МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ
УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
« ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З КУРСУ

«СИСТЕМНЕ ПРОГРАМУВАННЯ»

для студентів спеціальностей:
7.05010201 «Комп'ютерні системи та мережі»,
7.05010202 «Системне програмування»,
7.05010203 «Спеціалізовані комп'ютерні системи»

Затверджено
редакційно-видавничою
радою університету,
протокол № 1 від 20.06.2012

Харків
НТУ "ХПІ"
2013

Методичні вказівки до виконання лабораторних робіт з курсу «Системне програмування» для студентів спеціальностей: 7.05010201 «Комп'ютерні системи та мережі», 7.05010202 «Системне програмування», 7.05010203 «Спеціалізовані комп'ютерні системи» / уклад. О. М. Рисований. – Х. : НТУ «ХПІ», 2013. – 216 с.

Укладач О. М. Рисований

Рецензент А. І. Поворознюк

Кафедра обчислювальної техніки та програмування

ЗМІСТ

ВСТУП

Мова асемблера – система позначень, використовувана для уявлення в легкій формі для читання програм, записаних у машинному коді. Мова асемблера дозволяє програмістові користуватися алфавітними мнемонічними кодами операцій, на свій розсуд привласнювати символічні імена регістрам ЕОМ і пам'яті, а також задавати зручні для себе схеми адресації (наприклад, індексну або непряму). Крім того, вона дозволяє використовувати різні системи числення (наприклад, десяткову або шістнадцяткову) для подання числових констант і дає можливість позначати рядки програми мітками з символічними іменами, щоб до них можна було звертатися (по іменах, а не по адресах) з інших частин програми (наприклад, для передачі управління) [1, 4–6].

Переклад програми на мові асемблера в здійснимий машинний код (обчислення виразів, розкриття макрокоманд, заміна мнемонік власне машинними кодами і символьних адрес на абсолютні або відносні адреси) проводиться асемблером – програмою-транслятором, яка і дала мові асемблера її назву [2, 3].

Переваги мови асемблера:

– мова асемблера дозволяє писати найшвидший і найкомпактніший код, який взагалі можливий для даного процесора. Збільшення швидкості виконання програми досягається за рахунок оптимізації обчислювального алгоритму і/або раціональнішого звернення до ОП, перерозподілу даних. Зменшення обсягу кода досягається за рахунок ефективного використання проміжних результатів і раціонального використання елементів пам'яті. (Скорочення обсягу коду також нерідко підвищує швидкість виконання програми);

– забезпечення максимального використання специфічних можливостей конкретної платформи, що також дозволяє створювати ефективніші програми, зокрема менш ресурсомісткі. При програмуванні на мові асемблера можливий безпосередній доступ до апаратури, і, зокрема, портів введення-виведення, регістрів процесора та ін. Мова асемблера часто застосовується для створення драйверів пристроїв і ядра операційної системи (або машиннозалежних підсистем ядра ОС);

– мова асемблера використовується для створення «прошивок» BIOS;

– за допомогою мови асемблера часто створюються машиннозалежні підпрограми компіляторів та інтерпретаторів мов високого рівня, а також реалізується сумісність платформ;

– за допомогою дизасемблера можливо досліджувати існуючі програми за відсутності початкового коду.

Недоліки мови асемблера:

– через машинну орієнтацію («низького» рівня) мови асемблера людині складніше читати і розуміти програму в порівнянні з мовами програмування високого рівня; програма складається з дуже «дрібних» елементів – машинних команд, відповідно, ускладнюються програмування і налагоджування, зростають трудомісткість і ймовірність внесення помилок;

– потрібна підвищена кваліфікація програміста для отримання якісного коду: код, написаний середнім програмістом на мові асемблера, зазвичай виявляється не кращим або навіть гіршим за код, що породжується оптимізуючим компілятором для порівнянних програм, написаних на мові високого рівня;

– трудність перекомпіляції програми під особливості нової платформи.

Історично, якщо першим поколінням мов програмування вважати машинні коди, то мову асемблера можна розглядати як друге покоління мов програмування. Недоліки мови асемблера, складність розробки на ньому великих програмних комплексів зумовили появу мов третього покоління – мов програмування високого рівня (таких як Фортран, Лісп, Кобол, Паскаль, Сі та ін.). Саме мови програмування високого рівня та їх спадкоємці в основному використовуються в даний час в індустрії інформаційних технологій. Проте мова асемблера зберігає свою нішу, обумовлену її унікальними перевагами в частині ефективності і можливості повного використання специфічних засобів конкретної платформи.

З використанням програмування на мові асемблера проводяться:

– оптимізація критичних до швидкості ділянок програм у програмах на мовах високого рівня. Це особливо актуально для ігрових приставок, що мають фіксовану продуктивність, і для мультимедійних кодеків, які прагнуть робити менш ресурсомісткими і швидшими;

– створення операційних систем (ОС) або їх компонентів. У даний час переважну більшість ОС пишуть на більш високорівневих мовах (в основному на Сі – мові високого рівня, яка спеціально була створена для написання однієї з перших версій UNIX). Апаратні залежні ділянки коду, такі як завантажувач ОС, рівень абстрагування від апаратного забезпечення (hardware abstraction layer) і ядро часто пишуться на мові асемблера. Фактично асемблерного коду в ядрах Windows або Linux зовсім небагато, оскільки автори прагнуть забезпечити переносимість і надійність, але він там присутній. Деякі любительські ОС, такі як MENUETOS, цілком написані на мові асемблера. При цьому MENUETOS поміщається на дискету і містить графічний багатовіконний інтерфейс;

– створення драйверів. Деякі частини драйверів програмують на мові асемблера. Хоча в цілому у даний час драйвери також прагнуть писати на мовах високого рівня у зв'язку з підвищеними вимогами до надійності, достатньою продуктивністю сучасних процесорів і досконалістю компіляторів з мов високого рівня. Надійність для драйверів має особливе значення, оскільки в Windows NT і UNIX (зокрема в Linux) драйвери працюють в режимі ядра. Одна помилка в драйвері може привести до краху всієї системи;

– створення антивірусів та інших захисних програм;

– написання трансляторів мов програмування.

Методичні рекомендації призначені для студентів, які вивчають мову асемблера та мають мету отримання практичних навичок при:

– розробці прикладних програм з застосуванням ефективних прийомів та технологій;

– налагодженні та тестуванні програм;

– застосуванні низхідної технології проектування програм;

– використовуванні стандартних бібліотечних функцій, процедур та модулів, а також виконанні розробки власних компонентів;

– розбивці програм на складові частини (процедури, модулі, файли).

1. ЕТАПИ СТВОРЕННЯ ПРОГРАМИ НА МОВІ АСЕМБЛЕРА

Асемблер – це програма, яка перетворює початковий текст програми, написаної на мові асемблера, в машинний код. Асемблер може створювати лістинг програми з номерами рядків, адресами змінних, операторами та таблицею перехресних посилань символів і змінних. З асемблером використовується програма, названа компонувальником (linker), яка об'єднує окремі файли, створені асемблером, в єдину виконувану програму.

При вивченні мови асемблера та розробці реальних програм на цій мові під операційну систему (ОС) Windows процес розподілення комірок пам'яті для збереження кодів команд та чисел виконується автоматизовано.

Етапи створення програми на мові асемблера такі:

- підготовка (або внесення змін) вихідного тексту програми;
- асемблювання програми (отримання об'єктного коду);
- компонування програми (отримання виконавчого файлу програми);
- налагодження програми (виправлення помилок).

Зазвичай ці етапи циклічно повторюються, оскільки при виявленні помилок на всіх етапах доводиться повертатися до першого етапу і вносити зміни в текст програми для виправлення помилок.

Асемблери бувають двох типів: однофазні та двофазні.

Однофазні асемблери можуть обробляти тільки такі програми, в яких символи з'являються в полі назви до того, як на них дається посилання в полі операндів.

Трансляція програми двофазним асемблером проводиться в два етапи: у першій фазі трансляції асемблер послідовно рахує кожне речення початкової програми, частково її транслює і будує повну таблицю символів; у другій фазі асемблер закінчує трансляцію програми, використовуючи таблицю символів першої фази як вхідну інформацію.

Для обох типів асемблерів символ, записаний у полі операндів, обов'язково повинен бути визначений у полі назви, інакше буде повідомлення про помилку.

1.1. Підготовка тексту програми

Текст програми на мові асемблера записується в один чи декілька текстових файлів. Імена файлів і їх розширення можуть бути будь-які, але прийнято використовувати розширення *.asm. Ці файли є текстовими, їх можна підготовити за допомогою будь-якого текстового редактора, наприклад блокнота або NotePad, і зберігати у вигляді звичайних файлів у форматі ASCII. Використання Word небажано, бо такий текс

т містить велику кількість службових символів, які є невидимими та призводять до помилок трансляції.

Найпростіше при підготовці тексту можна використати саме середовище Masm32.

Структура програми під Win32 може бути такою:

```
TITLE  <ім'я програми
.386                ; директива визначення типу мікропроцесора
.model flat         ; задання лінійної моделі пам'яті
.data                       ; директива визначення даних
                            ; дані, які визначені
.data?                      ; неініціалізовані дані
                            ;
.code                         ; директива початку коду програми
label:                      ; мітка початку програми
            ;<code>
ret                         ; повернення управління ОС
 end  label                  ; закінчення програми
```

1.2. Асемблювання програми

Підготовлені тексти програми з розширенням *.asm є вхідними даними для програм, які називаються асемблерами (наприклад, програми Masm, Tasm, Wasm, Fasm). Завдання програми асемблера – перетворити текст програми у форму двійкових команд, які може виконати МП. Після асемблювання отримують файли об'єктних модулів, що мають розширення *.obj. Процес асемблювання частіше називають ***трансляцією***.

