Лабораторна робота

Основи програмування для Windows - консоль use32

Ціль даної лабораторної роботи:

- навчитися розробляти та кодувати консольні програми в середовищі windows.

- навчиться використовувати налагоджувач win32 для верифікації правильності роботи програми,

- розробити та навчиться застосовувати прості макро засоби мови програмування win32.

Програмування для Windows

Незважаючи на те, що Windows здаються більш складними операційними системами в порівнянні з DOS, програмувати для них на асемблері набагато простіше. З одного боку, Windows-додаток запускається в 32-бітному режимі з моделлю пам'яті flat, так що програміст отримує всі переваги роботи процесора в захищеному режимі, а з іншого боку - не потрібно вивчати в деталях, як програмувати різні пристрої комп'ютера на низькому рівні .

У операційних середовищах windows користувацькі програми (програми) користуються лише системні виклики, кількість яких перевищує 2000 (близько 2200 для Windows 95 і 2434 для Windows NT). Всі Windows-додатки використовують спеціальний формат файлів, що здійснюються - формат PE (Portable Executable). Такі файли починаються як звичайні EXE-файли старого зразка (їх також називають MZ за першими двома символами заголовка), і, якщо такий файл запустити з DOS, він виконається і видасть повідомлення про помилку (текст повідомлення залежить від компілятора, що використовується), в той час як Windows помітить, що після звичайного MZ-заголовка файлу йде PE-заголовок, і запустить програму. Це зазвичай означає лише те, що для компіляції програм потрібні інші параметри командного рядка.

Рекомендується "освіжити пам'ять" – згадати режими роботи процесора Intel.

Перша програма

Як наш перший приклад подивимося, наскільки простіше написати під Windows програму, яка завантажує іншу програму. Вся процедура містить кілька викликів функції і фактично виконує системну команду START.

```
; winurl.asm
; Приклад програми для Win32.
; Запускає встановлений за промовчанням браузер на адресу, вказану в рядку
URL
```

```
; аналогічно можна запускати будь-яку програму, документ та будь-який інший
файл,
; для якого визначено операцію open
;
include shell32.inc
include kernel32.inc

.386
.model flat
.const
URL db 'http://kit.znu.edu.ua/',0
.code
_start: ; мітка точки входу повинна починатися з підкреслення
xor ebx, ebx
push ebx; для здійснюваних файлів - спосіб показу
push ebx; робочий каталог
push ebx; командний рядок
push offset URL; ім'я файлу за допомогою
push ebx; операція open або print (якщо NULL – open)
push ebx; ідентифікатор вікна, яке отримає повідомлення
call ShellExecute; ShellExecute(NULL,NULL,url,NULL,NULL,NULL)
push ebx; код виходу
call ExitProcess; ExitProcess(0)
end _start
```

Отже, у програмі виконується виклик двох системних функцій Win32 — ShellExecute() (відкрити файл) і ExitProcess() (завершити процес). Щоб викликати системну функцію Windows, програма повинна помістити у стек всі параметри від останнього до першого та передати керування далекою командою CALL. Всі ці функції самі звільняють стек (завершаючись командою RET N) та повертають результат роботи в регістрі EAX.

Така домовленість про передачу параметрів називається STDCALL. З одного боку, це дозволяє викликати функції з нефіксованою кількістю параметрів, а з іншого — сторона, що викликає, не повинна піклуватися про звільнення стека.

Крім того, функції Windows зберігають значення регістрів ЄВР, ESI, EDI та EBX, цим ми користувалися в нашому прикладі - зберігали 0 у регістрі EBX і застосували 1-байтну команду PUSH EBX замість 2-байтної PUSH 0.

Перш ніж ми зможемо скомпілювати winurl.asm, потрібно створити файли kernel32.inc і shell32.inc, в які помістимо директиви, що описують системні функції, що викликаються:

```
; kernel32.inc
; файл, що включається з визначеннями функцій з kernel32.dll
;
ifdef _TASM_
includelib import32.lib
; імена використовуваних функцій
extrn ExitProcess:near
else
includelib kernel32.lib
; справжні імена функцій, що використовуються
extrn __imp__ExitProcess@4:dword
; присвоєння для полегшення читання коду
ExitProcess equ __imp__ExitProcess@4
endif

; shell32.inc
; файл, що включається з визначеннями функцій з shell32.dll
ifdef _TASM_
```

```
includelib import32.lib
; імена використовуваних функцій
extrn ShellExecuteA:near
; присвоєння для полегшення читання коду
ShellExecute equ ShellExecuteA
else
includelib shell32.lib
; справжні імена використовуваних функцій
extrn __imp__ShellExecuteA@24:dword
; присвоєння для полегшення читання коду
ShellExecute equ __imp__ShellExecuteA@24
endif
```

Очевидно, що це робиться для спрощення написання програм із використанням різних компіляторів мови асемблера. Для наступної найпростішої програми ці дії ілюструють загальний підхід, що використовується при програмуванні Win32.

Найважливіші примітки:

Імена всіх системних функцій Win32 модифікуються так, що перед ім'ям функції ставиться підкреслення, а потім знак «@» і число байт, яке займають параметри, що передаються їй в стеку, так ExitProcess() перетворюється на _ExitProcess@4().

Компілятори з мов високого рівня часто зупиняються на цьому і викликають функції на ім'я _ExitProcess@4(), але реально викликається невелика процедура-заглушка, яка нічого не робить, а тільки передає керування на таку ж мітку, але з доданим «__imp_» — __imp__ExitProcess @4().

