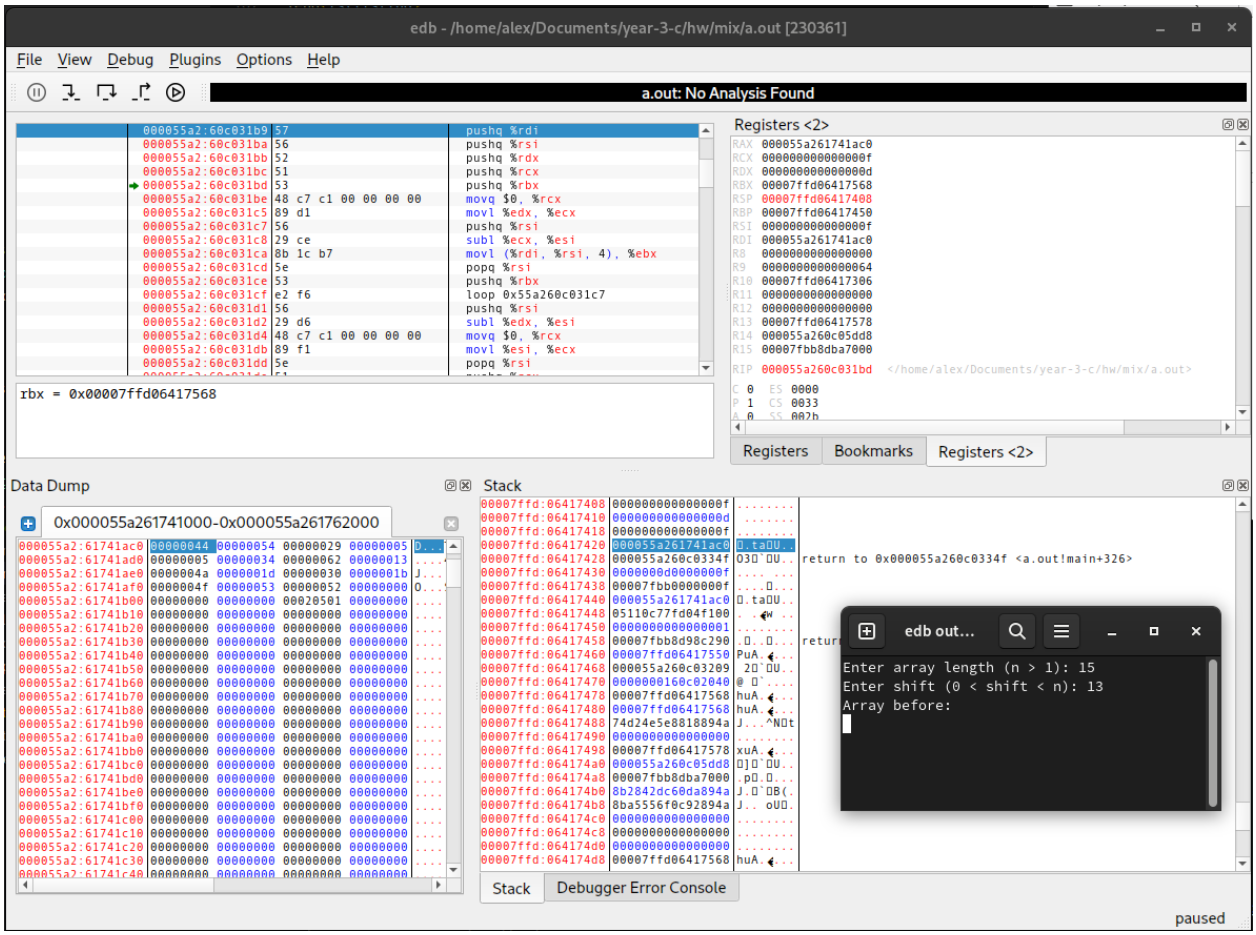

Рисунок 3 – Ассемблерная вставка в отладчике

2.3 Компилятор MSVC

Попробуем скомпилировать программу на Windows с помощью Visual Studio Build Tools. Компилятор MSVC использует транслятор masm для синтакиса 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 синтакисе Intel в листинге 3.

```
.CODE

f PROC
    push rcx  ; array
    push rdx  ; n
    push r8  ; shift
    push rsi
    push rbx

    ; move arguments to adequate registers
    push rcx
    push rdx
    push r8
    pop rdx
    pop rsi
    pop rdi


    ; store last (shift) items of array to stack

    ; init for loop
    mov rcx, 0
    mov ecx, edx

    array_to_stack:
        ; find position (number of elements - iterations left)
        push rsi
        sub esi, ecx  ; esi = n - items left
        mov ebx, [rdi + rsi * 4]  ; ebx = array[n - items left]
        pop rsi

        push rbx  ; store element array[n - items left] to stack
        loop array_to_stack


    ; move first (n - shift) elements to the end of array

    ; init for loop
    push rsi
    sub esi, edx  ; esi = n - shift
    mov rcx, 0
    mov ecx, esi
    pop rsi

    move_elements:
        push rcx

        ; source element
        sub rcx, 1  ; items left - 1
        mov ebx, [rdi + rcx * 4]  ; array[items left - 1]

        ; destination element
        add rcx, rdx  ; rcx = shift + items left
        mov [rdi + rcx * 4], ebx

