Міністерство освіти і науки, молоді та спорту України
Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

з лабораторної роботи №5 з дисципліни:
“Системного програмування”
на тему: «Програмування задач циклічної структури.
Робота з масивами.»
Варіант: № 24.

Виконав:
ст. групи КІ-203
Ширий Б. І.
Перевірила:
стар. вик. кафедри ЕОМ
Ногаль М. В.

Львів – 2023

МЕТА РОБОТИ:

Навчитися програмувати задачі циклічної структури, набути навики оголошення і опрацювання масивів.

ЗАВДАННЯ:

УМОВА ЗАВДАННЯ:

1. Створити *.exe програму, яка реалізовує задачу, задану варіантом і зберігає результат в пам'яті. Вхідні дані число A і масив чисел X вважати 32-х розрядними цілими знаковими числами. Розмір масиву X має бути не менше 10 елементів.
2. Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї.
3. Дати відповідь на контрольні запитання.

Завдання для 24 варіанту наведене нижче:

- Задані число A і масив чисел X. Визначити, яких чисел в масиві більше (1 – більших за A, 0 – менших або рівних A).

ВИКОНАННЯ:

Отож, написав текст Асемблера згідно свого варіанту, який навів у лістингу 1.

Лістинг 1. Текст програми.

```
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
.data
A dd 7
X dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ;масив
N equ ($-X) / type X ;розмір масиву
Greater db '1 - There are more numbers greater than A', 13, 10, 0
Smaller db '0 - There are more numbers smaller than A', 13, 10, 0
Equal db '0 - The number of smaller and larger numbers for A is the same',
13, 10, 0
hConsoleOutput dd 0
NumberOfCharsWritten dd 0
.code
start:
;обчислення кількості чисел більших та менших за A
mov ecx, N
mov ebx, 0 ;кількість чисел більших за A
mov edx, 0 ;кількість чисел менших за A
L:
mov eax, [X + ecx * type X - type X]
cmp eax, A
jg GreaterCount
```

```
jl SmallerCount
jmp Next
GreaterCount:
inc ebx
jmp Next
SmallerCount:
inc edx
Next:
loop L
;вивід результату
cmp ebx, edx
je EqualCount
jl MoreSmaller
push offset Greater
push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push sizeof Greater
push offset Greater
push hConsoleOutput
call WriteConsoleA
jmp EndProgram
EqualCount:
push offset Equal
push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push sizeof Equal
push offset Equal
push hConsoleOutput
call WriteConsoleA
jmp EndProgram
MoreSmaller:
push offset Smaller
push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push sizeof Smaller
push offset Smaller
push hConsoleOutput
call WriteConsoleA
EndProgram:
push 0
call ExitProcess
end start
```

ВИВІД КОНСОЛІ:

Запустивши програму, отримав такий вивід консольної програми, що зображений на рисунку 1.

Лістинг 2. Вивід консольної програми.

ВИСНОВКИ:

Під час виконання лабораторної роботи отримав навички програмування циклічних конструкцій та опрацювання масивів в асемблері. В результаті виконання лабораторної роботи студент засвоїв такі важливі концепції: використання циклічних конструкцій для повторення дій над масивами даних, оголошення масивів даних та робота з ними у пам'яті, знання основних операцій над масивами, таких як доступ до елементів масиву та зміна їх значень, здатність здійснювати обхід масиву даних та виконувати різні операції над елементами масиву.

Отже, виконання даної лабораторної роботи дозволило студенту отримати цінні навички програмування в асемблері та поглибити свої знання про масиви даних та циклічні конструкції.

КОНТРОЛЬНІ ПИТАННЯ:

ЩО РОБЛЯТЬ КОМАНДИ УМОВНИХ ПЕРЕХОДІВ?

Команди умовних переходів у Асемблері використовуються для зміни потоку виконання програми в залежності від певної умови. Ці команди дозволяють програмі пропустити або повторити деякі дії залежно від значень регістрів чи флагів процесора.