У всіх прикладах ми будемо звертатися безпосередньо до __imp__ExitProcess@4(). Зауважимо, що TASM (а точніше TLINK32) використовує власний спосіб виклику системних функцій, який не можна так обійти, і програми, скомпільовані за його допомогою, виявляються набагато більше і в деяких випадках працюють повільніше.

Ми відокремили описи функцій для TASM у файлах, що включаються за допомогою директив умовного асемблювання, які будуть використовувати їх, якщо в командному рядку асемблера вказати /D_TASM_.

Крім цього, всі функції, що працюють із рядками (як, наприклад, ShellExecute()), існують у двох варіантах. Якщо рядок розглядається у звичайному сенсі як набір символів ASCII, до імені функції додається "A" (ShellExecuteA()). Інший варіант функції, що використовує рядки у форматі UNICODE (два байти на символ), закінчується буквою «U». У всіх наших прикладах будемо використовувати звичайні ASCII-функції, але, якщо вам потрібно перекомпілювати програми на UNICODE, достатньо лише поміняти "A" на "U" у файлах, що включаються.

Якщо текст програми "підготовлений для MASM компілятора", а ви хочете відкомпілювати програму компілятором TASM, тоді Вам необхідно видалити з усіх API-імен суфікс @N і підключити замість USER32.LIB і KERNEL32.LIB бібліотеку IMPORT32.LIB.

Файл із бібліотекою імпорту import32.lib. Цей файл потрібен компонувальнику для дозволу зовнішніх посилань на функції Win32 API. Ви можете створити цей файл. Така потреба може виникнути, якщо вам знадобляться функції бібліотек DLL, інформація про які відсутня в існуючому варіанті файлу import32.lib. Для цього існує спеціальна утиліта

implib.exe, яка постачається у пакеті TASM 5.0. Командний рядок для її запуску має вигляд

implib имя_файла_lib список_DLL_бібліотек

Отримати інформацію про місцезнаходження конкретної функції Win32 API досить легко. Багато довідкових посібників Windows при описі конкретної функції наводять і інформацію про бібліотеку DLL, де ця функція міститься.

Кодування "швидкого початку сегмента дати та коду" .data .code для цього виду програми замінюється наступним кодом:

```
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'

.....

_DATA ends
```

```
_CODE SEGMENT DWORD PUBLIC USE32 'CODE'

.....

_CODE ends
```

Отже, тепер, коли ми маємо всі необхідні файли, можна скомпілювати першу програму для Windows.

Компіляція MASM:

```
ml/c/coff/Cp winurl.asm
link winurl.obj /subsystem:windows
```

(Тут і далі використовується 32-бітна версія link.exe)

Компіляція TASM:

```
tasm /m /ml /D_TASM_ winurl.asmtlink32 /Tpe /aa /c /x winurl.obj
```

Також для компіляції знадобляться файли kernel32.lib та shell32.lib у першому та третьому випадку та import32.lib – у другому. Ці файли входять до дистрибутивів будь-яких засобів розробки для Win32 від відповідних компаній — Microsoft, Watcom (Sybase) і Borland (Inprise), хоча їх можна відтворити з файлів kernel32.dll і shell32.dll, що у каталозі WINDOWS/SYSTEM.

Часто разом з дистрибутивами різних засобів розробки для Windows йде файл windows.inc, в якому дано макровизначення Invoke або замінена макросом команда call так, що вони приймають список аргументів, першим з яких йде ім'я функції, а потім через кому - всі параметри. З використанням цих макровизначень наша програма виглядала б так:

_start: xor ebx, ebx Invoke SnellExecute, ebx, ebx, offset URL, ebx, \ebx, ebx Invoke ExitProcess, ebx end _start

І цей текст компілюється в такий самий код, що й у нас, але виконується виклик не функції __imp__ExitProcess@4(), а проміжної функції _ExitProcess@4(). Використання цієї форми запису не дозволяє застосовувати окремі ефективні прийоми оптимізації, які ми наводитимемо в наших прикладах, — розміщення параметрів у стек заздалегідь та виклик функції командою JMP. І нарешті, файл windows.inc у вас може просто не виявитися, так що будемо писати push перед кожним параметром вручну.

Зазначимо, що для TASM (і не тільки для цієї системи програмування) використовується бібліотека IMPORT32.LIB, що є "перехідником" імен викликаних API функцій. Використовуючи цю бібліотеку, програмісту необхідно додати її під час збирання програми, а коді програми вказувати імена функцій без __imp__ і @4().

Рекомендовані процедурні файли:

компіляція aw.bat:

c:\tasm\bin\tasm32.exe /m /ml /D_TASM_ %1.asm,%1.obj,%1.lst

складання - lw.bat (нова консоль):

c:\TASM\BIN\TLINK32 /Tpe /aa /c /x /M /s %1.obj, %1.exe, %1.map

збирання lc.ba:

c:\TASM\BIN\TLINK32 /Tpe /ap /c /x /M /s %1.obj IMPORT32.LIB, %1.exe, %1.map

run.bat - запуск в окремому вікні з фіксацією коду завершення програми:

@echo off

if "%1" == "" goto e1

echo Start %1 program.