Для простоти використання асемблера Masm32 (www.masm32.com) його необхідно інсталювати на системний диск c:/. Якщо поставити асемблер Masm32 не на системний диск, а на інший логічний диск, то в такому випадку необхідно конкретно вказувати шляхи підключення бібліотек розташування API-функцій.

Програма ml.exe з Masm32 виконує трансляцію початкового тексту програми в проміжний об'єктний файл. Програма link.exe виконує

компонування об'єктного (об'єктних) файлу (файлів) і бібліотек в єдину виконувану програму.

В асемблері masm32 трансляцію файлів з розширенням asm можна виконати з командного рядка:

ml.exe /c /coff ім'я_файлу.asm

Створений за допомогою цієї команди об'єктний файл має формат COFF. Якщо параметр /coff не заданий, то форматом створеного об'єктного файлу буде OMF.

Параметр /C означає, що виконується тільки асемблювання. Виклик лінкеру link.exe поки не виконується, тому що його викликаємо вручну пізніше.

Компонувальник link.exe оперує з OBJ-файлами як у форматі COFF, так і у форматі OMF, при цьому виконується автоматичне перетворення формату файлу з OMF в COFF. Зазвичай при генерації виконуваних файлів використовується формат COFF. Якщо файл об'єктного модуля повинен використовуватися у застосуванні, написаному на Visual C++ .NET, то формат його обов'язково повинен бути COFF. У той же час при використанні об'єктного файлу у застосуванні, розробленому в Borland Delphi 2005, єдиним сприйманим форматом є OMF.

У процесі трансляції початкового тексту програми виконуються такі дії:

1. Аналізуються директиви умовного асемблювання, і у разі істинності вказаних у них умов виконуються ті або інші кроки.
2. Розгортаються макроси.
3. Обчислюються константні вирази, при цьому вони заміщаються обчисленими значеннями.
4. Декодуються команди та операнди, що не знаходяться в пам'яті. Наприклад, декодується команда mov EAX, 2008, оскільки вона не має операндів, розташованих у пам'яті.
5. Зберігаються зсуви змінних у пам'яті відносно зміщення у сегментах, в яких ці змінні розташовані.
6. Сегменти та їх атрибути розміщуються в об'єктному файлі.
7. У об'єктному файлі зберігаються переміщувані адреси (relocatable addresses).
8. При необхідності створюється файл лістингу.
9. Безпосередньо програмі link.exe передаються деякі директиви (наприклад, INCLUDELIB і DOSSEG).

Опції компіляції ML.EXE:

ML [/ опції] список_файлів [/link опції_лінкера]

/AT– дозволити надмаленьку модель пам’яті tiny (.COM файл);
/Bl<linker> – використовувати альтернативний лінкер;
/c – асемблювати без лінкування (використання зовнішнього лінкера (наприклад link.exe) для компонування файлів);
/Cp – зберегти регістр ідентифікаторів користувача;
/Cu – перевести всі ідентифікатори у верхній регістр;
/Cx – зберегти регістр publics, externs;
/coff – генерувати об’єктний файл у форматі COFF;
/D<name>[=text] – визначити текст макросу;
/EP – виводити лістинг препроцесора в стандартний потік виведення;
/F <hex> – встановити розмір стека (в байтах);
/Fe<file> – встановити ім’я виконуваного файлу;
/Fl[file] – генерувати лістинг;
/Fm[file] – генерувати карту пам’яті;
/Fo<file> – ім’я об’єктного файлу;
/FPi – генерувати 80x87 емулятор;
/Fr[file] – встановити обмежену інформацію;
/FR[file] – встановити повну інформацію;
/G<c|d|z> – використовувати формат виклику в стилі Pascal, C або Stdcall;
/H<number> – встановити максимальну довжину зовнішніх імен;
/I<name> – додати include-шлях;
/link <опції лінкеру та бібліотеки>;
/nologo – заборонити повідомлення ML.EXE про авторське право;
/Sa – максимізувати лістинг;
/Sc – видавати таймінги в лістинг;
/Sf – видавати лістинг першого проходження асемблера;
/Sl<width> – встановити ширину рядка;
/Sn – подавити лістинг таблиці символів;
/Sp<length> – встановити довжину сторінки;
/Ss<string> – встановити підзаголовок;
/St<string> – встановити заголовок;
/Sx – скласти список умовних виразів помилок;
/Ta<file> – асемблювати не .ASM файли;
/w – аналогічно /W0 /WX;
/WX – розцінювати попередження як помилку;
/W<number> – встановити рівень попереджень;
/X – проігнорувати INCLUDE шлях оточення;
/Zd – додати номери рядків у налагоджувальну інформацію;

/Zf – зробити всі символи загальнодоступними;
/Zi – додати символічну налагоджувальну інформацію;
/Zm – забезпечити сумісність з MASM 5.10;
/Zp[n] – встановити вирівнювання структур;
/Zs – виконати тільки перевірку синтаксису.

Для трансляції та лінкування *простих* програм **зручно** використовувати середовище masm32. Для цього необхідно запустити програму masm32 Editor, вставити або набрати свою програму, зберегти її з розширенням asm та вибрати шлях меню Project/ Assemble & Link. Якщо помилок не буде, то буде створено exe-файл.

1.3. Компонування програми

Після успішної трансляції початкового тексту асемблерної програми результат у вигляді об'єктного файлу передається компонувальнику link.exe. Він виконує об'єднання об'єктних модулів в один файл. Сегменти, визначені в програмі, групуються відповідно до інструкцій, що містяться в об'єктному файлі. Вся інформація про розміщення сегментів записується в заголовок виконуваного файлу.

Результатом компонування є виконавчі файли, що мають розширення *.exe.

В асемблері masm32 для генерування 32-розрядних EXE-файлів може бути використаний один із командних рядків:

link.exe /SUBSYSTEM:WINDOWS /OPT:NOREF ім'я_файлу.obj

або

link.exe /SUBSYSTEM:CONSOLE ім'я_файлу.obj

У **більшості випадків** для виконання програм зручно створити командний bat-файл, наприклад, зі змістом:

```
ml.exe  /c  /coff   7_9_1.asm
link.exe /subsystem:console 7_9_1.obj
```

Після того, як такий командний bat-файл створено, необхідно його виконати (двічі натиснути на нього).

Схему асемблювання, компонування і виконання програми наведено на рис. 1.1.

Структура програми (не тільки на асемблері) визначається декількома чинниками:

– архітектурою мікропроцесора;

– особливостями тієї операційної системи, під управлінням якої ця програма виконуватиметься;

– правилами роботи вибраного компілятора – різні компілятори ставлять різні вимоги до початкового тексту програми.

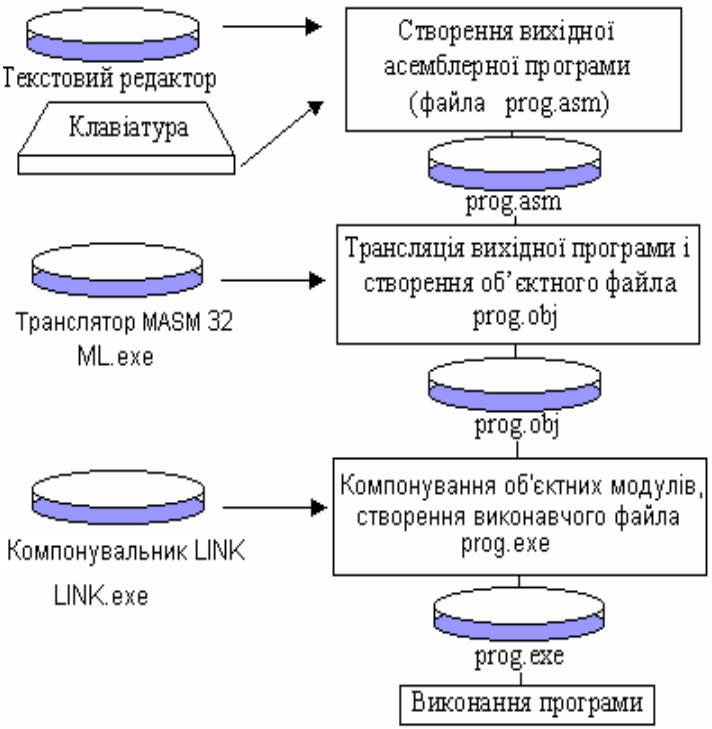


Рис. 1.1. Схема асемблювання, компонування і виконання програми

MASM і TASM підтримують ***спрощену сегментацію***. Суть такої сегментації в тому, що директиви .CODE і .DATA можуть з'являтися в тексті програми кілька разів. Транслятор потім збирає код і дані разом. Основною метою такого підходу є можливість наблизити в тексті програми дані до тих рядків, де вони використовуються.

Основні опції компонування наведені в табл. 1.1.

