Лабораторна робота №1

Знайомство з пакетом програм MASM32

Мета: Навчитися використовувати пакет MASM32, а також вивчити типову структуру вихідного тексту простої програми Win32 на мові асемблера.

Завдання:

1. Інсталювати програмний пакет MASM32.
2. Створити проект. Написати вихідний текст програми на асемблері.
3. Скомпілювати вихідний текст і отримати виконуємий файл програми.
4. Перевірити роботу програми.
5. Внести зміни у вихідний текст згідно завданню, перевірити результат.
6. Знайти відомості у довідці MASM32 для відповідей на контрольні питання.

Теоретичні відомості

Асемблер – це мова програмування, яка максимально наближена до машинної мови цифрового комп'ютера.

Формат вихідного тексту програми

Текст програми на мові асемблера складається з багатьох рядків, у кожному рядку загалом можуть бути записані такі елементи: мітка, команда або директива, коментар:

[мітка] [команда] [;коментар]

або

[мітка] [директива] [;коментар]

Квадратні дужки позначають необов'язкові елементи. Будь-який елемент є необов'язковим – можуть бути і порожні рядки (це використовується для покращення сприйняття людиною окремих блоків тексту). Порожні рядки асемблер ігнорує.

Мітка – складається з букв англійського алфавіту, цифр та символів _, @ (окрім вказаних, для мітки можуть використовуватися й інші символи, проте це не рекомендується). Мітка не повинна починатися з цифри.

Коментар – це текст пояснень, починається з символу '; ' і містить будь-який набір символів до кінця рядка. Коментар асемблер ігнорує.

Скелет вихідного тексту програми

Текст програми складається з набору базових блоків тексту, які виділяються директивами асемблера. Запис директив, їхній набір та синтаксис можуть бути різним – це залежить, по-перше від типу та версії асемблера, а по-друге типом програми (або програмного модуля) і середовищем, у якому повинна виконуватися програма – типом процесора, режимом роботи процесора та операційною системою.

Узагальнено скелет вихідного тексту можна представити як три обов'язкові директиви, які повинні бути завжди, та решту частину тексту:

```
Директива, що визначає тип процесора
Директива, що визначає модель памяті

Решта частина тексту

Директива END
```

Розглянемо скелет тексту програми Win32 – звичайної програми, розробленої для роботи у середовищі 32-бітної операційної системи Windows. Така програма впродовж роботи буде викликати різноманітні системні функції. Зокрема, будь-яка програма для коректного завершення своєї роботи обов'язково повинна викликати функцію ExitProcess системної бібліотеки kernel32. Крім того, для програм Win32 типовим є використання елементів графічного інтерфейсу користувача (бібліотека user32) та графіки (gdi32).

```
.386
.model flat, stdcall
option casemap :none  ; розрізнювати великі та маленькі букви

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\gdi32.inc
; а також include для інших заголовочних файлів

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib
; а також includelib для інших бібліотек
```

```
.const
; оголошення та ініціалізація констант

.data
; оголошення статичних даних (наприклад, глобальних перемінних)

.code
start:

; програмний код

  invoke ExitProcess, 0     ; завершення роботи програми
end start
```

Директива .386 означає, що програма призначається для виконання на процесорах типу Intel 80386 і вище.
Директивою .MODEL вказана модель адресації пам'яті **flat** та спосіб (конвенція) виклику **stdcall**.
Точка входу у програму позначається міткою, ім'я якої потім вказане у директиві END – у наведеному вище тексті це ім'я "start". Точкою входу буде перша команда за цією міткою.
Директива INVOKE асемблера MASM означає виклик процедури. ЇЇ зручно використовувати замість традиційного **call** для виклику процедур з параметрами, які передаються через стек командами **push**. Наприклад, замість

```
push style
push offset Caption
push offset Text
push hWnd
call MessageBoxA
```

зручніше записати

```
invoke MessageBoxA, hWnd, ADDR Text, ADDR Caption, style
```

Це робить синтаксис асемблера дещо подібним до мови високого рівня.