%1

echo Program %1 виконано, exit code is %ERRORLEVEL%

goto ee

:e1

```
echo !!!!! error - 1-st parameter (program_name) не працює ...

goto ee

:ee

:PAUSE
```

Консольні програми

Виконані програми для Windows поділяються на два основних типи - консольні та графічні програми. При запуску консольної програми відкривається текстове вікно, з яким програма може спілкуватися функціями WriteConsole()/ReadConsole() та іншими (відповідно при запуску з іншої консольної програми, наприклад, файлового менеджера FAR, програмі відводиться поточна консоль і керування не повертається до FAR, поки програма не закінчиться). Графічні програми відповідно не отримують консолі і повинні відкривати вікна, щоб вивести щось на екран.

Для компіляції консольних програм будемо користуватися наступними командами:

MASM:

```
ml/c/coff/Cp winurl.asm
link winurl.asm / subsystem console
```

```
TASM:
tasm /m /ml /D_TASM_ winurl.asm
tlink32 /Tpe /ap /c /x winurl.obj
```

WASM:

```
wasm winurl.asm
wlink file winurl.obj form windows nt runtime console op з
```

Спробуйте скомпілювати програму winurl.asm у такий спосіб, щоб побачити, як відрізняється робота консольної програми від графічної.

Як приклад повноцінної консольної програми напишемо програму, яка перерахує всі підключені мережеві ресурси (диски та принтери), використовуючи системні функції WNetOpenEnum(), WNetEnumResource() та WNetCloseEnum().

```
; netenum.asm
; Консольний додаток для win32, що перераховує мережеві ресурси
include def32.inc
include kernel32.inc
include mpr.inc

.386
.model flat
.const
greet_message db 'Example win32 console program',0Dh,0Ah,0Dh,0Ah,0
error1_message db 0Dh,0Ah,'Could no get current user name',0Dh,0Ah,0
error2_message db 0Dh,0Ah,'Could not enumerate',0Dh,0Ah,0
good_exit_msg db 0Dh,0Ah,0Dh,0Ah,'Normal termination',0Dh,0Ah,0
enum_msg1 db 0Dh,0Ah,'Local',0
enum_msg2 db 'remote -',0
```

```
.data
user_name db 'List of connected resources for user '
user_buff db 64 dup (?); буфер для WNetGetUser
user_buff_l dd $-user_buff; розмір буфера для WNetGetUser
enum_buf_l dd 1056; довжина enum_buf у байтах
enum_entries dd 1; кількість ресурсів, які в ньому містяться
.data?
enum_buf NTRESOURCE <?,?,?,?,?,?,?,?>; буфер для WNetEnumResource
dd 256 dup (?); 1024 байт для рядків
message_l dd? ; змінна для WriteConsole
enum_handle dd? ; ідентифікатор для WNetEnumResource
.code
_start:
; отримаємо від системи ідентифікатор буфера виводу stdout
push STD_OUTPUT_HANDLE
call GetStdHandle; повертає ідентифікатор STDOUT у eax
mov ebx,eax; а ми зберігатимемо його в EBX

; виведемо рядок greet_message на екран
mov esi,offset greet_message
call output_string

; визначимо ім'я користувача, якому належить наш процес
mov esi,offset user_buff
push offset user_buff_l; адреса змінної з довжиною буфера
push esi; адреса буфера
push 0; NULL
call WNetGetUser
cmp eax, NO_ERROR; якщо сталася помилка
jne error_exit1; вийти
mov esi, offset user_name; інакше – виведемо рядок на екран
call output_string

; почнемо перерахування мережевих ресурсів
push offset enum_handle; ідентифікатор для WNetEnumResource
push 0
push RESOURCEUSAGE_CONNECTABLE ; всі ресурси, що приєднуються
push RESOURCETYPE_ANY ; ресурси будь-якого типу
push RESOURCE_CONNECTED ; тільки приєднані зараз
call WNetOpenEnum; почати перерахування
cmp eax, NO_ERROR; якщо сталася помилка
jne error_exit2; вийти

; цикл перерахування ресурсів
enumeration_loop:
push offset enum_buf_l; довжина буфера в байтах
push offset enum_buf; адреса буфера
push offset enum_entries; кількість ресурсів
push dword ptr enum_handle; ідентифікатор від WNetOpenEnum
call WNetEnumResource
cmp eax,ERROR_NO_MORE_ITEMS; якщо вони закінчились
je end_enumeration; завершити перерахування
cmp eax, NO_ERROR; якщо сталася помилка
jne error_exit2; вийти з повідомленням про помилку

; виведення інформації ресурсі на екран
mov esi, offset enum_msg1; перша частина рядка
call output_string; на консоль
mov esi,dword ptr enum_buf.lpLocalName; локальне ім'я пристрою
call output_string; на консоль
mov esi, offset enum_msg2; друга частина рядка
call output_string; на консоль
mov esi,dword ptr enum_buf.lpRemoteName ; віддалене ім'я пристрою
call output_string; туди ж
```

```
jmp short enumeration_loop; продовжимо перерахування
; кінець циклу
end_enumeration:
push dword ptr enum_handle
call WNetCloseEnum; кінець перерахування

mov esi,offset good_exit_msg
exit_program:
call output_string; виведемо рядок
push 0; код виходу
call ExitProcess; кінець програми
; виходи після помилок
error_exit1:
mov esi,offset error1_message
jmp short exit_program
error_exit2:
mov esi,offset error2_message
jmp short exit_program

; процедура output_string
; виводить на екран рядок
; введення: esi - адреса рядка
; ebx - ідентифікатор stdout чи іншого консольного буфера

output_string proc near

; визначимо довжину рядка
cld
xor eax,eax
mov edi,esi
repne scasb
dec edi
sub edi, esi
; пошлемо її на консоль
push 0
push offset message_l; скільки байт виведено на консоль
push edi; скільки байт треба вивести на консоль
push esi; адреса рядка для виведення на консоль
push ebx; ідентифікатор буфера виводу
call WriteConsole; WriteConsole(hConsoleOutput,lpvBuffer,cchToWrite,
; lpcchWritten,lpvReserved)
ret
output_string endp

end _start
```