```

```
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.

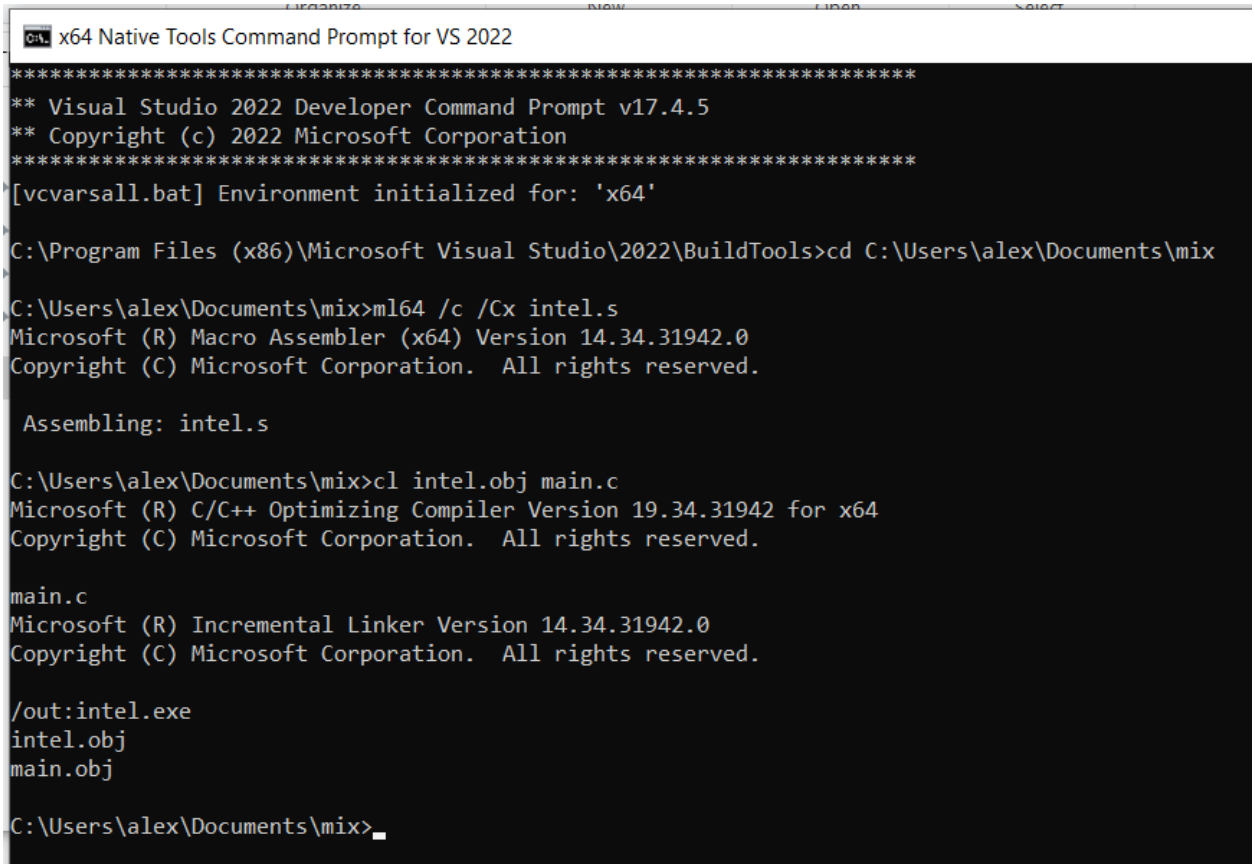

Рисунок 4 – Сборка программы

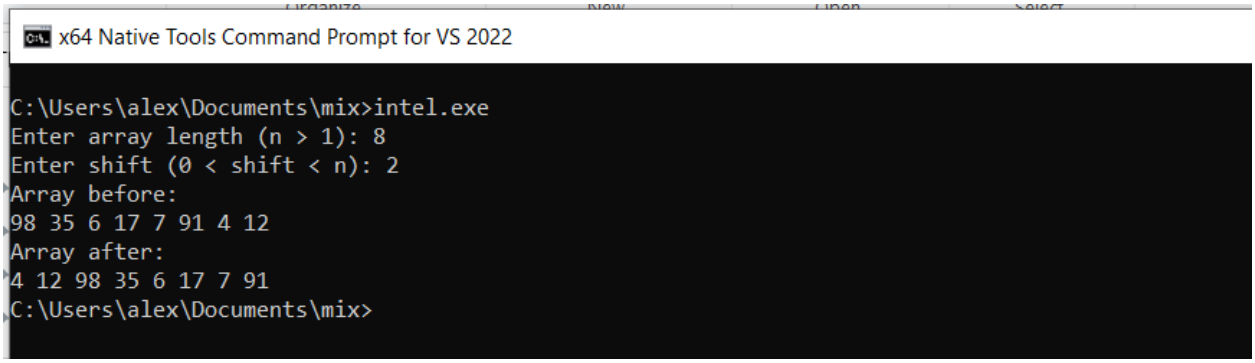

Рисунок 5 – Работа программы

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

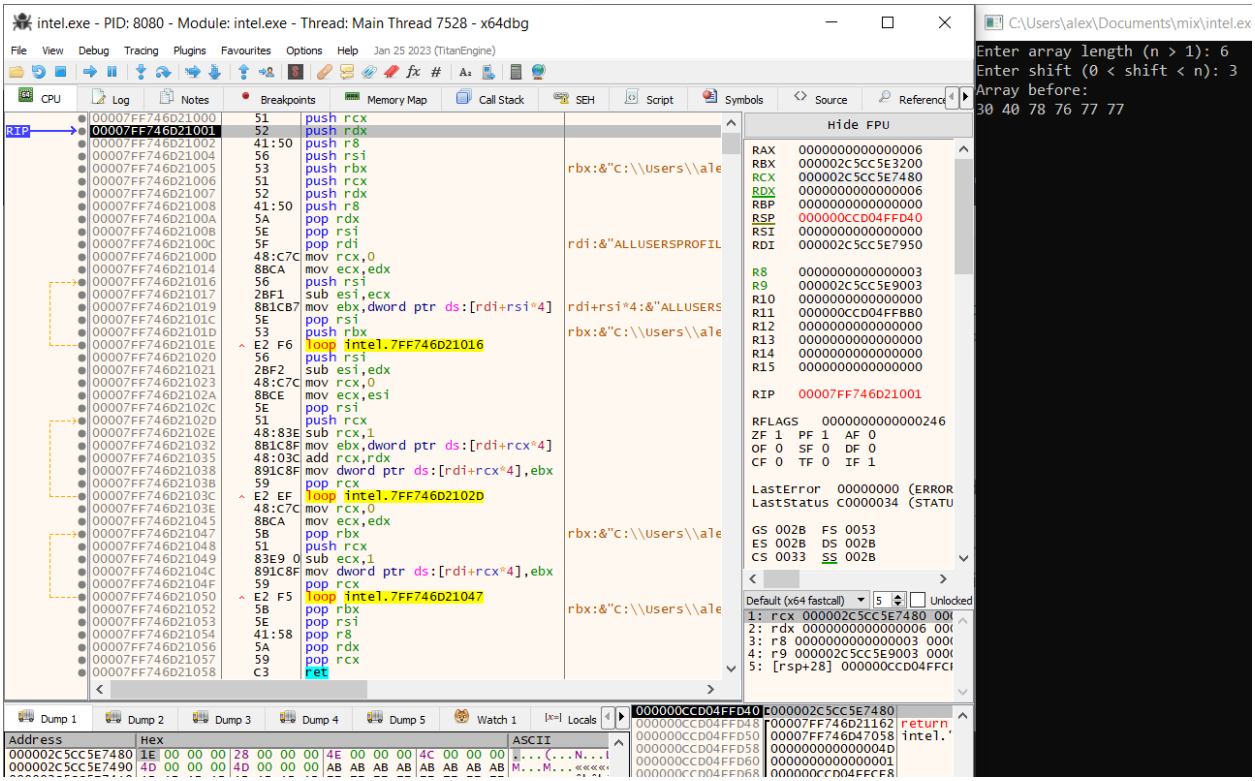

Рисунок 6 – Ассемблерная вставка в отладчике

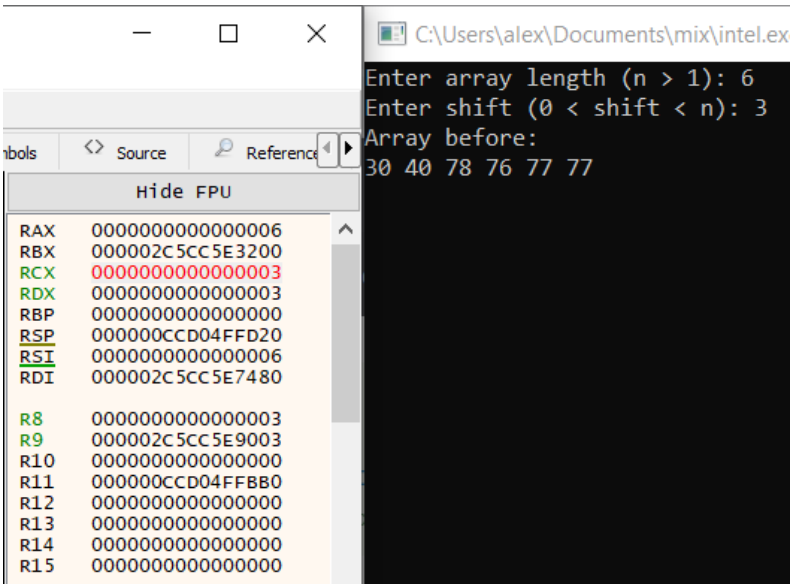

Рисунок 7 – Пользовательские данные в регистрах процессора

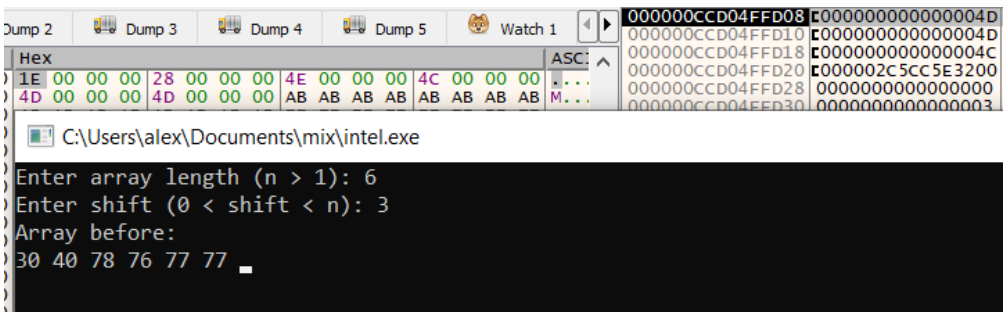

Рисунок 8 – Элементы массива в дампе.

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

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

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

int main()
{
    srand(time(NULL));

    // user input
    int32_t n, shift;

    printf("Enter array length (n > 1): ");
    scanf("%d", &n);

    printf("Enter shift (0 < shift < n): ");
    scanf("%d", &shift);

    // memory allocation for array
    int32_t *array = malloc(n * sizeof(int32_t));

    if (array)
    {
        printf("Array before: \n");
        for (int i = 0; i < n; ++i)
        {
            // populate array
            array[i] = rand() % 100;
            printf("%d ", array[i]);
        }

        f(array, n, shift);

        // print array
        printf("\nArray after: \n");
        for (int i = 0; i < n; ++i)
            printf("%d ", array[i]);
    }

    free(array);
    return 0;
}
```

