МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи №7
на тему: «Опрацювання рядка символів засобами асемблера мікропроцесорів x86. Робота з файлами»
з дисципліни: «Архітектура комп'ютера»

Лектор:
доц. кафедри ПЗ
Крук О. Г.

Виконав:
ст. гр. ПЗ-22
Чаус О. М.

Прийняв:
доц. кафедри ПЗ
Крук О. Г.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Опрацювання рядка символів засобами асемблера мікропроцесорів x86. Робота з файлами.

Мета роботи: освоїти команди асемблера для роботи з рядками символів; опанувати функції Win32 для роботи з файлами; розвинути навики складання програми для опрацювання рядка символів та програми для створення, записування і читання текстового файла; відтранслювати і виконати в режимі відлагодження програми, складені відповідно до свого індивідуального завдання.

Варіант 27

27	$П_6, П_2, П_5, П_7, П_1, П_4, П_3$

Теоретичні відомості

I. Команди оброблення рядкових примітивів

В системі команд процесорів Intel передбачено п'ять груп команд для оброблення масивів байтів, слів та подвійних слів. Незважаючи на те, що всі вони називаються рядковими примітивами, область їх використання не обмежується тільки масивами рядків. З огляду на це доцільніше використовувати їх іншу назву – ланцюжкові команди.

Для адресації пам'яті в цих командах використовуються регістри ESI та EDI. Особливість цих команд полягає в тому, що обидва операнди розташовані в пам'яті. При обробленні рядкових примітивів ці команди можуть автоматично повторюватися, що робить їх застосування особливо зручним для роботи з довгими рядками та масивами.

При роботі програми в захищеному режимі адресація пам'яті в командах оброблення рядкових примітивів може здійснюватися через регістри ESI або EDI. При цьому зміщення, що міститься в регістрі ESI, відраховується відносно сегмента, чий дескриптор вказаний в регістрі DS, а зміщення, вказане в регістрі EDI, відраховується відносно сегмента, чий дескриптор вказаний в регістрі ES. При використанні лінійної моделі пам'яті в сегментних регістрах DS та ES міститься одне і те ж значення, яке в програмі неможна змінювати.

Використання префікса повторення. Самі по собі команди оброблення рядкових примітивів виконують тільки одну операцію над байтом, словом або подвійним словом пам'яті. Однак, якщо перед ними вказати префікс повторення, виконання команди буде повторено стільки разів, скільки вказано в регістрі ECX. Тобто з допомогою префікса можна виконати оброблення цілого масиву за допомогою всього однієї команди. Існує кілька типів префіксів повторення:

REP - Повторювати команду, поки ECX> 0;

REPZ, REPE - Повторювати команду, поки ECX > 0 і прапорець нуля установлений (ZF = 1);

REPNZ, REPNE - Повторювати команду, поки ECX > 0 і прапорець нуля скинутий (ZF = 0).

Прапорець напрямку DF. Стан цього прапорця впливає на те, який напрямок переміщення по рядку і як в процесі виконання команд оброблення рядкових примітивів змінюються значення регістрів ESI та EDI. Якщо прапорець DF скинутий (напрямок - прямий), вони збільшуються на розмір оброблюваного операнда (1, 2 або 4 байти), а якщо встановлений (напрямок - зворотний), то відповідно зменшуються.

Значення прапорця напрямку DF можна явно задати за допомогою команд CLD та STD:

```
CLD         ; Скидає прапорець напрямку DF (напрямок – прямий)
STD         ; Встановлює прапорець напрямку DF (напрямок - зворотний)
```

1. Команди MOVSB, MOVSW та MOVSD

Ці команди дозволяють скопіювати дані з однієї ділянки пам'яті, адреса якого вказана в регістрі ESI, в іншу ділянку пам'яті, адреса якого вказана в регістрі EDI. При цьому, залежно від стану прапорця напрямку, значення в регістрах ESI та EDI або збільшується, або зменшується.

MOVSB – копіює послідовність байтів.

MOVSW – копіює послідовність слів.

MOVSD – копіює послідовність подвійних слів.

З командами MOVSB, MOVSW і MOVSD може використовуватися префікс повторення. При цьому значення регістрів ESI та EDI будуть автоматично змінюватися залежно від стану прапорця напрямку і типу команди.

Приклад: копіювання масиву подвійних слів. Потрібно скопіювати масив, що складається з двадцяти подвійних слів зі змінної source в змінну target.