Таблиця 1.1

/DEBUG	Налагодження (при використанні налагоджувача)
/DEBUGTYPE:CV|COFF	Тип налагодження: codeview / coff (вибирає вихідний формат налагоджувальної інформації. Налагоджувачі SoftIce і Visual C++ можуть обробляти CV (codeview))
/DEF:ім'я_файлу	DEF файл (вказує файл визначення (.def). Використовується з dll для функцій, що експортуються)
/DLL	DLL (вихідний файл з розширенням .DLL)
/LIBPATH:path	Шлях до бібліотек (указує шлях до файлів бібліотек (*.lib).
/I< ім'я >	Встановлює шлях для inc-файлів
/OUT: ім'я_файлу	Out:імя_файлу (для зміни імені вихідного файлу)
/SUBSYSTEM:{...}	Підсистема (вибір ОС: NATIVE|WINDOWS|CONSOLE|WINDOWSCE|POSIX)

1.4. Налагодження програми

Будь-яка програма потребує налагодження (виправлення помилок). Сучасні налагоджувальні програми для асемблера платформи x86, наприклад програми Dbg_x86 (http://www.microsoft.com/whdc/devtools/debugging), SoftICE (яка вже не підтримується, але за звичкою ще використовується), Emu8086 v.4.04, OllyDbg 1.10 (**http://cracklab.ru**/), OllyDbg 2 (**http://www.ollydbg.de/**), Syser (www.syser.com), ImDbg.v1.50 (Immunity Debugger на http://cracklab.ru/) дозволяють у процесі виконання програми контролювати значення регістрів загального призначення чи змінних і змінювати їх.

За допомогою налагоджувачів можна, наприклад, переглядати вміст різних ділянок пам'яті, виконувати програму крок за кроком, змінювати програмний код, зберігати зміни в .exe-файлі та ін. Для налагодження професійних **асемблерних** програм, які застосовують останні версії команд паралельної обробки дійсних чисел або нові API-функції, які постійно з'являються, використовують середовище **Visual Studio**, порядок роботи в якому буде наведено далі.

При написанні програм **обов'язково** використовувати довідник MSDN (http://msdn.microsoft.com/ru-ru/library).

1.5. Редактори тексту

Як редактор тексту при написанні та налагодженні невеликих навчальних програм простіше всього використовують офісний блокнот, в якому пишуть текст. Виконують такі програми з використанням командного bat-файлу. ***Як середовище виконання програми на начатковому етапі використовують частіше саме середовище masm32*** **(програму masm32 Editor**) або командний рядок файлового менеджера FAR.

У даний час застосування Windows готуються, як правило, за допомогою інтегрованих середовищ розробки (**Integrated Development Environment, IDE**), які забезпечують весь цикл розробки додатка (підготовку початкового тексту, компіляцію, налагоджування та пробного виконання) у повноекранному інтерактивному режимі в середовищі Windows. При цьому сама IDE широко використовує стандартні засоби графічного інтерфейсу Windows – меню, діалогові вікна, інструментальні кнопки та ін. Для невеликих програм можна використовувати блокнот Notepad++ (http://notepad-plus.sourceforge.net/) або UltraEdit (www.ultraedit.com). Всі такі редактори підтримують підсвічування синтаксису, можливість запуску зовнішніх застосувань.

Для професійного використання можна рекомендувати IDE-редактори: RadASM *(***Ketil Olsen, http://radasm.cherrytree.at),** Negatory Assembly Studio (www.negatory.com/asmstudio/), Visual SlickEdit (www.slickedit.com), **WinAsm** (**Antonis Kyprianou,** http://winasm. how.to/), Source Insight 3.5 (Source Dynamics, Inc., www.sourceinsight. com) та інші. **Найбільш повними можливостями з асемблювання програм володіє середовище Visual Studio останніх версій, тому що підтримує всі нові API-функції та нові асемблерні команди.**

При самостійному опрацьовуванні навчального матеріалу можна використати програми з сайта ресурсу **wasm.ru.**

2. НАЛАГОДЖУВАЧ OLLYDBG

Для ***виправлення команд в exe-файлі*** у налагоджувачі OllyDbg необхідно виконати таке:

– скопіювати програму в нове вікно для корекції. Для цього натиснути праву кнопку миші у вікні дизасемблера і вибрати у нижній частині спливаючого меню пункт Copy to execution / Selection. У результаті весь дизасембльований модуль разом з виправленими командами буде скопійований у нове вікно;

– вказати імя нового файлу. Для цього клацнути правою кнопкою миші по новому вікну і вибрати пункт Save file та ввести ім'я нового exe-файлу;

– виконати необхідні зміни в програмі. Для цього підвести курсор до команди, яку необхідно змінити, клацнути на ній, натиснути на праву кнопку миші та обрати пункт Assemble;

– зберегти змінений файл. Для цього натиснути на праву кнопку миші та обрати Copy/To file.

ТОЧКИ ЗУПИНКИ

Звичайні точки зупинки

Для того щоб поставити *звичайні точки зупинки* (ordinary breakpoints), необхідно у вікні дизасемблера використовувати клавішу <F2> або подвійне клацання миші в другій колонці вікна коду. У цьому випадку команда забарвлюється в червоний колір. Цей вид зупинки необхідний для того, щоб у точці зупинки можна було б перевірити стан регістрів, змінних, стека. Вторинне натиснення на клавішу <F2> в точці зупинки або подвійне клацання миші видаляють точку зупинки.

Умовні точки зупинки

Умовні точки зупинки (conditional breakpoints) необхідні для того, щоб здійснити зупинку, якщо виконується необхідна умова. Умовні точки зупинки встановлюються натисненням комбінації клавіш <Shift>+<F2>. При цьому з'являється вікно з комбінованим списком, куди можна занести точку зупинки, наприклад, з такою умовою:

- EAX == 5 – зупинка на відміченій команді (перед її виконанням) буде здійснена, якщо вміст регістра EAX дорівнюватиме 5;
- EAX = 0 AND ECX > 6 – зупинка на відміченій команді буде здійснена, якщо вміст регістра EAX дорівнюватиме нулю та вміст регістра ECX буде більше 6;

- [403050] = 1234 – зупинка, якщо вміст елемента пам’яті дорівнює 1234.

Умовні точки зупинки із записом у журнал

Умовні точки зупинки із записом у журнал (conditional logging breakpoint) є розширенням умовних точок зупинки, яке встановлюється натисненням комбінації клавіш <Shift>+<F4>. Для розгляду вмісту журналу необхідно скористатися комбінацією клавіш <Alt>+<L> або вибрати з меню пункт **View** | **Log.**

Точка зупинки на повідомлення Windows

Для встановлення зупинки на повідомлення Windows необхідно натиснути комбінацію клавіш <Ctrl>+<F8>. Для того щоб вийти на функцію вікна, треба викликати список створених застосувань вікон за допомогою пункту меню **View | Windows.** При роботі з віконними функціями слід ефективніше використовувати точку зупинки на повідомлення. Для цього клацнемо по вікну, яке з’явилося, і виберемо з контекстного меню пункт Message breakpoint on Classproc. У вікні, що з’явилося, можна встановити параметри точки зупинки, а саме:

– вибрати повідомлення з випадного списку;

– визначити перелік відстежуваних вікон на предмет надходження даного повідомлення;

– визначити лічильник спрацьовувань точки зупинки;

– встановити чи зупинити виконання програми;

– встановити чи провести запис у журнал.

Точка зупинки на функції імпорту

Список імен, що імпортуються з налагоджуваним модулем, можна отримати за допомогою натиснення комбінації клавіш <Ctrl>+<N>. Далі, клацнувши правою кнопкою миші по вікну, можна встановити:

– точку зупинки на виклик функції, що імпортується (команда **Toggle breakpoint on import);**

– умовну точку зупинки на виклик функції, що імпортується (команда **Conditional breakpoint on import);**

– умовну точку зупинки на імпорт із записом у журнал (команда **Conditional log breakpoint on import);**

– точки зупинки на всі посилання на дане ім’я (команда **Set breakpoint on every reference);**

– точки зупинки із записом у журнал на всі посилання на дане ім’я (команда **Set log breakpoint on every reference)** або видалити всі точки зупинки (команда **Remove all breakpoints).**

Точка зупинки на область пам'яті

Налагоджувач Ollydbg дозволяє встановити одну точку зупинки на область пам'яті. Вибираємо вікно дизасемблера або вікно даних (dump). Далі використовуємо контекстне меню і вибираємо пункт **Breakpoint | Memory on access** (на доступ до пам'яті) або **Breakpoint | Memory on write** (на запис у пам'ять). Після цього точка зупинки готова до використання. Видалити точку зупинки на область пам'яті можна з контекстного меню **Breakpoint | Remove memory breakpoint.**

Точка зупинки у вікні Memory

Для цієї точки зупинки використовується контекстне меню, що з'являється за допомогою клацання правою кнопкою миші і вибору пункту **Set memory breakpoint on access** (встановити точку зупинки на доступ до пам'яті) або **Set memory breakpoint on write** (встановити точку зупинки на запис у пам'ять). Видалити точку зупинки можна з того контекстного меню командою **Remove memory breakpoint.**

Апаратні точки зупинки

Зазвичай для точки зупинки використовують стандартний вектор переривання int3. У мікропроцесора Intel Pentium є чотири налагоджувальні регістри: DR0 – DR3. Ці регістри можуть містити чотири контрольні точки. Як тільки адреса, яку використовує команда, дорівнює адресі в одному з вказаних регістрів, то тоді генерується виключення, що перехоплюється налагоджувачем. Встановити апаратну точку зупинки можна з вікна дизасемблера за допомогою пункту **Breakpoint | Hardware on execution** контекстного меню або у вікні даних за допомогою пунктів **Breakpoint | Hardware on access** або **Breakpoint | Hardware on access.** Видалити апаратні точки зупинки можна за допомогою контекстного меню **Breakpoint | Remove hardware breakpoints.**

Пошук інформації

Налагоджувач Ollydbg дозволяє ефективно шукати різну інформацію. За командою від натиснення комбінації клавіш <Ctrl>+<B> з'являється вікно пошуку, де можливо визначити рядок, який розшукуватиметься. Рядок для пошуку можна вводити у вигляді послідовності символів, байтів, символів у кодуванні Unicode.