Наприклад, команда "JZ" використовується для переходу на вказану мітку, якщо прапорець "ZF" (zero flag) встановлений, що означає, що результат попередньої операції дорівнює нулю. Інші команди, такі як "JE", "JNE", "JA", "JB" та інші, використовуються для порівняння значень регістрів чи флагів процесора та здійснення переходу залежно від результату порівняння.

ЩО РОБИТЬ КОМАНДА LOOP?

Команда LOOP - це команда мови Асемблера, що використовується для здійснення циклічних дій. Команда використовується для повторення певної групи інструкцій задану кількість разів.

Команда LOOP вимагає одного операційного регістру, який зазвичай містить лічильник циклу. Після виконання команди LOOP значення лічильника циклу зменшується на одиницю. Якщо значення лічильника циклу стає рівним нулю, виконання циклу припиняється і виконування програми продовжується з наступної інструкції після циклу.

Наприклад, команда LOOP може використовуватись для повторення групи інструкцій, що виводять на екран рядок тексту деяку кількість разів, або для перебору масиву даних.

ЩО РОБИТЬ КОМАНДА LEA?

Команда LEA (Load Effective Address) використовується для завантаження в регістр адреси операнду, а не самого значення операнду. Це означає, що команда LEA завантажує в регістр адресу операнду, а не дані, що знаходяться за цією адресою.

Зазвичай команда LEA використовується для обчислення адреси пам'яті, яка потрібна для доступу до даних, що зберігаються в пам'яті. Наприклад, команда LEA може використовуватись для завантаження адреси масиву даних в регістр, щоб звертатись до окремих елементів масиву.

ЯК МОЖНА ЗВЕРНУТИСЬ ДО І-ГО ЕЛЕМЕНТА МАСИВУ?

Існує кілька способів звернутись до і-го елемента масиву в мові Асемблер, але найбільш поширеним є використання команди LEA (Load Effective Address) для завантаження адреси першого елемента масиву в регістр, а потім використовувати команду MOV (Move) для отримання адреси і-го елемента. Наприклад, якщо масив розташований в пам'яті за адресою 1000, а кожен елемент масиву займає 4 байти, то адресу і-го елемента можна отримати таким чином: LEA EAX, [1000] ; завантаження адреси першого елемента MOV EBX, [EAX + 4*i] ; отримання адреси і-го елемента.

ЯК ВИЗНАЧИТИ ЗМІЩЕННЯ ДО І-ГО ЕЛЕМЕНТА МАСИВУ?

Перш за все, потрібно знати розмір даних, що зберігаються в масиві, наприклад, 1 байт, 2 байти, 4 байти і т.д. Тоді зміщення до і-го елемента можна визначити, використовуючи формулу:

зміщення = і * розмір_елемента

Наприклад, якщо масив містить 4-байтові цілі числа, і потрібно знайти зміщення до 3-го елемента, то формула буде виглядати наступним чином:

зміщення = 3 * 4

зміщення = 12

Отже, зміщення до 3-го елемента буде 12.

Проте, слід зазначити, що в Assembler-i, зміщення може бути визначено за допомогою різних інструкцій, залежно від конкретної архітектури процесора і використовуваного синтаксису.

ЯК ЗАВАНТАЖИТИ АДРЕСУ МАСИВУ У РЕГІСТР?

Залежно від архітектури процесора та мови асемблера, синтаксис для завантаження адреси масиву до регістру може відрізнятися. Нижче наведено загальний приклад для завантаження адреси масиву у регістр на основі синтаксису x86 асемблера.

Для того, щоб завантажити адресу масиву у регістр в асемблері x86, необхідно використати команду "lea" (load effective address). Синтаксис цієї команди наступний:

lea регістр, масив

де "регістр" - це регістр, у який необхідно завантажити адресу масиву, а "масив" - це мітка, що вказує на початок масиву.

Наприклад, якщо масив має назву "myArray", а ми хочемо завантажити його адресу у регістр eax, то необхідно написати такий код:

lea eax, myArray

Після виконання цієї команди, регістр eax буде містити адресу початку масиву