У файл kernel32.inc треба додати між ifdef _TASM_ і else рядки:

```
extrn GetStdHandle:near
extrn WriteConsoleA:near
WriteConsole equ WriteConsoleA
```

і між else та endif:

```
extrn __imp__GetStdHandle@4:dword
extrn __imp__WriteConsoleA@20:dword
GetStdHandle equ __imp__GetStdHandle@4
WriteConsole equ __imp__WriteConsoleA@20
```

Крім того, потрібно створити файл mpr.inc:

```
; mpr.inc
; файл, що включається з визначеннями функцій з mpr.dll
;
ifdef _TASM_
includelib import32.lib
; імена використовуваних функцій
extrn WNetGetUserA:near
extrn WNetOpenEnumA:near
extrn WNetEnumResourceA:near
extrn WNetCloseEnum:near
; присвоєння для полегшення читання коду
WNetGetUser equ WNetGetUserA
WNetOpenEnum equ WNetOpenEnumA
WNetEnumResource equ WNetEnumResourceA
else
includelib mpr.lib
; справжні імена функцій, що використовуються
extrn __imp__WNetGetUserA@12:dword
extrn __imp__WNetOpenEnumA@20:dword
extrn __imp__WNetEnumResourceA@16:dword
extrn __imp__WNetCloseEnum@4:dword
; присвоєння для полегшення читання коду
WNetGetUser equ __imp__WNetGetUserA@12
WNetOpenEnum equ __imp__WNetOpenEnumA@20
WNetEnumResource equ __imp__WNetEnumResourceA@16
WNetCloseEnum equ __imp__WNetCloseEnum@4
endif
```

Ще буде потрібно файл def32.inc, в який помістимо визначення констант і структур з різних файлів, що включаються для мови С.

Існує утиліта h2inc, що перетворює ці файли повністю, але ми створимо свій вмиканий файл, в який будемо додавати нові визначення при необхідності.

```
; def32.inc
; файл з визначеннями констант та типів для прикладів програм під win32

; з winbase.h
STD_OUTPUT_HANDLE equ -11

; з winerror.h
NO_ERROR equ 0
ERROR_NO_MORE_ITEMS equ 259

; з winnetwk.h
RESOURCEUSAGE_CONNECTABLE equ 1
RESOURCETYPE_ANY equ 0
RESOURCE_CONNECTED equ 1
NTRESOURCE struc
dwScope dd?
dwType dd?
dwDisplayType dd ?
dwUsage dd ?
lpLocalName dd ?
lpRemoteName dd ?
lpComment dd?
lpProvider dd?
NTRESOURCE ends
```

Цей приклад, зрозуміло, можна було побудувати більш ефективно, виділивши великий буфер для WNetEnumResource(), наприклад за допомогою LocalAlloc() або GlobalAlloc() (у

Win32 це те саме), і потім, прочитавши інформацію про всі ресурси з нього, довелося б стежити за тим, чи скінчилися ресурси, і викликати WNetEnumResource() ще раз.

Звернімо увагу на параметри лінковника link32.exe /a... - його опис:

-ax Specify application type

-ap Windowing Compatible

-aa Uses Windowing API

У разі використання ключа /aa замість ключа /ap для консольної програми, необхідно на початку програми "отримати консоль" спеціальною API функцією

```
;запит консолі
call AllocConsole;
;перевірити успіх запиту консолі
test eax,eax
jz GetCon_err_exit          ;невдача
```

Для налагодження програми можна використовувати налагоджувачі Olly Debugger або, наприклад, -w32dasm

Для форматування та перекладу в текстовий рядок різноманітних даних зручно використовувати API функцію _wsprint (див. нижче)

)

Завдання лабораторної роботи

1. Підготувати робоче місце для роботи з консольними програмами "32 бітний додаток"

2. Виконати трансляцію та "вивчення" тестових програм

3. Встановити налагоджувачOlly Debuggerта виконати спостереження за роботою будь-якої тестової програми.

4. "Адаптувати" обчислення виразу в програму 32 бітного додатка та виконати перевірку роботи з використанням обраного вами відлатника.

Успіхів у роботі.

Завдання роботи

Основні обчислювальні операції.

План виконання.

1.Постановка задачі

Написати програму для обчислення значення заданого арифметичного виразу.

Типи змінних - цілі, довжиною 2 байти.

Працювати у полі цілих чисел.

Вихідні дані описати як ініціалізовані поля оперативної пам'яті.

Результати помістись у виділені слова.

Використовуючи налагоджувач – перевірити отримані результати.