Наприкінці роботи програма викликає системну функцію **ExitProcess** з параметром (аргументом) нуль – нульове значення буде передане операційній системі як код завершення процесу. Зазвичай прийнято, що нульове значення означає нормальне завершення процесу.

Директиви створення даних

У програмах часто використовуються дані у вигляді перемінних та констант. Для створення таких даних у асемблері є директиви:

DB (*define byte*) – визначає перемінну розміром у 1 байт;
DW (*define word*) – визначає перемінну розміром у 2 байти (слово);
DD (*define double word*) – визначає перемінну розміром у 4 байти (подвійне слово);
DQ (*define quad word*) – визначає перемінну розміром у 8 байтів (квадрослово);
DT (*define ten bytes*) – визначає перемінну розміром у 10 байтів.

Усі ці директиви можуть бути використані як для створення простих перемінних та констант, так і для створення масивів. Масиви для рядків символів зазвичай створюють директивою DB. Формат запису директиви для створення перемінної або константи такий:

<ім'я> D* <операнд> [, <операнд>]

де позначка D* означає одну з директив DB, DW, DD, DF, DQ або DT.

Створення перемінної може бути суміщено із початковою ініціалізацією. Операнд вказує початкове значення перемінної. У якості операнду може записуватися число, символ або знак питання '?', який використовується для перемінних без початкової ініціалізації.

Якщо у якості операнду вказується рядок або якщо записані декілька операндів через кому, то пам'ять виділяється для декількох перемінних вказаного типу, тобто буде масив. Щоб при ініціалізації масивів не вказувати багаторазово одне значення, записують DUP (значення).

Приклади створення даних:

```
a dd ?                      ; перемінна розміром 4 байти, неініціалізована
b dd 1.25                   ; перемінна 4-байтова, ініціалізована числом 1.25 з плаваючою точкою
c db 10011001b              ; перемінна розміром 1 байт, ініціалізована двійковим числом
d db '!'                    ; перемінна розміром 1 байт, ініціалізована символом '!'
e dw 1234h                  ; перемінна розміром 2 байти, ініціалізована шістнадцятьковим числом
f db 'рядок',0                          ; рядок з 6 байтів, завершується нулем (як прийнято у C, C++)
g db 'рядок',13,10                      ; рядок з 7 байтів, завершується символами CR (13), LF (10)
h db 'рядок1',13,10,'рядок2',0          ; символи у двох рядках, завершуються нулем
k dd 100 dup (?)                        ; масив із 100 подвійних слів, неініціалізований
m dd 100 dup (0)                        ; масив із 100 подвійних слів, у які записано 0
n dq 1.25                   ; перемінна 8-байтова, ініціалізована числом 1.25 з плаваючою точкою
p dq 10 dup (0,1,2)         ; масив з 30 квадрослів – 10 трійок, ініціалізованих значеннями 0, 1, 2
q dt 1.25                   ; перемінна 10-байтова, ініціалізована числом 1.25 з плаваючою точкою
```

Для перемінних та констант можна застосувати операції **offset** і **type**. Операція **offset** знаходить адресу розташування у пам’яті, **type** – розмір.

Засоби розробки програм на асемблері

Для створення програми, написаної на асемблері, потрібні: редактор вихідного тексту, компілятор, лінкер. Вихідний текст на асемблері записується у файл *.asm, який потім компілюється, лінкується і створюється виконуємий файл *.exe. Наприклад, файл вихідного тексту **myprog.asm** компілюється у об'єктний файл **myprog.obj**. Потім на основі об'єктного файлу myprog.obj лінкер записує виконуємий файл **myprog.exe**. Виконуємий файл є результатом роботи програміста – користувача пакету розробки програм.

Окрім вказаних засобів, корисно мати налагоджувач (debugger) для відстеження роботи програми, щоб узнати, де виникає помилка. У пакеті MASM32 його немає.

Порядок виконання роботи та методичні рекомендації

1. Інсталяція пакету MASM32 та підготовка до роботи.