Для пошуку команд використовуються комбінації клавіш <Ctrl>+<F> – для одиночної команди і <Ctrl>+<S> – для послідовності команд.

Натиснення комбінації клавіш <Ctrl>+<L> повторює останній зроблений пошук.

3. ЛАБОРАТОРНІ РОБОТИ

3.1. Лабораторна робота 1
ПОДАННЯ ДАНИХ

Мета заняття

1. Виконати дослідження переводу чисел з десяткової в двійкову систему числення. Дати їх внутрішнє (машинне) подання відповідно до діапазону в знакових і беззнакових форматах типів Shortlnt (signed char), Byte (unsigned char), Integer (int), Word (unsigned int). Машинне подання даних повинно бути наведено в двійковій і шістнадцятковій системах числення.

2. Виконати дослідження переведення чисел з десяткової в двійкову систему числення. Дати їх внутрішнє (машинне) подання у форматах типів Single (float), Double (double), Extended (long double). Машинне подання даних повинно бути наведено в двійковій і шістнадцятковій системах числення.

Вхідний контроль знань

1. Перевести числа вашого дня, місяця та року народження у двійкову, вісімкову та шістнадцяткову системи числення.

2. Перевести число DD.MM у двійкову та шістнадцяткову системи числення для 32-розрядного формату дійсного числа, де DD – день вашого народження, MM – місяць вашого народження.

Завдання 1. Внутрішнє подання цілочислових даних у IBM PC:

– вибрати для свого варіанта цілі числа;

– перевести їх з десяткової у двійкову (або шістнадцяткову) систему числення;

– одержати їх внутрішнє подання;

– написати програму опису цих чисел мовою асемблера й одержати лістинг програми;

– перевірити правильність своїх викладок.

Завдання 2. Внутрішнє подання дійсних даних у IBM PC:

– обчислити для свого варіанта дійсні числа;

– перевести їх з десяткової у двійкову систему числення і зробити нормалізацію чисел;

– одержати їх внутрішнє подання (числа вказує викладач);

– написати програму опису цих чисел мовою асемблера й одержати лістинг програми;

– перевірити правильність своїх викладок.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Протокол переведення всіх заданих чисел з десяткової у двійкову і шістнадцяткову системи числення.
3. Лістинг програм.
4. Висновки за результатами роботи.

Порядок виконання роботи

Для виконання ***завдання 1*** необхідно вибрати свій варіант із табл. 3.1.1 та записати тільки число *X* зі знаками.

Таблиця 3.1.1 – Варіанти завдань

№	Завдання		№	Завдання		№	Завдання	
	±*X*	±*Y*		±*X*	±*Y*		±*X*	±*Y*
1	200	20	11	900	70	21	1400	120
2	250	25	12	950	75	22	1450	125
3	300	30	13	1000	80	23	1500	130
4	350	35	14	1050	85	24	1550	135
5	400	40	15	1100	90	25	1600	140
6	450	45	16	1150	95	26	1650	145
7	500	50	17	1200	100	27	1700	150
8	600	55	18	1250	105	28	1750	155
9	700	60	19	1300	110	29	1800	160
10	800	65	20	1350	115	30	2000	170

Наприклад, для 1-го варіанта цілі числа будуть такі:

+ 200 – 200

Написати програму з використанням всіх директив визначення даних. При поданні даних **необхідно враховувати те, що не всі числа можуть вміститися в обрану розрядну сітку.** У такому випадку такий формат даних використовувати непотрібно. Правильність одержаних результатів необхідно проконтролювати. Як приклад напишемо програму з ім'ям Lab1.asm (лістинг 3.1.1).

Лістинг 3.1.1. Програма подання цілих чисел:

```
.386            ; директива визначення команд мікропроцесора
 .MODEL flat            ; завдання лінійної моделі пам'яті
```

```
        .DATA                   ; початок сегмента даних
; цілі числа в пам'яті розміром у байт (8 розрядів)
        db 80 ;
        db -80 ;
        byte 80 ;
        byte -80 ;
        sbyte 80 ;
        sbyte -80 ;
; цілі числа в пам'яті розміром у слово (16 розрядів)
        dw 80 ;
        dw -80 ;
        word 80 ;
        word -80 ;
        sword 80 ;
        sword -80 ;
; цілі числа в пам'яті розміром у подвійне слово (32 розряди)
        dd 80 ;
        dd -80 ;
        dword 80 ;
        dword -80 ;
        sdword 80 ;
        sdword -80 ;
; цілі числа в пам'яті розміром у 48 розрядів
        fword 80 ;
        fword -80 ;
; цілі числа в пам'яті розміром у 64 розряди
        dq 80 ;
        dq -80 ;
        qword 80 ;
        qword -80 ;
; цілі числа в пам'яті розміром у 80 розрядів
        dt 80 ;
        dt -80 ;
        tbyte 80 ;
        tbyte -80 ;
    .CODE                       ; початок сегмента команд
    _start:                     ; мітка початку основного тіла програми
            ret                 ; вихід із ОС Windows
            end _start          ; закінчення програми з ім'ям _start
```

Для одержання лістингу програми в асемблері MASM32 (опція Fl – **fil**e) необхідно викликати файловий менеджер FAR, відкрити

C:\masm32\bin

та поставити курсор на файл ml.exe. Натиснути Ctrl + <Enter>. У результаті цього в командний рядок буде скопійовано:

C:\masm32\bin> ml.exe

Потім необхідно добрати додаткові опції компілятора та натиснути <Enter>:

Ml.exe /c /coff /Fl Lab1.asm <Enter>

Командний рядок у програмі FAR наведено на рис. 3.1.1.

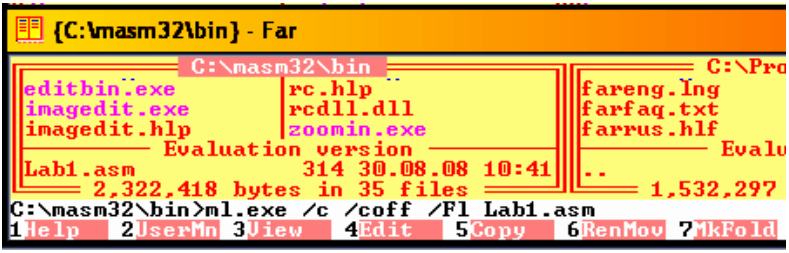


Рис. 3.1.1. Командний рядок у програмі FAR

У результаті таких дій буде створено файл **Lab1.lst.**

Опції компілятора MASM, які використовуються в цьому командному рядку, означають:

c – компілювати без компонування;

coff – генерувати об'єктний файл у форматі COFF;

Fl (file) – генерувати лістинг.

Файловий менеджер FAR зручний тим, що дозволяє контролювати помилки, які виникли при трансляції. Для видимості результату трансляції необхідно натиснути клавіші Ctrl + F1 або Ctrl + F2. Повторне натиснення цих клавіш поверне зображення вікон файлового менеджера в початковий стан.

Лістинг програми Lab1.**lst** може мати такий вигляд:

```
Microsoft (R) Macro Assembler Version 6.14.8444    10/06/10 16:42:36
Lab1.asm                                                Page 1 - 1
        .386    ; директива визначення команд мікропроцесора
       .MODEL flat ; задання лінійної моделі пам'яті
 00000000      .DATA ; початок сегмента даних
                     ; цілі числа в пам'яті розміром у байт (8 розрядів)
 00000000  50        db 80 ;
 00000001  B0        db -80 ;
 00000002  50        byte 80 ;
 00000003  B0        byte -80 ;
 00000004  50        sbyte 80 ;
```

```
00000005  B0          sbyte -80 ;
                    ; цілі числа в пам'яті розміром у слово (16 розрядів)
00000006  0050              dw 80 ;
00000008  FFB0              dw -80 ;
0000000A  0050              word 80 ;
0000000C  FFB0              word -80 ;
0000000E  0050              sword 80 ;
00000010  FFB0              sword -80 ;
            ; цілі числа в пам'яті розміром у подвійне слово (32 розряди)
00000012  00000050                  dd 80 ;
00000016  FFFFFFB0          dd -80 ;
0000001A  00000050          dword 80 ;
0000001E  FFFFFFB0          dword -80 ;
00000022  00000050                  sdword 80 ;
00000026  FFFFFFB0          sdword -80 ;
                            ; цілі числа в пам'яті розміром у 48 розрядів
0000002A  000000000050              fword 80 ;
00000030  FFFFFFFFFFB0              fword -80 ;
                             ; цілі числа в пам'яті розміром у 64 розряди
00000036  0000000000000050          dq 80 ;
0000003E  FFFFFFFFFFFFFFB0          dq -80 ;
00000046  0000000000000050          qword 80 ;
0000004E  FFFFFFFFFFFFFFB0          qword -80 ;
                            ; цілі числа в пам'яті розміром у 80 розрядів
00000056  00000000000000000050              dt 80 ;
00000060  FFFFFFFFFFFFFFFFFFB0                  dt -80 ;
0000006A  00000000000000000050              tbyte 80 ;
00000074  FFFFFFFFFFFFFFFFFFB0                  tbyte -80 ;
00000000            .CODE           ; початок сегмента команд
00000000            _start:  ; мітка початку основного тіла програми
00000000  C3          ret           ; вихід із ОС Windows
                      end _start ;

Microsoft (R) Macro Assembler Version 6.14.8444   09/03/08 13:44:50
    Lab1.asm                                            Symbols 2 - 1
    Segments and Groups:
            N a m e             Size     Length   Align   Combine Class
    FLAT . . . . . . . . . .GROUP
    _DATA  . . . . . . . .32 Bit   0000007E DWord        Public  'DATA'
    _TEXT  . . . . . . . . 32 Bit   00000001 DWord        Public  'CODE'
    Symbols:
            N a m e             Type        Value          Attr
    @CodeSize  . . . . . . . . . .  Number      00000000h
    @DataSize  . . . . . . . . . .  Number      00000000h
    @Interface . . . . . . . . . .  Number      00000000h
    @Model . . . . . . . . . . . .  Number      00000007h
```

```
@code . . . . . .    Text      _TEXT
@data . . . . . .    Text      FLAT
@fardata? . . . . .  Text      FLAT
@fardata . . . . . .  Text     FLAT
@stack . . . . . .   Text      FLAT
_start . . . . . .   L Near    00000000 _TEXT   Public
      0 Warnings
      0 Errors
```

Порядок розміщення цілих чисел у пам'яті необхідно також проконтролювати за допомогою налагоджувача OllyDbg. Для цього спочатку слід одержати .exe-файл за допомогою команди

link.exe /subsystem:console Lab1.obj.