2. Команди CMPSB, CMPSW та CMPSD

Ці команди дозволяють порівняти дані з однієї ділянки пам'яті, адреса якої вказана в регістрі ESI, з іншою ділянкою пам'яті, адреса якої вказана в регістрі EDI.

CMPSB – порівнює послідовність байтів.

CMPSW – порівнює послідовність слів.

CMPSD – порівнює послідовність подвійних слів.

З командами CMPSB, CMPSW і CMPSD може використовуватися префікс повторення. При цьому значення регістрів ESI та EDI будуть автоматично змінюватися залежно від стану прапорця напрямку і типу команди.

3. Команди SCASB, SCASW і SCASD

Ці команди порівнюють значення, що міститься в регістрах AL/AX/EAX з байтом, словом або подвійним словом, які адресуються через регістр EDI.

Дана група команд зазвичай використовується для пошуку деякого значення в довгому рядку або масиві. Якщо перед командою SCAS помістити префікс REPE (або REP), рядок або масив сканувatиметься до тих пір, поки значення в регістрі ECX не стане рівним нулю, або поки не буде знайдено значення в рядку або масиві, відмінне від того, що міститься в регістрі AL/AX/EAX (тобто поки не буде скинутий прапорець нуля ZF). При використанні префікса REPNE, рядок або масив сканувatиметься до тих пір, поки значення в регістрі ECX не стане рівним нулю, або поки не буде знайдено значення в рядку або масиві, що збігається з тим, яке міститься в регістрі AL/AX/EAX (тобто поки не буде встановлено прапорець нуля ZF).

Пошук символів в рядку. У наведеному нижче фрагменті коду виконується пошук символу "F" в рядку alpha. При знаходженні даного символу, в регістрі EDI буде міститися його адреса плюс одиниця. Якщо ж шуканого символу немає в заданому рядку, то робота програми завершується в результаті переходу по команді JNZ.

4. Команди STOSB, STOSW і STOSD

Ця група команд дозволяє зберегти вміст регістра AL/AX/EAX в пам'яті, яка адресується через регістр EDI. При виконанні команди STOS вміст регістра EDI змінюється відповідно до значення прапорця напрямку DF і типу використовуваного в команді операнда. При використанні спільно з префіксом REP, за допомогою команди STOS можна записати одне і те ж значення в усі елементи масиву або рядки.

5. Команди LODSB, LODSW і LODSD

Ця група команд дозволяє завантажити в регістр AL/AX/EAX вміст байта, слова або подвійного слова пам'яті, що адресується через регістр ESI. При виконанні команди LODS вміст регістра ESI змінюється відповідно до значення прапорця напрямку DF і типом операнда, який використовується в команді. Префікс REP практично ніколи не використовується з командою LODS, оскільки при цьому буде губитися попереднє значення, завантажене в акумулятор. Таким чином, ця команда використовується для завантаження одного значення в акумулятор.
Множення елементів масиву. У наведеній нижче програмі кожен елемент масиву подвійних слів array множиться на постійне значення. Для завантаження в регістр EAX поточного елемента масиву використовується команда LODSD, а для збереження - STOSD.

II. Функції Win32 для роботи з файлами

Створення файла
Приклад: програма WriteFile.asm
Нижче наведена програма WriteFile.asm, в якій створюється новий файл, і в нього записується деякий текст. При створенні файлу використовується опція CREATE_ALWAYS, тому якщо файл з таким ім'ям вже існує, його вміст стирається.
Переміщення файлового вказівника
Функція SetFilePointer призначена для переміщення вказівника у відкритому файлі. За допомогою цієї функції можна зробити так, щоб під час записування дані додавалися в кінець файлу, а також організувати доступ до довільних ділянок даних файлу.
Параметр moveMethod визначає відправну точку, відносно якої виконується переміщення вказівника. Він може приймати одне з трьох значень: FILE_BEGIN, FILE_CURRENT та FILE_END. Власне значення, що визначає кількість байтів для переміщення вказівника, є 64-розрядним числом зі знаком, розділеним на 2 частини:
• nDistanceLo- молодші 32-біти;
• pDistanceHi - адреса змінної, яка містить старші 32-біти.
Якщо при виклику функції SetFilePointer параметр pDistanceHi дорівнює нулю, для переміщення файлового вказівника буде використовуватися тільки значення параметра nDistanceLo.