Спочатку необхідно інсталювати пакет MASM32. Цей пакет являє собою програмне забезпечення, яке є вільним (freeware) для розповсюдження та використання, у тому числі для навчальних цілей. Знайти інсталяційні файли можна у мережі Інтернет за електронною адресою **http://www.masm32.com**. Після завантаження потрібних файлів перевірте їх на відсутність комп'ютерних вірусів. Потім запустіть на виконання файл **install.exe**. Дочекайтеся повідомлення про успішне завершення інсталяції MASM32. Після завершення інсталяції на жорсткому диску комп'ютера буде записано множину різноманітних файлів – у тому числі виконуємі файли програм. Увага! Перед тим, як щось робити з отриманими файлами, обов'язково знову перевірте усі файли на відсутність комп'ютерних вірусів. Порада: у папках **\masm32\examples** вилучіть з жорсткого диску усі файли ***.exe** скомпільованих прикладів (попередньо записавши їхні імена – ці імена буде корисно знати для подальшого вивчення пакету MASM32).

2. Створення проекту у середовищі MASM32.

У розділі (локальному диску), де міститься інстальований MASM32, створіть робочу папку вашої програми. Наприклад, якщо MASM32 розташовується у E:\masm32, то створіть робочу папку, наприклад, E:\work\lab1.

Потім викличте програму MASM32 Quick Editor (виконуємий файл \masm32\qeditor.exe).

Введіть текст найпростішої програми для Windows. Потім запишіть цей текст у файл типу *.asm у робочу папку на локальному диску – наприклад, як E:\work\lab1\lab1.asm. Зробити це можна через меню "File - Save As".

У такий спосіб створюються файли вихідного тексту. Дана програма містить лише один файл вихідного тексту. Складніші програми можуть містити декілька файлів у робочій папці проекту.