Приложение Б. Код atnt.s

```
.text
    .global f
    .type f, @function

    f:
        pushq %rdi  # array
        pushq %rsi  # n
        pushq %rdx  # shift
        pushq %rcx
        pushq %rbx


        # store last (shift) items of array to stack

        # init for loop
        movq $0, %rcx
        movl %edx, %ecx

        array_to_stack:
            # find position (number of elements - iterations left)
            pushq %rsi
            sub %ecx, %esi  # esi = n - items left
            movl (%rdi, %rsi, 4), %ebx  # ebx = array[n - items left]
            popq %rsi

            pushq %rbx  # store element array[n - items left] to stack
            loop array_to_stack


        # move first (n - shift) elements to the end of array

        # init for loop
        pushq %rsi
        sub %edx, %esi  # esi = n - shift
        movq $0, %rcx
        movl %esi, %ecx
        popq %rsi

        move_elements:
            pushq %rcx

            # source element
            sub $1, %rcx  # items left - 1
            movl (%rdi, %rcx, 4), %ebx  # array[items left - 1]

            # destination element
            add %rdx, %rcx  # rcx = shift + items left
            movl %ebx, (%rdi, %rcx, 4)

            popq %rcx
            loop move_elements


        # fill in the head of array with elements from stack

        # init for loop
        movq $0, %rcx
        movl %edx, %ecx

        return_from_stack:
            # get element
            popq %rbx

            # find position (iterations left)
            pushq %rcx
            sub $1, %ecx
            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