Хід роботи

1. Склав програму відповідно індивідуального завдання.
   Текст програми:

```
INCLUDE Irvine32.inc

.686
.model flat, c
.stack

.data
info BYTE "Chaus   Oleh     Mykolaiovych   01.04.2004  Zhovkva  Lvivska PZ-22   ", 0, 0
len_info DWORD 70
string2 BYTE 100 DUP (?)
len_string2 DWORD 100
string3 BYTE 70 DUP (?)
len_string3 DWORD 70
surname BYTE 22 DUP (?)
p_name BYTE 22 DUP (?)
patronymic BYTE 22 DUP (?)
date BYTE 22 DUP (?)
place BYTE 22 DUP (?)
oblast BYTE 22 DUP (?)
p_group BYTE 22 DUP (?)
len_surname DWORD 0
len_name DWORD 0
len_patronymic DWORD 0
len_date DWORD 0
len_place DWORD 0
len_oblast DWORD 0
len_group DWORD 0
space_count DWORD 0
word_begin DWORD 0
word_end DWORD 0
endline BYTE 13, 10, 0

filename BYTE "chaus.txt", 0
filehandle DWORD ?
bytes_written DWORD ?
byte_count DWORD ?

character_count DWORD 0
subjects BYTE "OOP: 88, Mathematical Analysis: 51", 0, 0
len_subjects DWORD 36
.code

jump_spaces PROC
       mov EAX, [ESP + 4] ;отримання першої позиції пропуску зі стеку
       mov EDI, OFFSET info ;отримання вказівник на стрічку
       add EDI, EAX ;зміщення вказівника на позицію першого пропуску
       mov EAX, ' '
       mov ECX, len_info
       sub ECX, [ESP + 4]
       repe scasb ;знаходження першого символа не рівного пропуску
       mov EAX, EDI
       mov EDI, OFFSET info
       sub EAX, EDI ;отримання індекс зміщення для стрічки
       dec EAX
       mov [ESP + 4], EAX ;завантаження знайденого індекса у стек
       ret
jump_spaces ENDP

jump_word PROC
       mov EAX, [ESP + 4]
       mov EDI, OFFSET info
       add EDI, EAX
       mov EAX, ' '
       mov ECX, len_info
       sub ECX, [ESP + 4]
       repne scasb ;знаходження першого символа пропуску
       mov EAX, EDI
       mov EDI, OFFSET info
```

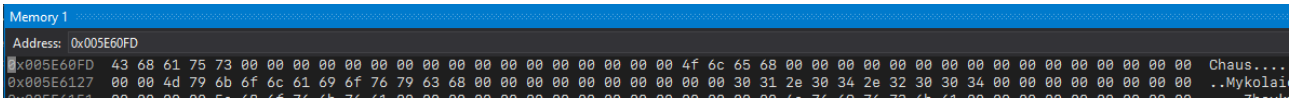
```
        sub EAX, EDI
        dec EAX
        mov [ESP + 4], EAX
        ret
jump_word ENDP
main:
        push word_begin
        call jump_word
        pop word_end ;отримання зміщення закінчення слова

        mov EDI, OFFSET surname
        mov ESI, OFFSET info
        add ESI, word_begin
        mov ECX, word_end
        sub ECX, word_begin ;отримання довжини прізвища
        mov len_surname, ECX
        rep movsb ;запис прізвища у пам'ять

        push word_end
        call jump_spaces
        pop word_begin ;отримання зміщення початку наступного слова

        push word_begin ;name
        call jump_word
        pop word_end

        mov EDI, OFFSET p_name
        mov ESI, OFFSET info
        add ESI, word_begin
        mov ECX, word_end
        sub ECX, word_begin
        mov len_name, ECX
        rep movsb

        push word_end
        call jump_spaces
        pop word_begin

        push word_begin
        call jump_word
        pop word_end

        mov EDI, OFFSET patronymic
        mov ESI, OFFSET info
        add ESI, word_begin
        mov ECX, word_end
        sub ECX, word_begin
        mov len_patronymic, ECX
        rep movsb

        push word_end
        call jump_spaces
        pop word_begin

        push word_begin
        call jump_word
        pop word_end

        mov EDI, OFFSET date
        mov ESI, OFFSET info
        add ESI, word_begin
        mov ECX, word_end
        sub ECX, word_begin
        mov len_date, ECX
        rep movsb

        push word_end
        call jump_spaces
```

```
pop word_begin

push word_begin
call jump_word
pop word_end

mov EDI, OFFSET place
mov ESI, OFFSET info
add ESI, word_begin
mov ECX, word_end
sub ECX, word_begin
mov len_place, ECX
rep movsb

push word_end
call jump_spaces
pop word_begin

push word_begin
call jump_word
pop word_end

mov EDI, OFFSET oblast
mov ESI, OFFSET info
add ESI, word_begin
mov ECX, word_end
sub ECX, word_begin
mov len_oblast, ECX
rep movsb

push word_end
call jump_spaces
pop word_begin

push word_begin
call jump_word
pop word_end

mov EDI, OFFSET p_group
mov ESI, OFFSET info
add ESI, word_begin
mov ECX, word_end
sub ECX, word_begin
mov len_group, ECX
rep movsb

push word_end
call jump_spaces
pop word_begin
;----------------------------------------
;запис інформації у порядку 6-2-5-7-1-4-3-2

mov EAX, ' '
mov EDI, OFFSET string2
mov ECX, 6
rep stosb ;записати 6 пропусків у стрічку
mov ESI, OFFSET oblast
mov ECX, len_oblast ;записати відповідне поле у стрічку
rep movsb

mov ECX, 2
rep stosb
mov ESI, OFFSET p_name
mov ECX, len_name
rep movsb

mov ECX, 5
rep stosb
```