Скопіювати вигляд налагоджувача у звіт можна, натиснувши клавіші PrtScr або Alt + PrtScr.

Потім треба викликати налагоджувач OllyDbg (http://cracklab.ru/_dl/new/ollydbg110xp.rar або www.ollydbg.de/) та завантажити одержаний файл Lab1.exe (рис. 3.1.2).

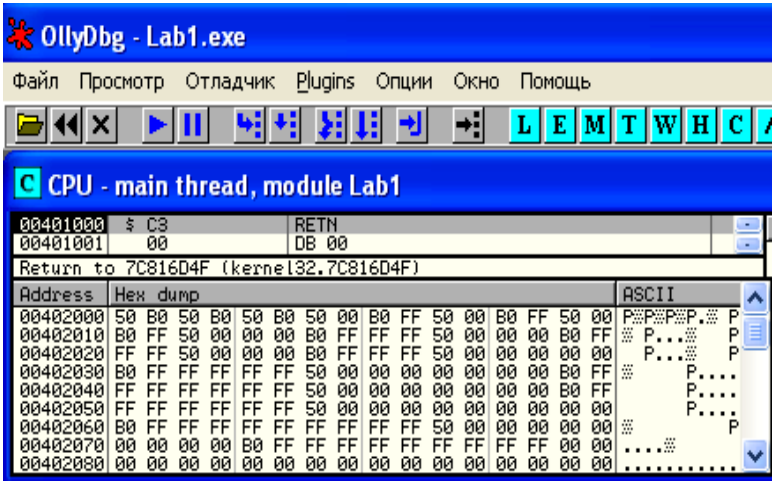


Рис. 3.1.2. Розміщення цілих чисел у налагоджувачі OllyDbg

На завершення слід проаналізувати порядок розміщення даних у пам'яті комп'ютера та їх відповідність до адрес.

Для виконання ***завдання 2*** необхідно згідно з варіантом із табл. 3.1 вибрати числа:

115.	3200.	115.3200
–115.	–3200.	–115.3200
0.115	0.32	
–0.115	–0.32	

Перевести одержані числа в шістнадцяткову систему числення.

Правильність одержаних результатів необхідно проконтролювати. Дійсні числа у записі повинні мати точку у форматі числа.

Для цього напишемо програму з ім'ям, наприклад Lab1_2.asm (лістинг 3.1.2).

Лістинг 3.1.2. Програма подання дійсних чисел:

```
.386                    ; директива визначення типу мікропроцесора
.model flat             ; задання лінійної моделі пам'яті
.data                   ; директива визначення даних
; дійсні числа в пам'яті розміром у 32 розряди
dd 115.
dd -115.
dd 0.115
dd -0.115
dd 3200.
dd -3200.
dd 0.32
dd -0.32
dd 115.3200
dd -115.3200
real4 115.
real4 -115.
; дійсні числа в пам'яті розміром у 64 розряди
dq 115.
dq -115.
dq 0.115
dq -0.115
dq 3200.
dq -3200.
dq 0.32
dq -0.32
dq 115.3200
dq -115.3200
real8 115.
real8 -115.
; дійсні числа в пам'яті розміром у 80 розрядів
dT 115.
```

```
        dT -115.
        dT 0.115
        dT -0.115
        dT 3200.
        dT -3200.
        dT 0.32
        dT -0.32
        dT 115.3200
        dT -115.3200
        real10 115.
        real10 -115.
.code                   ; директива початку коду програми
 _start:                ; мітка початку програми з ім'ям start
   ret                          ; повернення управління ОС
    end _start                  ; закінчення програми з ім'ям _start
```

Як відомо, для одержання лістингу програми необхідно викликати файловий менеджер FAR та вже з нього відкомпілювати програму за допомогою компілятора з мови асемблера MASM32, набравши в командному рядку команду

ml.exe /c /coff /Fl Lab1_2.asm <Enter>

Сторінка 1 лістингу програми Lab1_2.asm буде мати такий вигляд:

```
Microsoft (R) Macro Assembler Version 6.14.8444  09/03/10 15:40:46
Lab1_2.asm                                              Page 1 - 1
                .386        ; директива визначення типу мікропроцесора
                .model flat             ; задання лінійної моделі пам'яті
00000000        .data               ; директива визначення даних
                            ; дійсні числа в пам'яті розміром у 32 розряди
00000000  42E60000                  dd 115.
00000004  C2E60000                  dd -115.
00000008  3DEB851F                  dd 0.115
0000000C  BDEB851F                  dd -0.115
00000010  45480000                          dd 3200.
00000014  C5480000                  dd -3200.
00000018  3EA3D70A                  dd 0.32
0000001C  BEA3D70A                  dd -0.32
00000020  42E6A3D7                  dd 115.3200
00000024  C2E6A3D7                  dd -115.3200
00000028  42E60000                  real4 115.
0000002C  C2E60000                  real4 -115.
                            ; дійсні числа в пам'яті розміром у 64 розряди
00000030  405CC00000000000          dq 115.
00000038  C05CC00000000000          dq -115.
00000040  3FBD70A3D70A3D71          dq 0.115
00000048  BFBD70A3D70A3D71          dq -0.115
```

```
00000050  40A9000000000000          dq 3200.
00000058  C0A9000000000000          dq -3200.
00000060  3FD47AE147AE147B          dq 0.32
00000068  BFD47AE147AE147B          dq -0.32
00000070  405CD47AE147AE14          dq 115.3200
00000078  C05CD47AE147AE14          dq -115.3200
00000080  405CC00000000000          real8 115.
00000088  C05CC00000000000          real8 -115.
                    ; дійсні числа в пам'яті розміром у 80 розрядів
00000090  4005E600000000000000             dT 115.
0000009A  C005E600000000000000             dT -115.
000000A4  3FFBEB851EB851EB851F             dT 0.115
000000AE  BFFBEB851EB851EB851F             dT -0.115
000000B8  400AC800000000000000             dT 3200.
000000C2  C00AC800000000000000             dT -3200.
000000CC  3FFDA3D70A3D70A3D70A             dT 0.32
000000D6  BFFDA3D70A3D70A3D70A             dT -0.32
000000E0  4005E6A3D70A3D70A3D7             dT 115.3200
000000EA  C005E6A3D70A3D70A3D7                      dT -115.3200
000000F4  4005E600000000000000             real10 115.
000000FE  C005E600000000000000             real10 -115.
00000000            .code       ; директива початку коду програми
00000000            _start:    ; мітка початку програми з ім'ям start
00000000  C3        ret                    ; повернення управління ОС
                    end _start   ; закінчення програми з ім'ям start
```

Порядок розміщення дійсних чисел у пам'яті необхідно також проконтролювати за допомогою налагоджувача OllyDbg. Для цього треба одержати .exe-файл за допомогою команди

link.exe /subsystem:console Lab1_2.obj,

або, що й ще зручніше, – створити bat-файл, та вже при виконанні цього файлу автоматично отримати зразу два файли: .obj та .exe. Такий bat-файл може мати вміст:

```
ml /c /coff /Fl   "Lab1_2.asm"
link /SUBSYSTEM:CONSOLE  "Lab1_2.obj"
```

Потім слід викликати налагоджувач OllyDbg та завантажити одержаний файл Lab1_2.exe (рис. 3.1.3).

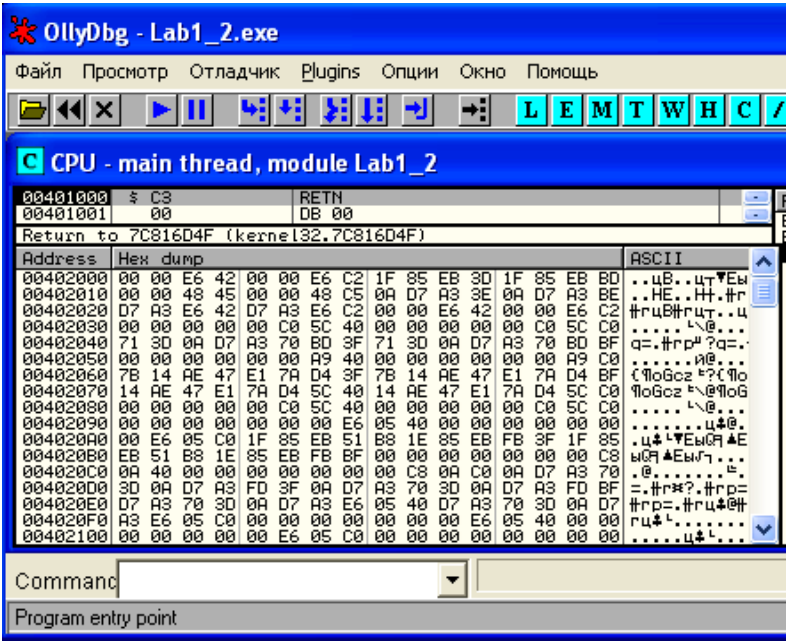


Рис. 3.1.3. Розміщення дійсних чисел у налагоджувачі OllyDbg

На закінчення, треба проаналізувати порядок розміщення даних у пам’яті комп’ютера та їх відповідність до адрес.