2. Варіанти завдань

№	Завдання 1
1	$(a+b)^2-(a^2+2ab)$, при a=3, b=2
2	$(a-b)^2-(a^2-2ab)$, при a=2, b=5
3	$(a+b)^3-(a^3+3a^2b)$, при a=4, b=2
4	$3ab^2+b\ +3a^2b$, при a=1, b=4
5	$(a-b)^3-(a^3-3a^2b)$, при a=4, b=3
6	$(a-b)^3-(a^3-3ab^2)$, при a=5, b=3

7	$(a-b)^3-(a^3)$, при a=6, b=2
8	$(a+b)^4-(a^4+4a^3b+6a^2b^2)$, при a=1, b=2
9	$(a+b)^4-(a^4+4a^3b)$, при a=1, b=2
10	$(a-b)^4-(a^4-4a^3b+6a^2b^2)$, при a=5, b=2
11	$(a-b)^4-(a^4-4a^3b)$, при a=10, b=5
12	$(a+b)^2-(a^2+2ab)$, при a=4, b=2
13	$\frac{(a-b)^2-(a^2-2ab)}{b^2}$, при a=4, b=5
14	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$, при a=10, b=4
15	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$, при a=10, b=1
16	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$, при a=10, b=-1

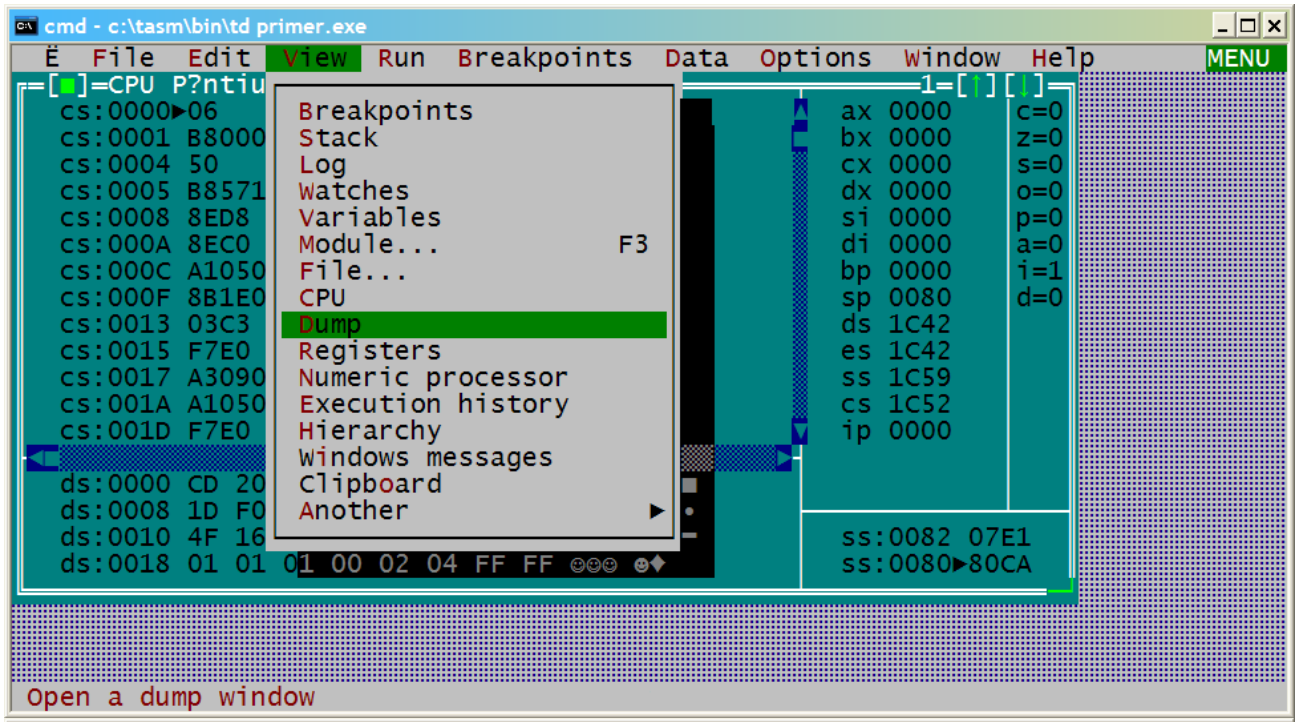
17	$\frac{(a-b)^3-(a^3-3ab^2)}{b^3-3a^2b}$, при a=5, b=10
18	$\frac{(a-b)^3-(a^3)}{b^3-3ab^2-3a^2b}$, при a = 8, b = 2
19	$\frac{(a+b)^4-(a^4+4a^3b+6a^2b^2)}{4ab^3+b^4}$, при a=1, b=5
20	$\frac{(a+b)^4-(a^4+4a^3b)}{6a^2b^2+4ab^3+b^4}$, при a = 1, b = 8
21	$\frac{(a-b)^4-(a^4-4a^3b+6a^2b^2)}{b^4-4ab^3}$, при a=1, b=10
22	$\frac{(a-b)^4-(a^4-4a^3b)}{6a^2b^2-4ab^3+b^4}$, при a=10, b=1
23	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$, при a=10, b=2
24	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$, при a = 8, b = 3
25	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$, при a=5, b=10