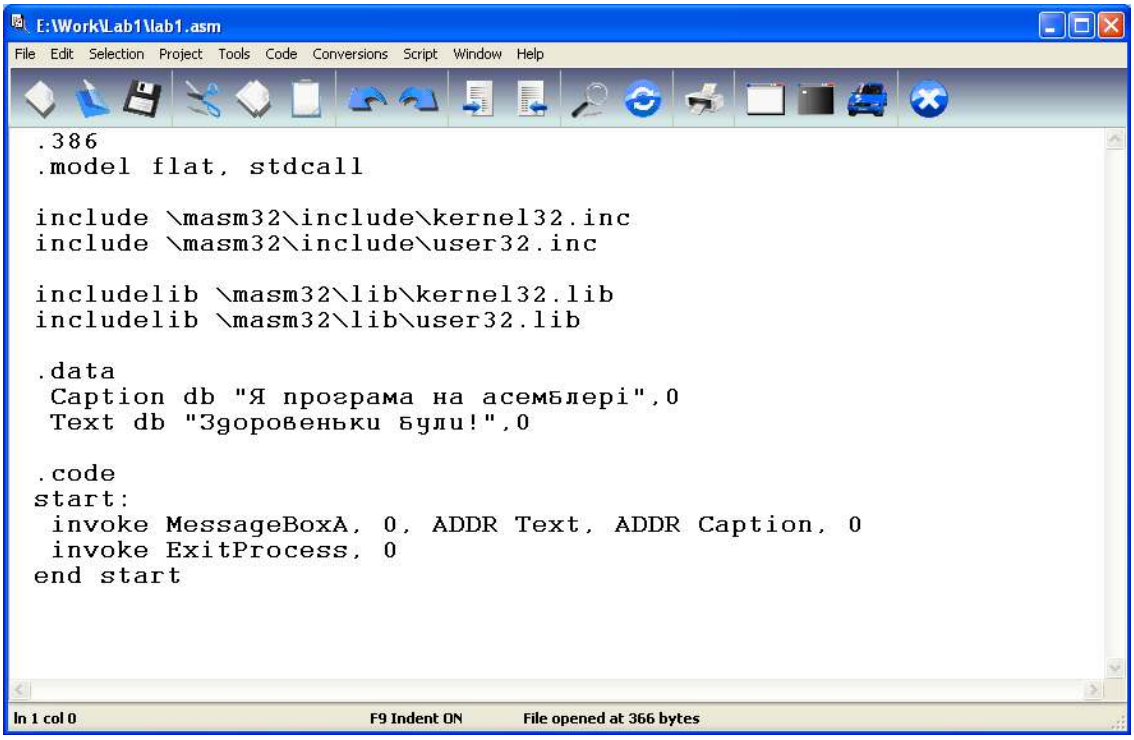

Рис. 1. Вихідний текст у вікні MASM32 Quick Editor

3. Компіляція та створення виконуємого файлу програми.

Виберіть меню "Project - Assemble ASM File". Якщо компілятор не знайде помилок, то у робочу папку буде записаний файл lab1.obj.

Наступним кроком буде лінкування. Виберіть меню "Project – Link OBJ File". У разі відсутності помилок лінкер запише виконуємий файл програми lab1.exe.

Необхідно вказати, що компіляцію та лінкування можно об’єднати – виконати за один раз вибором меню "Project – Assemble & Link". Проте і у цьому випадку послідовність обробки вихідного тексту буде тою самою – спочатку компіляція, потім лінкування. Інформація про виконання цієї роботи буде відображатися у вікні командного рядка

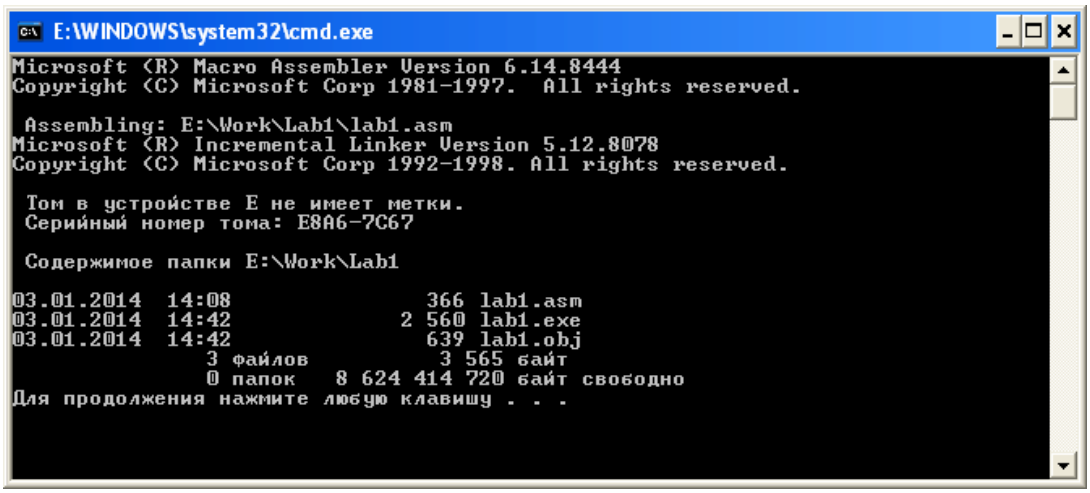

Рис. 2. Інформація про виконання компіляції та лінкування

Таким чином, у робочій папці з’явився виконуємий файл lab1.exe.

4. Перевірка роботи програми.

Щоб перевірити, як працює новостворена програма lab1.exe, її можна викликати засобами операційної системи, або безпосередньо у середовищі MASM32 Editor. Для цього виберіть меню "Project – Run Program".