Довідковий матеріал

Формати даних для подання двійкових чисел

У форматі з фіксованою комою числа подаються в додаткових кодах. Знак числа кодується в старшому біті (0 – “+”, 1 – “-”).

Наприклад, для подання числа A = $[-1990]_{10}$ скористаємося переведенням його в додатковий код і форматом для цілих чисел із знаком, причому доповнимо число A2 до п’ятнадцяти розрядів:

A = $[–1990]_{10}$ = $[–11111000110]_{2пк}$ = $[–000\ 0111\ 1100\ 0110]_{2пк}$ =

= $[1.111\ 1000\ 0011\ 1010]_{2дк}$.

Формати даних для подання дійсних чисел

Числа з плаваючою комою призначені для подання дійсних чисел і можуть бути типу *float* (32 розряди), *double* (64 розряди) або *long double* (*extended)* (80 розрядів).

У masm32 довідковий матеріал відносно форматів дійсних чисел наведено за адресою masm32\tutorial\fputute\fpuchap2.htm.

Внутрішнє (машинне) подання цих чисел достатньо складне:

S	Характеристика	Нормалізована мантиса

Біт *S* – знак числа.

Характеристика = Зміщення ± Порядок.

Зміщення – число, що дорівнює ПОЛОВИНІ максимально можливого, яке може поміститься в полі "Характеристика". Таким чином, економиться місце і час, оскільки не потрібно виділяти розряд **для знаку порядку** і робити додатковий код для негативних порядків – він завжди позитивний.

Розмір простору, який ПК використовує для зберігання характеристики і мантиси, встановлений стандартом IEEE 724-1985 і підтримується всією сучасною комп'ютерною архітектурою.

Нормалізованим двійковим числом називається двійкове число, яке починається з одиниці і має такий вигляд:

$$X = \pm 1.m_2 \cdot 2^p,$$

де m_2 – двійкова мантиса числа; p – порядок двійкового числа.

Формат короткого (32-розрядного) дійсного числа:

1 – прихований розряд

v

S	Характеристика – **8 розрядів**			Нормалізована мантиса – **23 розряди**				
31	30	. . .	23	22	21	. . .	1	0

Зміщення складає $\mathbf{7F}_{16} = 127_{10}$.

Загальний формат 32-розрядного дійсного числа дорівнює

S (1 біт) + E (8 біт) + M (23 біти),

де: S – знак числа; E – зміщена характеристика (порядок + $7F_{16}$); M – мантиса.

Задача 1. Подати число $A = 0.25_{10}$ у форматі REAL4 (DD) – 32-розрядного числа з плаваючою точкою.

Розв'язання

Мантиса: $A = 0.25_{10} = 0.01_2 = 1.0_2 \cdot 2^{-2.}$

Характеристика: $7F - 2 = 7D_{16} = 01111101_2$.

$[0.25_{10}]_{пл} = 0\ 01111101\ 00000000000000000000000_2 = 3E800000_{16}$;

$[-0.25_{10}]_{пл} = 1\ 01111101\ 00000000000000000000000_2 = BE800000_{16}$;

Задача 2. Подати число A = $115._{10}$ у форматі REAL4 (DD) – 32-розрядного числа з плаваючою точкою.

Розв'язання

Мантиса: $A = 115._{10} = 1110011_2 = 1.110011_2 \cdot 2^6$.

Характеристика: 7F + 6 = 85h = 1000 0101.

Число: A = $115._{10}$ = 0 1000 0101 1100 11000000000000000000b = 42E6 0000h.

Задача 3. Подати число A = -115.32_{10} у форматі REAL4 – 32-розрядного числа з плаваючою точкою.

Розв'язання

Мантиса: $A = -115.32_{10} = 1.110011.01010001111010111_2 \cdot 2^6$.

Характеристика: 7F + 6 = 85h = 1000 0101b.

Число: $A = 115.32_{10}$ = **1** 1000 0101 110 0110 1010 0011 1101 0111 = = C2E6 A3D7h.

Формат довгого (64-розрядного) дійсного числа:

1 – прихований розряд

v

S	Характеристика			Нормалізована мантиса				
63	62	. . .	52	51	50	. . .	1	0

Зміщення складає **3FF**$_{16} = 1023_{10}$.

Задача 4. Подати число A = -115.32_{10} у форматі REAL8 – 64-розрядного числа з плаваючою точкою.

Розв'язання

Мантиса: $A = -115.32_{10} = 1.1100\ 11.01\ 0100\ 0111\ 1010\ 1110\ 0001\ 0100\ 0111\ 1010\ 1110\ 0001\ 0100_2 \cdot 2^6$.

Характеристика: 3FF + 6 = 405h = 100 0000 0101.

Число: $A = -115.32_{10}$ =

= **1** 100 0000 0101 110 0 110 1 0100 0111 1010 1110 0001 0100 0111 1010 1110 0001 0100 = C05C D47A E147 AE14h.

Формат розширеного (80-розрядного) дійсного числа:

S	Характеристика			Нормалізована мантиса				
79	78	. . .	64	63	62	. . .	1	0

Особливість останнього формату полягає у тому, що мантиса НЕ МАЄ прихованого розряду.

Зміщення становить **3FFF**$_{16}$ = 16383$_{10}$.

Загальний формат 64-розрядного дійсного числа дорівнює

S (1 біт) + E (15 біт) + M (64 біти),

де S – знак числа; E – зміщена характеристика (порядок + 3FFF$_{16}$); M – мантиса.

Задача 5. Подати число A = 115.32$_{10}$ у форматі REAL10 – 80-розрядного числа з плаваючою точкою.

Розв'язання

Мантиса: A = 115.32$_{10}$ = 0.110011.01010001111010111$_2 \cdot 2^6$.

Характеристика: 3FFF + 6 = 4005h = 100 0000 0000 0101.

Число: A = 115.32$_{10}$ =

= **0** 100 0000 0000 0101 **1**110 0110 1010 0011 1101 0111 0000 1010 0011 1101 0111 0000 1010 0011 1101 0111 = 4005 E6A3 D70A 3D70 A3D7h.

3.2. Лабораторна робота 2
ПРОГРАМУВАННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм, написаних мовою асемблера для програмування арифметичних виразів для МП платформи x86.

Вхідний контроль знань

Написати та налагодити програму згідно з одержаним варіантом задання. Результат записати у пам'ять.

1. Скласти день, місяць та рік свого народження.
2. Перемножити день, місяць та рік свого народження.
3. Поділити рік свого народження на місяць народження.
4. Виконати зсув дня свого народження ліворуч на 2 розряди.
5. Виконати зсув дня свого народження ліворуч на 4 розряди.
6. Виконати зсув року свого народження праворуч на 2 розряди.
7. Виконати операцію логічного додавання дня та року свого народження.
8. Виконати операцію логічного перемноження місяця та року свого народження.
9. Виконати операцію логічного додавання місяця та року свого народження.
10. Виконати операцію логічного порівняння місяця та року свого народження.

Постановка задачі

Згідно з номером у групі вибрати варіант та написати на асемблері програму обчислення одного з виразів:

1. $a + ab - e/c$;	7. $a + 11de - e/b$;	13. $(de + 21e)/(c - a)$;
2. $(a + 2d) - d/b$;	8. $(b + 12de)/(a - c)$;	14. $(a + 22e)b/(a + c)$;
3. $c + 3ad - e/b$;	9. $ae + 13b - d/c$;	15. $a + 23ac - e/b$;
4. $(a + 4e)b - d/b$;	10. $(ab + 14d)/(b - d)$;	16. $(a + 24b) - d/c$;
5. $a + 5de - e/b$;	11. $(de + 15d)/(c + a)$;	17. $a + 25db - e/b$;
6. $(b + 6de)/(a - c)$;	12. $(a + 16e)b/(e - c)$;	18. $(b + 26de)/(a - b)$;

19. *ae* + 7*b* – *d*/*c*; 23. *a* + 17*ab* – *d*/*b*; 27. *ae* + 27*b* – *d*/*a*;
20. *(ab* + 8*d)/(b – d)*; 24. *(a* + 18*e)* + *d*/*a*; 28. *(ab* + 28*d)/(b – a)*;
21. *(de* + 9*e)/(c – a)*; 25. *c* + 19*bd* – *d*/*b*; 29. *(de* + 29*a)/(c – a)*;
22. *(a* + 10*e)b/(a + c)*, 26. *(a* + 20*e)a* – *d*/*b*; 30. *(a* + 30*d)e/(a – c)*;

де *a* – № у списку;

b – № навчальної групи;

c – *рр* – дві останні цифри року народження;

e – *рррр* – чотири цифри року народження;

d – *ддмм*:

дд – день народження;

мм – місяць народження.

Результат обчислення виразу зберегти в пам'яті. Навести значення та порядок розміщення даних у пам'яті.

Треба пам'ятати, що при виконанні операції ***ділення*** *в регістрі EDX зберігається не дрібна частина результату, а залишок від цілочислового ділення, що не є таким самим.*

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Лістинг програми з детальним коментарем та описом роботи.
3. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.
4. Короткий опис виконання програми.
5. Висновки за результатами роботи.