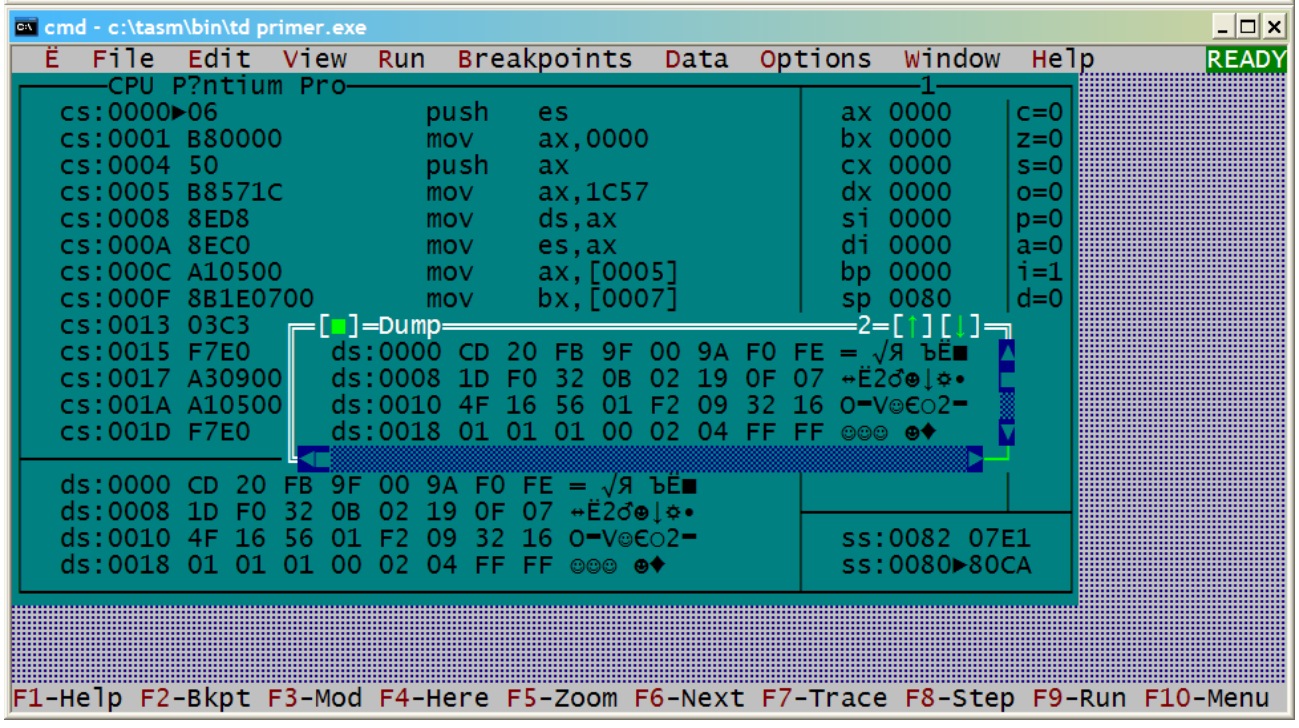
3. Методичні вказівки

1. Записати «за операторним» алгоритмом розрахунку виразу. Усі проміжні результати зберігати в ОП.

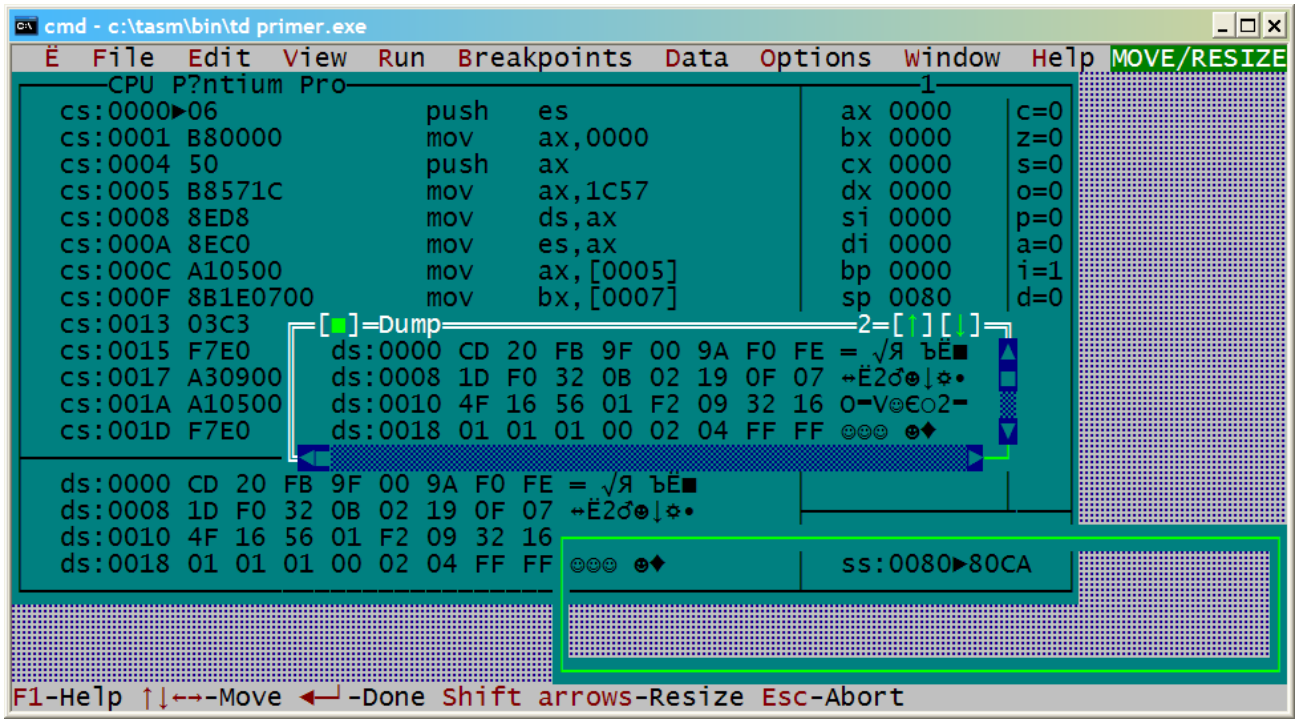
2. Розмістити всі дані у окремому сегменті.

