Министерство образования и науки Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

Лабораторная работа № 3. Работа с массивами.
ОТЧЕТ
по лабораторной работе.

Руководитель лабораторной работы - Ронкин М.В. **Подпись**

Студент: Амбрушкевич Артем Антонович
Специальность (направление подготовки): Информационная безопасность
Группа РИ-211002

Екатеринбург 2023

Ход работы:

1. **Задание №1**. Прокомментировать пример кода работы с массивами. Файл с комментариями прикреплен к отчету и доступен по ссылке.

```
format PE CONSOLE
include 'H:\Programming\assembler\FASM\INCLUDE\win32ax.inc'

entry start

section '.data?' data readable writeable
   x1 dd 6 dup(?)       ; dup - для инициализации массива одинаковыми значениями,
эквивалентно ?,?,?,?,?,?.
   size_x1 = $-x1       ; вычисляем размер массива, $ - взять текущий адрес
   x2 dw 6 dup(?)
   size_x2 = $-x2
   A  dw ?


section '.data' data readable
   array1   dw 31,32,5,4,5,6
   size_a1 =  $ - array1

   array2   db 1,2,3,4,5,6
   size_a2 =  $ - array2

   array3   dw 7 dup(3)
   size_a3 =  $ - array3

   array4   dw 2 dup(71,10,11)
   size_a4 =  $ - array4

   size_of_dd = 4     ; размер двойного слова = 4


section '.msg' data readable
   msg_d  db ' %d ', 0Dh, 0Ah,0
   msg_s  db 0Dh, 0Ah, ' %s ', 0Dh, 0Ah,0


section '.code' code readable executable

; функция макрос для печати массива
; принимает три агрумета: arr, arr_size и word_size
macro print_array arr, arr_size, word_size
{
     mov ebx, 0   ; обнуляем регистр ebx
     @@:     ; '@@' - безымянная метка
       cinvoke printf,  ' %d ', [arr+ebx], 0
       add ebx, word_size
       cmp ebx, arr_size
       jne @b
}

start:
     ; размер массива - в байтах
     cinvoke printf,  msg_s, '1 part', 0   ; Выводим сообщение "1 part"
     cinvoke printf,  msg_d, size_a1       ; Выводим размер массива array1

     cinvoke printf,  msg_d, size_a2, 0    ; Выводим размер массива array2
     cinvoke printf,  msg_d, size_x1, 0    ; Выводим размер массива x1

     cinvoke printf,  msg_d, [array1+0], 0 ; Выводим первый элемент массива array1
     cinvoke printf,  msg_d, [array1+2], 0 ; Выводим второй элемент массива array1
     cinvoke printf,  msg_d, [array1+4], 0 ; Выводим третий элемент массива array1
                                           ; + 2 так как элементы в массиве array1 имеют
тип word(т.е. размер элемента 2 байта)
```

```
    cinvoke printf,  msg_s, '2 part', 0   ; Выводим сообщение "2 part"
    ;------------------ Выводим массив array3 -----------------------------
    ; пока ebx != size_a3 прибавлять к ebx по 2(т.к тип элементов: word)
    xor ebx, ebx        ; тоже самое что и  mov ebx,0  но быстрее
    @@:
      cinvoke printf,  ' %d ', [array3+ebx], 0  ; выводим элемент массива
      add ebx, 2
      cmp ebx, size_a3
      jne @b     ; '@b' - back, переход на предыдущую безымянную метку;
    ;----------------------------------------------------------------------
    print_array array4, size_a4, 2     ; выводим элементы массива array4 с помощью
макроса print_array

    cinvoke printf, msg_s, '3 part', 0 ; Выводим сообщение "3 part"
    mov ax, [array1]    ; в ax помещаем содержимое array1[0]
    add ax, 2           ; к ax прибавляем 2.
    mov [x2], ax        ; записываем значение из ax в x2[0]
    print_array x2, size_x2, 2   ; печатаем массив x2

    cinvoke printf,  msg_s,'4 part', 0         ; Выводим сообщение "4 part"
    print_array x1, size_x1, size_of_dd        ; печатаем массив x1

    ;-------------------- Заполняем массив x1 ---------------
    xor ebx, ebx
    @@:
      mov [x1+ebx], ebx     ; записываем значение из ebx в элемент массива x1
      add ebx, size_of_dd   ; увеличиваем ebx на 4
      cmp ebx, size_x1
      jne @b
    ;--------------------------------------------------------

    cinvoke printf, msg_s, '', 0           ; Печатаем пустую строку
    print_array x1, size_x1, size_of_dd    ; печатаем массив x1

    cinvoke printf, msg_s, '5 part', 0            ; Выводим сообщение "5 part"
    cinvoke printf, msg_s, ' Enter number', 0     ; Выводим приглашение к вводу
    cinvoke scanf, ' %d', A        ; считываем введенное число в A
    mov eax, dword [A]             ; помещаем A в eax   dword - double word
    mov [x2+2], ax                 ; записать значение ax на место второго элемента
массива x2
    print_array x2, size_x2, 2     ; печатаем массив x2



    invoke  sleep, 5000     ; 5 sec. delay

    invoke  exit, 0
    ret


section '.idata' import data readable

   library msvcrt,'MSVCRT.DLL',\
      kernel32,'KERNEL32.DLL'

   import kernel32,\
      sleep,'Sleep'

   import msvcrt,\
      puts,'puts',\
      scanf,'scanf',\
      printf,'printf',\
      lstrlen,'lstrlenA',\
      exit,'exit'
```