Приклад 3.2.1. Написати на асемблері програму обчислення виразу

(e + d – ed)/a,

де *a* – № у списку;

b – № навчальної групи;

c – *рр* – дві останні цифри року народження;

e – *рррр* – чотири цифри року народження;

d – *ддмм*:

дд – день народження;

мм – місяць народження.

Результат обчислення виразу зберегти в пам'яті. Навести значення та порядок розміщення даних у пам'яті.

Приклад використання формату даних, збільшеного учетверо, наведено у лістингу 3.2.1. У цій програмі операція віднімання виконана у додаткових кодах. Тому **операція віднімання замінена на операцію додавання в додаткових кодах.** Результат обчислення виразу теж наведено у додаткових кодах.

Лістинг 3.2.1. Приклад використання формату даних, збільшеного учетверо:

```
        title CopyRight by Rysovaniy A. N.
        ; (e + d – ed)/a
        .686                ; директива визначення типу мікропроцесора
       .model flat          ; задання лінійної моделі пам'яті
     .data                  ; директива визначення даних
        a dd 6                      ;  a = 00000006h
        d dd 2112                   ;  d = 00000840h
        e dd 1989                   ;  e = 000007C5h
    rez1 dd 0                       ;  rez1 = 00000000h
    rez2 dd 0                       ;  rez2 = 00000000h

        .code       ; (e + d – ed)/a – початок сегмента команд
_start:             ; мітка початку програми з ім'ям _start
     ; e*d
        mov eax,d                       ; eax = 00000840h
        imul e                  ; edx = 00000000h, eax = 00401940h
     ; – e*d  (перетворення у додатковий код)
        not eax                 ; інверсія eax = FFBFE6BFh
        not edx                 ; інверсія edx = FFFFFFFFh
        inc eax                 ; eax = FFBFE6C0h
        adc edx,0               ; edx = FFFFFFFFh  (CF=0)
     ; + d – e*d
        add eax,d                       ; eax = FFBFEF00h
        adc edx,0                       ; edx = FFFFFFFFh  (CF=0)
     ;e + d – e*d
        add eax,e                       ; eax = FFBFF6C5h
        adc edx,0                       ; edx = FFFFFFFFh  (CF=0)
     ; (e+d-e*d)/a
        idiv a              ; edx = FFFFFFFDh,  eax = FFF553CCh
     mov rez1,eax           ; rez1 = FFF553CCh – частка
     mov rez2,edx           ; rez2 = FFFFFFFDh – залишок
        ret                 ; повернення управління ОС
        end _start      ; закінчення програми з ім'ям _start
```

Порядок розміщення даних у пам'яті та стан налагоджувача OllyDbg з програмою лістингу 3.2.1 наведено на рис. 3.2.1.

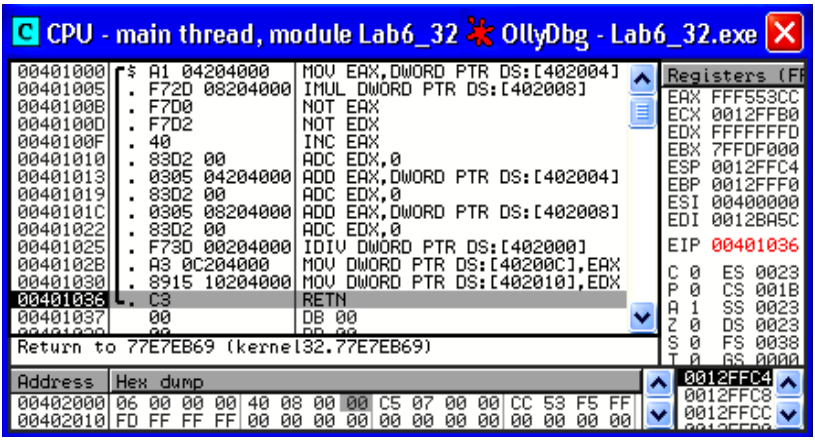


Рис. 3.2.1. Порядок розміщення даних у пам’яті та стан налагоджувача OllyDbg з програмою з лістингу 3.2.1

У зв’язку з тим, що значення виразу (e+d-e*d) при таких початкових даних повністю розміщується у регістрі eax та не “переходить” у регістр edx, програма стає достатньо простою. У протилежному разі потрібно виконувати спочатку ділення регістра edx, потім залишок пристиковувати до регістра eax та знову виконувати ділення (ділення у стовпчик).

Для одержання результату в **прямому** коді необхідно провести зворотне перетворення вмісту комірок пам’яті з додаткового коду.

Приклад 3.2.2. Написати на асемблері програму обчислення виразу: (*ab* – 9)/*b*.

Лістинг 3.2.2. Програма виконання прикладу 3.2.2:

```
    ; (ab – 9)/b
    .386          ; директива визначення команд мікропроцесора
  .model flat             ; задання лінійної моделі пам’яті
   .data                          ; початок сегмента даних
a db 8   ; запис у 8-розрядну комірку пам’яті номера варіанта
b db 28 ; запис у 8-розрядну комірку пам’яті номера групи
d dw 1706        ; запис у 16-розрядну комірку пам’яті
res1 db 0 ; резервування пам’яті для цілої частини результату
res2 db 0 ; резервування пам’яті для залишку результату
  .code                   ; директива початку сегмента команд
_start:                   ; мітка початку основного тіла програми
```

```
        mov al, a          ; al = a
        mul b              ; ax = al * b
         mov bx,d          ; bx = d
         not bx            ; інверсія bx
         add bx,1          ; додатковий код bx
        add ax, bx         ; ax = ax + b
        not ax             ; інверсія ax
        inc ax             ; прямий код в ax: (ab – d)
        mov dx,0           ; ініціалізація dx
        sub a,9            ; примусове встановлення знака
        idiv b             ; al - ціла частина результату, ah - залишок
        mov res1, al       ; res1 = al
        mov res2, ah       ; res2 = ah
        ret                ; повернення управління ОС
    end _start             ; закінчення програми
```

Порядок розміщення даних у пам'яті та стан налагоджувача OllyDbg програми наведено на рис. 3.2.2.

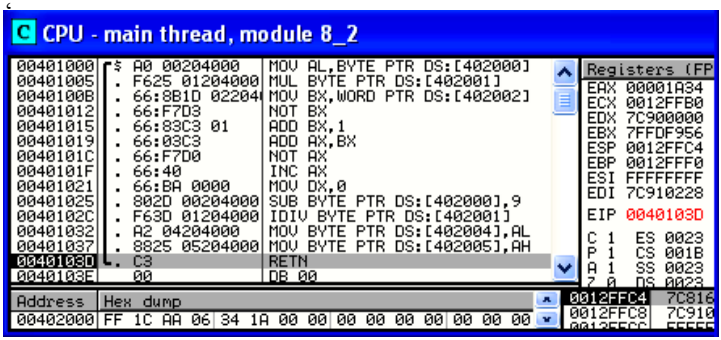


Рис. 3.2.2. Стан налагоджувача OllyDbg з програмою прикладу 3.2.2

Ціла частина результату (число 34h) зберігається в комірці пам'яті за адресою 402000, а залишок в – 402005.

Приклад 3.2.3. Написати на асемблері програму обчислення виразу $(a + b)c – d/c$, де $a = 5$, $b = 16$, $c = 81$, $d = 2209$.

Лістинг 3.2.3. Програма обчислення прикладу 5.4 з використанням змінних розміром у байт:

```
title CopyRight by Rysovaniy A. N.          rysov@rambler.ru
```

```
    .686                ; директива визначення типу мікропроцесора
.model flat             ; задання лінійної моделі пам'яті
.data                   ; директива визначення даних
_a db 5   ; запис у 8-розрядну комірку пам'яті з ім'ям _a числа 5
_b db 16                ; _b = 10h
_c db 81                ; _c = 51h
_d dw 2209  ; запис у 16-розрядну комірку пам'яті числа _d = 8A1
rez1 dw 0    ; резервування пам'яті для збереження змінної rez1
rez2 dw 0    ; резервування пам'яті для збереження змінної rez2
.code                   ; директива початку сегмента команд
_start:                 ; мітка початку програми з ім'ям _start
    mov al,_a           ; al = 05h
    add al,_b           ; al = 5h + 10h = 15h       ;  – (a + b)
    mul _c              ; ax = 15h × 51h = 06A5h ;   – (a + b)c
    push ax             ; стек = XXXX06A5h
    mov ax,_d           ; ax = 08A1h
    div _c              ; ah,al = ax/_c = 16 1Bh    ;   –  d/c
    movzx bx,ah                 ; bx = 0016h – залишок
    xor cx,cx           ; cx = 0
    sub cx,bx           ; cx = cx – bx = FFEAh – залишок
    mov rez1,cx                 ; збереження залишку
    pop cx              ; cx = 06A5h  – відновлення (a + b)c
    movzx ax,al  ; al = 001Bh – вирівнювання формату
    sbb cx,ax    ; cx = 06A5h – 001Bh – 1 = 0689h – ціла частина
    mov rez2,cx         ; збереження цілої частини результату
    ret                 ; повернення управління ОС
end _start              ; закінчення програми з ім'ям _start
```

Порядок розміщення даних у пам'яті та стан налагоджувача OllyDbg програми наведено на рис. 3.2.3.

Результатом виконання програми буде число 0689.FFEA, де 689h – ціла частина, а FFEAh – залишок (не дрібна частина) результату.

Для копіювання активного вікна програми, наприклад програми OllyDbg, треба натиснути Alt + PrtScr.

Збереження тимчасових результатів частини виразу (a + b)c виконується у стека , хоча для цього можна використовувати як один із вільних регістрів, так й комірку пам'яті.