3. При використанні відладчика – відкрити вікно View – Dump





4. Перемістити вікно – використовуючи клавіші CTRL-F5 та клавіші «стрілки» у зручне для перегляду місце



5. Використовуючи F7 – виконати програму, відстежуючи стан регістрів та областей ГП. Визначити адреси розміщення слів ОП, де охоронятимуться проміжні та остаточні результати. При необхідності – перемикання між вікнами – ALT – <клавіша – номер – вікна>

6. Помістити до звіту текст програми та фрагмент ОП (копія екрана вікна відладчика) з проміжними та остаточними результатами.

7. Вказати адреси та значення всіх використовуваних фрагментів ОП.

4. Приклад

Розробити алгоритм та програму мовою програмування асемблер для знаходження

$$(a+b)^2-(a^2+2ab)$$

при a=1, b=2.

Алгоритм покрокового рішення:

p1=(a+b)

p1 = p1 * p1

p2=a

p2=p2*p2

p3=2*a

p3=p3*b

p3=p2+p3

rez = p1-p3

Ручний розрахунок

=5

Текст програми

```
; обчислення (a+b)**2 - (a**2 + 2*a*b)
; результат у rez

code SEGMENT
ASSUME cs:code, ds:mydata, es:mydata
main proc FAR; стандартний початок прогр.
push es
mov ax,0
push ax
```

```
;------------------------

mov ax, mydata; завантаження адреси сегментів

mov ds,ax

mov es,ax

;------------------------ початок програми

mov ax,a; (a+b)**2

mov bx,b

add ax, bx

mul ax

mov p1,ax; збереження проміжного значення

;-----

mov ax,a

mul ax; a**2

mov p2,ax

;----

mov ax,2

mul a

mul b; 2*a*b

mov p3,ax

;----

mov ax,p2

add ax, p3; (a**2+2*a*b)

;-----

mov ax,p1

sub ax, p3

mov rez,ax
```

```
;-----------------------------------
RET
main endp

code ends

mydata segment
xxx db ''; що б у дампі легко можна було знайти
a dw 1
b dw 2
p1 dw?
p2 dw?
p3 dw?
rez dw?
mydata ends
-----------------------------------
STK SEGMENT STACK
DW 64 DUP (?)
STK ENDS
END main

  Результат розрахунку
  Адреси та значення змінних у сегменті mydata:
  xxx 00000 20 20 20 20
a 00004 01 00
b 00006 02 00
```

```
p1 00008 09 00

p2 0000a 01 00

p3 0000c 04 00

rez 0000e 05 00
```

Функція wsprintf

Функція wsprintf форматує та зберігає ряд символів та значень у буфері. Будь-які параметри перетворюються і копіюються в буфер даних, що виводяться відповідно до відповідної специфікації формату у форматованому рядку. Функція додає в кінець символ завершального нуля до символів, які вона пише, але у значенні, що повертається вона не включає його в число символів.

Синтаксис

```
int wsprintf(

LPTSTR lpOut,
LPCTSTR lpFmt,
...
);
```

Параметри

lpOut

[out]Вказівник на буфер, який отримає форматований виведення даних. Максимальний розмір буфера складає 1024 байти.

lpFmt

[in]Вказівник на рядок із завершальним нулем, який містить у собі специфікації управління форматом. На додаток до звичайних символів ASCII, специфікація формату для кожного параметра відображається в цьому рядку. Додаткові відомості про специфікацію формату див. у розділі Примітки.

...

[in]Вказує один або кілька параметрів. Число та тип параметрів параметра залежать від відповідних специфікацій управління форматом у параметрі lpFmt.

Значення, що повертається

Якщо функція завершується успішно, значення, що повертається - число символів, збережених у буфері виведених даних, не рахуючи символ завершального нуля.

Якщо функція завершується помилкою, значення, що повертається менше, ніж довжина очікуваного виведення даних. Щоб отримати додаткову інформацію про помилку, викличте**GetLastError**.

Зауваження

Попередження захисту!Використовуючи цю функцію неправильно, можна поставити під загрозу забезпечення безпеки Вашої програми. Рядок, повернутий у lpOut, не гарантовано від того, що буде ЗАВЕРШЕНО СИМВОЛОМ КІНЦЯ РЯДКУ ('\0'). Крім того, уникайте формату %s - він може призвести до переповнення буфера. Якщо відбувається порушення прав доступу, воно є причиною відмови від обслуговування спираючись на Ваш додаток. У гіршому випадку, зломщик захисту може вставити код, що виконується.

Рядок керування форматом містить специфікації формату, які визначають формат виведення даних для параметрів після параметра lpFmt. Специфікації формату, обговорені нижче, завжди починаються зі знака відсотка (%). Якщо знак відсотка супроводжується символом, який не має значення як поля формату, символ не форматує (наприклад, %% створює одиничний символ знака відсотка).