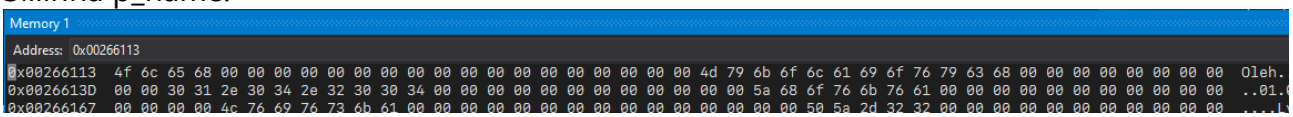
```
mov ESI, OFFSET place
mov ECX, len_place
rep movsb

mov ECX, 7
rep stosb
mov ESI, OFFSET p_group
mov ECX, len_group
rep movsb

mov ECX, 1
rep stosb
mov ESI, OFFSET surname
mov ECX, len_surname
rep movsb

mov ECX, 4
rep stosb
mov ESI, OFFSET date
mov ECX, len_date
rep movsb

mov ECX, 3
rep stosb
mov ESI, OFFSET patronymic
mov ECX, len_patronymic
rep movsb

mov ESI, OFFSET endline ;запис '\r\n\0' у кінець стрічки
mov ECX, 3
rep movsb


mov EDX, OFFSET string2
call WriteString ;консольний вивід стрічки

invoke CreateFile, ;створення нового файлу
              ADDR filename,
              GENERIC_WRITE,
              DO_NOT_SHARE,
              NULL,
              CREATE_ALWAYS,
              FILE_ATTRIBUTE_NORMAL,
              0
mov filehandle, EAX
invoke WriteFile, ;запис стрічки у файл
              filehandle,
              ADDR string2,
              len_string2,
              ADDR bytes_written,
              0
invoke WriteFile,
              filehandle,
              ADDR info,
              len_info,
              ADDR bytes_written,
              0
invoke CloseHandle, ;закриття файлу
              filehandle

invoke CreateFile, ;відкриття вже створеного файлу
              ADDR filename,
              GENERIC_READ or GENERIC_WRITE,
              DO_NOT_SHARE,
              NULL,
              OPEN_EXISTING,
              FILE_ATTRIBUTE_NORMAL,
              0
```

```
mov filehandle, EAX
invoke SetFilePointer, ;зміщення курсору файлу на початок другої стрічки
              filehandle,
              len_string2,
              0,
              0
invoke ReadFile, ;запис вмісту файлу у стрічку
              filehandle,
              ADDR string3,
              len_string3,
              ADDR byte_count,
              0
;-----------------------------------------
;підрахунок кількості символів 'C' у стрічці
mov EAX, 'C'
mov ESI, OFFSET string3
mov ECX, len_string3
loop1:
       repne scasb ;знайдення першої зустрічі символа
       jnz stop ;якщо кінець стрічки, вихід з програмиб інакше продовження циклу
       inc character_count
       jmp loop1
stop:
       invoke SetFilePointer,
                     filehandle,
                     0,
                     0,
                     FILE_END
       invoke WriteFile,
                     filehandle,
                     ADDR subjects,
                     len_subjects,
                     ADDR byte_count,
                     0
       invoke CloseHandle,
                     filehandle
       ret
END main
```