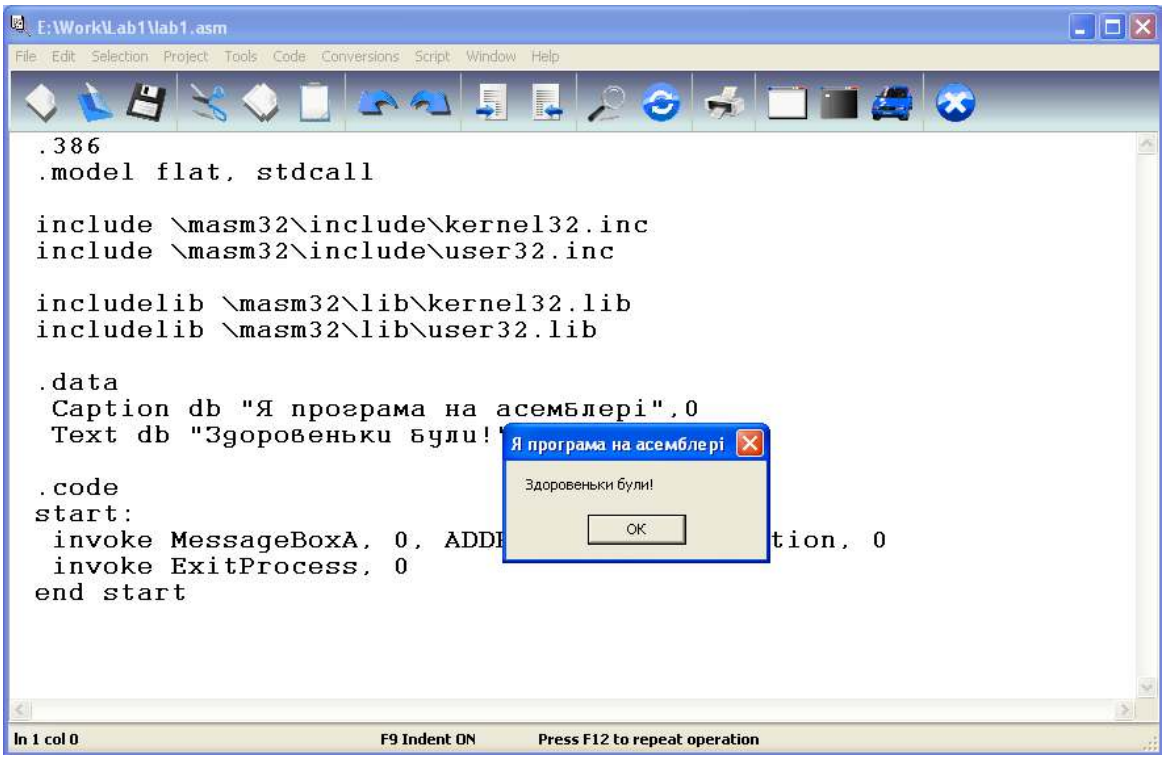

Рис. 3. Виклик програми на виконання

Наша перша програма на асемблері виводить текст вітання у діалоговому вікні.

5. Редагування вихідного тексту програми

Потрібно змінити, відредагувати вихідний текст програми згідно варіанту завдання. Це виконується у програмі MASM32 Quick Editor.

Для вирішення завдання необхідно відповідно оформити потрібний текст директивою DB у секції .DATA. Студент повинен самостійно розібратися у питанні, як можна записувати текст з декількох рядків у діалоговому вікні. Порада: для цього слід використовувати пари символів з кодами 13 та 10 у якості роздільників рядків.

Варіанти завдання

Варіант завдання – це прізвище, ім’я та по-батькові студента (П.І.Б.).

Необхідно запрограмувати, щоб разом із попереднім текстом вітання у цьому ж діалоговому вікні програми виводилася б інформація (П.І.Б.) про студента як автора програми.

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка тексту програми
4. Роздруківка результатів виконання програми
5. Аналіз результатів
6. Висновки

Контрольні питання:

1. З чого складається вихідний текст на асемблері?
2. Що означає **.386**?
3. Що означає **.model flat, stdcall ?**
4. Що означають **include** та **includelib**?
5. Що означає директива **DB**?
6. Де точка входу у програму?
7. Що таке **invoke**?
8. Які типи аргументів у функції **MessageBox**?
9. Навіщо потрібна функція **ExitProcess**?
10. Що роблять компілятор та лінкер?