Рядок керування форматом читається зліва направо. Коли перша специфікація формату (якщо вона є) зустрічається, вона змушує значення першого параметра після рядка управління форматом бути перетвореним і скопійованим у буфер даних, що виводяться відповідно до специфікації формату. Друга специфікація формату змушує другий параметр бути перетвореним і скопійованим, і таке інше. Якщо більше параметрів, ніж у специфікації формату, додаткові параметри ігноруються. Якщо недостатньо параметрів для всіх специфікацій формату, результати не визначаються.

У специфікації формату - нижченаведена форма:

```
%[-][#][0][width][.precision]type
```

Кожне поле – одиничний символ або число, що показує конкретний варіант вибору формату. Символи типу, які відображаються після останнього додаткового поля формату, визначають, чи розуміється пов'язаний параметр символ, рядок, чи число. Найпростіша специфікація формату містить лише знак відсотка і символ типу (наприклад, %s). Додаткові поля керують іншими аспектами форматування. Нижче - додаткові та необхідні поля та їх значення.

Поле	*Призначення*
-	Доповнює виведення даних пробілами або нулями праворуч, щоб заповнити ширину поля, вирівнюючи виведення ліворуч. Якщо це поле пропускається, висновок доповнюється ліворуч, вирівнюючи його праворуч.
#	Префікс шістнадцяткового значення 0x (нижній регістр) або 0X (верхній регістр).

<table>
<tr><td>0</td><td>Доповнює вихідне значення нулями, щоб заповнити ширину поля. Якщо це поле пропускається, вихідне значення доповнюється пробілами.</td></tr>
<tr><td>width</td><td>Копіює вказану мінімальну кількість символів у буфер даних. Поле width - невід'ємне ціле число. Специфікація ширини ніколи не змушує значення обрізатись; якщо число символів у вихідному значенні більше, ніж вказана ширина, або, якщо поле width відсутнє, всі значення символів, що друкуються, підпорядковані специфікації точності.</td></tr>
<tr><td>.precision</td><td>Для чисел, скопіюйте вказане мінімальне число цифр у буфер даних. Якщо число цифр у параметрі менше, ніж вказана точність, вихідне значення доповнюється ліворуч із нулями. Значення не обрізається, коли кількість цифр виходить за межі вказаної точності. Якщо вказана точність 0 або повністю не включена, або якщо точка (.) відображається без числа після неї, точність встановлюється в 1.</td></tr>
<tr><td></td><td>Для рядків, скопіюйте вказану максимальну кількість символів у буфер даних.</td></tr>
<tr><td>type</td><td>Виберіть відповідний параметр як символ, рядок або число. Це поле може бути будь-яким з наведених нижче значень.
<table>
<tr><th>Значення</th><th>Призначення</th></tr>
<tr><td>c</td><td>Одиничний символ. Це значення інтерпретується як тип WCHAR, якщо додаток визначає Unicode і як тип __ wchar_t в іншому випадку.</td></tr>
<tr><td>C</td><td>Одиничний символ. Це значення інтерпретується як тип __wchar_t, якщо додаток визначає Unicode і як тип WCHAR в іншому випадку.</td></tr>
<tr><td>d</td><td>Знакове десяткове ціле число. Це значення еквівалентне i.</td></tr>
<tr><td>hc, hC</td><td>Одиничний символ. Функція wsprintf ігнорує символьні параметри із числовим значенням нуля. Це значення завжди інтерпретується як тип __ wchar_t, навіть тоді, коли додаток визначає Unicode.</td></tr>
<tr><td>hd</td><td>Параметр знакового короткого цілого числа.</td></tr>
<tr><td>hs, hS</td><td>Рядок. Це значення завжди інтерпретується як тип LPSTR, навіть тоді, коли додаток, що викликає, визначає Unicode.</td></tr>
<tr><td>hu</td><td>Беззнакове коротке ціле число.</td></tr>
<tr><td>i</td><td>Знакове десяткове ціле число. Це значення еквівалентне d.</td></tr>
<tr><td>lc, lC</td><td>Одиничний символ. Функція wsprintf ігнорує символьні параметри із числовим значенням нуля. Це значення завжди інтерпретується як тип WCHAR, навіть тоді, коли додаток не визначає Unicode.</td></tr>
<tr><td>ld</td><td>Знакове довге ціле число. Це значення еквівалентне li.</td></tr>
<tr><td>li</td><td>Знакове довге ціле число. Це значення еквівалентно ld.</td></tr>
</table></td></tr>
</table>

	ls, lS	Рядок. Це значення завжди інтерпретується як тип LPWSTR, навіть тоді, коли додаток не визначає Unicode. Це значення еквівалентне ws.
	lu	Беззнакове довге ціле число.
	lx, lX	Довге шістнадцяткове число без знака в нижньому регістрі або верхньому регістрі.
	p	**Windows 2000/XP:**Покажчик. Адреса друкується, використовуючи шістнадцяткову систему.
	s	Рядок. Це значення інтерпретується як тип LPWSTR, коли додаток визначає Unicode і як тип LPSTR в іншому випадку.
	S	Рядок. Це значення інтерпретується як тип LPSTR, коли додаток визначає Unicode і як тип LPWSTR в іншому випадку.
	u	Параметр цілого числа без значка.
	x, X	Шістнадцяткове ціле число без знаку у нижньому або верхньому регістрі.

Зверніть увагу! Це важливо!на те, що wsprintf, використовує угоду про виклики C (_cdecl), а не стандартний виклик (_stdcall). В результаті цього - відповідальність викликаного процесу витягти зі стека параметри зі стека і помістити параметри в стек праворуч наліво. У C-мовних модулях компілятор C виконує це завдання.