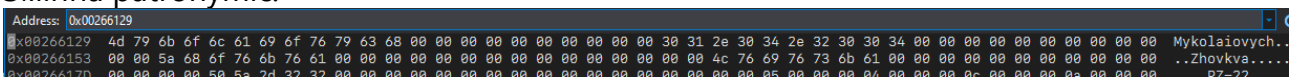
Змінна surname:



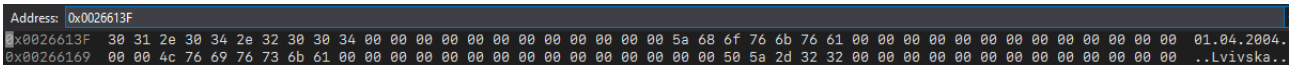
Змінна p_name:



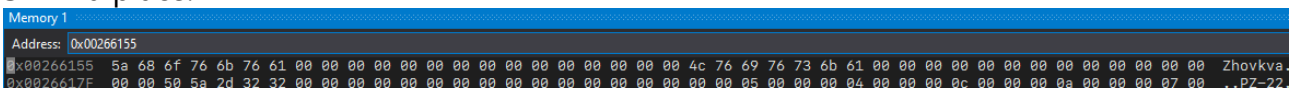
Змінна patronymic:



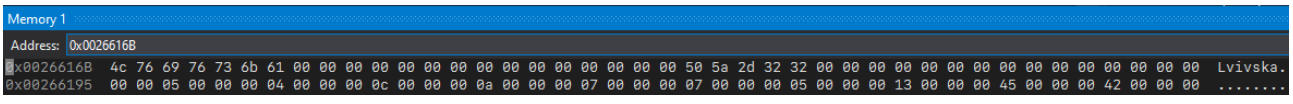
Змінна date:



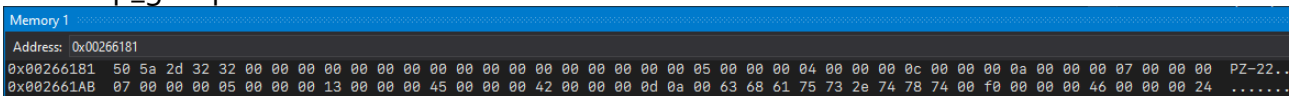
Змінна place:



Змінна oblast:



Змінна p_group:



Змінні довжин цих полів:

len_surname	5
len_name	4
len_patronymic	12
len_date	10
len_place	7
len_oblast	7
len_group	5

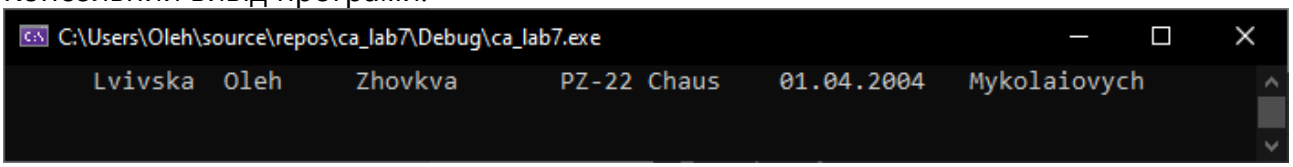
Змінна space_count:

Name	Value
space_count	19
Add item to watch	

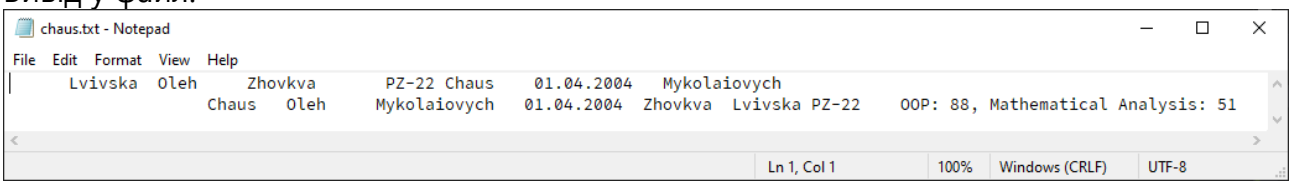
Змінна char_count:

character_count	1

Консольний вивід програми:



Вивід у файл:



Висновки: на цій лабораторній роботі я освоїв команди асемблера для роботи з рядками символів; опанував функції Win32 для роботи з файлами; розвинув навики складання програми для опрацювання рядка символів та програми для створення, записування і читання текстового файла; відтранслював і виконати в режимі відлагодження програми, складені відповідно до свого індивідуального завдання.