```
.CODE

f PROC
        push rcx  ; array
        push rdx  ; n
        push r8  ; shift
        push rsi
        push rbx

        ; move arguments to adequate registers
        push rcx
        push rdx
        push r8
        pop rdx
        pop rsi
        pop rdi


        ; store last (shift) items of array to stack

        ; init for loop
        mov rcx, 0
        mov ecx, edx

        array_to_stack:
            ; find position (number of elements - iterations left)
            push rsi
            sub esi, ecx  ; esi = n - items left
            mov ebx, [rdi + rsi * 4]  ; ebx = array[n - items left]
            pop rsi

            push rbx  ; store element array[n - items left] to stack
            loop array_to_stack


        ; move first (n - shift) elements to the end of array

        ; init for loop
        push rsi
        sub esi, edx  ; esi = n - shift
        mov rcx, 0
        mov ecx, esi
        pop rsi

        move_elements:
            push rcx

            ; source element
            sub rcx, 1  ; items left - 1
            mov ebx, [rdi + rcx * 4]  ; array[items left - 1]

            ; destination element
            add rcx, rdx  ; rcx = shift + items left
            mov [rdi + rcx * 4], ebx

            pop rcx
            loop move_elements


        ; fill in the head of array with elements from stack

        ; init for loop
        mov rcx, 0
        mov ecx, edx

        return_from_stack:
            ; 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
```