2. **Задание №2**. Написать программу для работы с массивами, введенными с клавиатуры. Файл с ответом прикреплен к отчету и доступен по ссылке.

```
format PE CONSOLE
include 'H:\Programming\assembler\FASM\INCLUDE\win32ax.inc'

entry start


section '.data?' data readable writeable
   arr dd 6 dup(?)
   size_arr = $-arr
   len_arr = 6
   A  dd ?


section '.msg' data readable
   msg_s  db '%s ', 0Dh, 0Ah,0


section '.code' code readable executable

; функция макрос для печати массива
; принимает три агрумета: arr, arr_size и word_size
macro print_array arr, arr_size, word_size
{
     mov ebx, 0
     @@:
       cinvoke printf,  ' %d ', [arr+ebx], 0
       add ebx, word_size
       cmp ebx, arr_size
       jne @b
}

; функция макрос для заполнения массива с клавиатуры
macro fill_array arr, size_arr, word_size
{
    mov ebx, 0
    @@:
      cinvoke scanf, ' %d', A   ; считываем введенное число в A
      mov eax, dword [A]        ; помещаем A в eax
      mov [arr+ebx], eax        ; записываем значение элемента в массив
      add ebx, word_size
      cmp ebx, size_arr
      jne @b
}

; сортировка пузырьком по возрастанию
macro sort_array arr, len_arr
{
    mov edi, arr   ; передаем в edi указатель на массив
    mov ecx, len_arr     ; передаем в ecx длину массива

    sort:
        lea ebx, [edi+ecx*4]    ; вычисляем текущий адрес второго операнда и помещаем его
в ebx
        mov eax, [edi]
    .cmploop:
        sub ebx, 4
        cmp eax, [ebx]
        jle .again
        xchg eax, [ebx]   ; xchg - меняет операнды местами
    .again:
        cmp ebx, edi
        jnz .cmploop
        stosd             ; сохраняет eax по адресу ES:(E)DI; d - double word
        loop sort
```

```
}

start:
     cinvoke printf,  msg_s, 'Enter six numbers via Enter, to fill the array: ', 0   ;
Выводим приглашение к вводу

     ; вызываем макрос для заполнения массива
     fill_array arr, size_arr, 4

     ; вывод того, что ввел пользователь
     cinvoke printf,  msg_s, 'You entered: ', 0
     print_array arr, size_arr, 4

     cinvoke printf,  msg_s, '', 0

     ; вызываем сортировку
     sort_array arr, len_arr

     ; печатаем массив после сортировки
     cinvoke printf,  msg_s, 'Array after sorting: ', 0
     print_array arr, size_arr, 4


     ; пауза и далее выход
     invoke  sleep, 5000     ; 5 sec. delay
     invoke  exit, 0
     ret




section '.idata' import data readable

   library msvcrt,'MSVCRT.DLL',\
      kernel32,'KERNEL32.DLL'

   import kernel32,\
      sleep,'Sleep'

   import msvcrt,\
      scanf,'scanf',\
      printf,'printf',\
      exit,'exit'
```

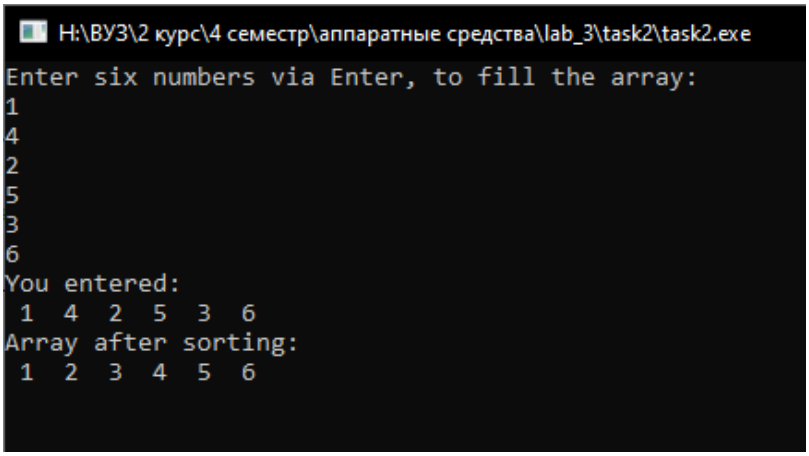

Рис. Пример работы программы

3. **Вывод**. В результате проделанной работы я познакомился с тем, как можно работать с массивами в ассемблере.

- Объявление массива в секции ‘`.data?`’ мне показалось знакомым и не сильно отличающимся от способа объявления в высокоуровневых языках программирования.
- Было интересно узнать о инструкции `dup`, которая дает возможность заполнить массив одинаковыми значениями.
- Необычным показалось мне вычисление размера массива с помощью знака `$`, что размер это совсем не количество элементов массива, а то, сколько байт занимает массив.
- В функциях макросах я использовал безымянные метки `@@` и узнал об операциях над ними: `@b` и `@f`, ссылаются на ближайшую предшествующую анонимную метку и ближайшую следующую анонимную метку соответственно.
- Очень непривычным я считаю реализацию сортировок на ассемблере, это сложно по сравнению, например, с C++. Но зато я узнал о команде `xchg`, которая меняет операнды местами и чем-то похожа на `std:swap` из C++.
- По итогу я получил программу, которая запрашивает ввод шести элементов массива, потом она их выводит на экран, сортирует, и снова выводит в консоль.