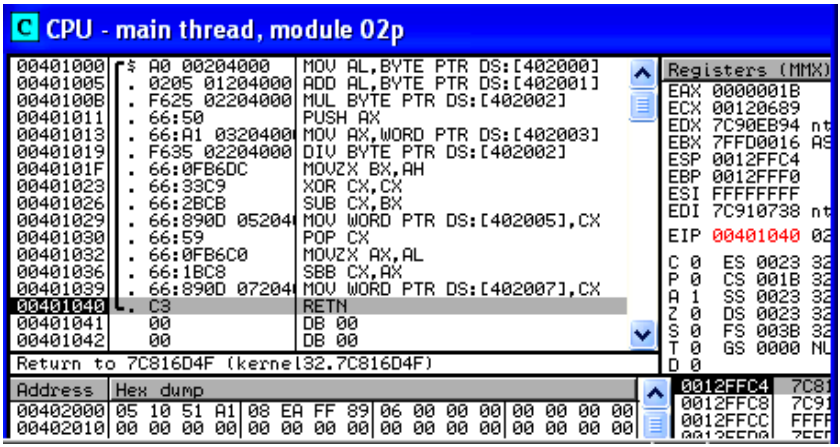


Рис. 3.2.3. Стан налагоджувача OllyDbg з програмою прикладу 3.2.3

Приклад 3.2.4. Написати на асемблері програму обчислення виразу e/3*c + 5*a*c, враховуючи можливе перенесення результату.

Лістинг 3.2.4. Програма обчислення прикладу 3.2.4 з використанням можливого перенесення результату:

```
        title Лебєдєва Олена, НТУ “ХПІ”, КІТ29Б
        ; e/3*c + 5*a*c = 1993/3*93 + 5*5*93
        .386            ; директива визначення типу мікропроцесора
        .model flat     ; задання лінійної моделі пам`яті
.data           ; директива визначення даних
 _e dd 1993
 _c dd 93
 _a dd 5
res1 dd ? ; визначення у пам`яті комірки під молодшу частину результату
res2 dd ?       ; комірка для збереження старшої частини результату
.code           ; директива початку сегмента команд
_start:         ; мітка початку основного тіла програми
    mov eax, _c         ; eax:=_c
    mov ebx, _a         ; ebx:=_a
    mul ebx       ; edx:eax=_c*_a (ст. частина в EDX, молодша – в EAX)
    mov ebx, 5          ; ebx:=5
    mul ebx             ; edx:eax:=_c*_a*5
    mov esi, edx ; збереження старшої частини
    mov edi, eax ; збереження молодшої частини
    mov eax, _c         ; eax:=_c
    mov ebx, 3          ; ebx:=3
```

```
    mul ebx              ; edx:eax:=_c*3
    mov ebx, eax    ; ebx:=_c*3, Збереження проміжного результату в EBX
    mov eax, _e          ; eax:=_e
    mov edx, 0           ; edx:=0, обнуління для подальшого ділення
    div ebx              ; edx:eax:=_e/(_c*3)(частка->EAX, залишок->EDX)
    add eax, edi         ; eax:=93*5*5 + 1993/(93*3)
    jc m1                ; перейти, якщо є перенесення
    jmp m2               ; безумовний перехід на збереження результату
m1:
    inc esi        ; додавання одиниці перенесення до старшої частини
m2:
    mov res1, eax ; збереження у пам`яті молодшої частини
    mov res2, esi  ; збереження у пам`яті старшої частини
    ret                ; повернення управління ОС
    end _start       ; закінчення програми з ім'ям   _start
```

Результат налагодження програми наведено на рис. 3.2.4.

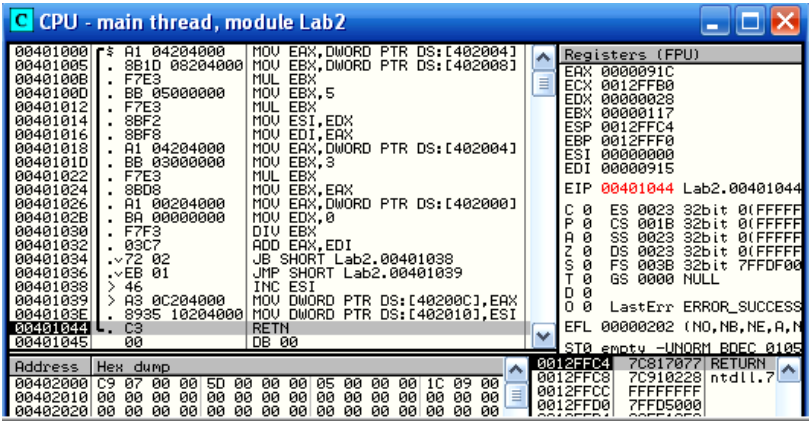


Рис. 3.2.4. Результат налагодження програми з лістингу 3.2.4

У зв`язку з тим, що результат множення 5*93 повністю розміщується у регістрі EAX та не переходить у регістр EDX (старша частина результату), то для спрощення програми у наступній дії не враховується. Значення 5*a*c (5*93*5) також повністю розміщується в EAX, однак для наочності значення регістру EDX зберігається в ESI (у нашому випадку 00000000), та використовується при подальшому обчисленні.

При виконанні команди **add eax, edi** у разі появи перенесення враховується одиниця переносу, додаючи її до старшої частини результату, яка зберігається у ESI командою **inc esi.**

3.3. Лабораторна робота 3
ПЕРЕДАЧА ПАРАМЕТРІВ ЧЕРЕЗ ТАБЛИЦЮ АДРЕС

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм, написаних мовою асемблера для програмування задач організації масивів для МП платформи x86.

Вхідний контроль знань

Написати та налагодити програму згідно з отриманим варіантом з обов'язковим урахуванням необхідних ознак виконання операції. Результат записати у пам'ять.

1. Виконати операцію логічного порівняння (команда TEST) числа та місяця свого народження. Якщо нульовий розряд результат операції дорівнює нулю, то додати ці числа.

2. Виконати операцію логічного перемноження (команда AND) місяця та року свого народження. Якщо другий розряд результату операції дорівнює одиниці, то додати ці числа.

3. Виконати зсув року свого народження праворуч на 3 розряди. Якщо перший розряд результату операції дорівнює одиниці, то виконати зсув праворуч на один розряд.

4. Скласти день, місяць та рік свого народження. Якщо третій розряд результату операції дорівнює нулю, то з року народження відняти місяць та день.

5. Перемножити два числа. Якщо нульовий розряд результату операції дорівнює нулю, то додати ці числа.

6. Поділити рік свого народження на місяць народження. Якщо перший розряд результату операції дорівнює нулю, то додати ці числа.

7. Виконати зсув дня свого дня народження ліворуч на 3 розряди. Якщо четвертий розряд результат операції дорівнює одиниці, то виконати зсув праворуч на 1 розряд.

8. Виконати операцію логічного додавання за модулем (команда XOR) місяця та року свого народження. Якщо нульовий розряд результату операції дорівнює нулю, то виконати зсув праворуч на 2 розряди.

9. Виконати зсув року свого народження ліворуч на 4 розряди. Якщо третій розряд результату операції дорівнює нулю, то додати числа дня та місяця свого народження.

10. Виконати операцію логічного додавання за модулем (команда XOR) місяця та дати свого народження. Якщо нульовий розряд результату операції дорівнює нулю, то додати ці числа.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати варіант завдання та написати на асемблері програму обчислення одного з виразів:

1. Задано масиви A і B по $N = 12$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai \le Bi$, то $Ci = Bi - Ai$; інакше – $Ci = Ai + Bi$.

2. Задано масив A з $N = 55$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \ge Ai \ge M$, де $L = 9$ та $M = 30$.

3. Задано масив A з $N = 45$ елементів. Навести алгоритм та програму формування масиву B з останніх 13 елементів масиву A, які дорівнюють нулю.

4. Задано масив A з $N = 55$ елементів. Навести алгоритм та програму формування масиву B з елементів масиву A, які задовольняють умову $Ai \le E$ при $E = 12$.

5. Задано масив A з $N = 70$ елементів. Навести алгоритм та програму визначення суми і кількості елементів масиву A, які задовольняють умову $Ai \le E$ при $E = -13$.

6. Задано масиви A і B по $N = 50$ елементів. Навести алгоритм та програму визначення кількості пар елементів, які задовольняють умову $Ai \le Bi$.

7. Задано масиви A і B по $N = 65$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai - Bi \le 0$, то $Cj = Ai$.

8. Задано масив A з $N = 50$ елементів. Навести алгоритм та програму визначення максимального з негативних елементів масиву A.

9. Задано масив A з $N = 50$ елементів. Структура масиву A така: $X1$, $Y1$; $X2$, $Y2$; … . Навести алгоритм та програму визначення кількості пар, для яких виконується умова $Xi < Yi$.

10. Задано масив A з $N = 28$ елементів. Навести алгоритм та програму формування масиву B з елементів масиву A, у яких біти 0, 2 та 5 мають нулі.

11. Задано масив A з $N = 40$ елементів. Навести алгоритм та програму визначення суми елементів масиву A, для яких біти 2 та 10 збігаються.

12. Задано масив A з $N = 80$ елементів. Структура масиву A така: $X1$, $Y1$; $X2$, $Y2$; … . Навести алгоритм та програму визначення кількості пар, для яких виконується умова $Xi > Yi$.

13. Задано масив A з $N = 20$ елементів. Навести алгоритм та програму визначення мінімального з позитивних елементів масиву A.

14. Задано масиви A і B по $N = 30$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai + Bi > 0$, то $Cj = Bi$.

15. Задано масиви A і B по $N = 20$ елементів. Навести алгоритм та програму визначення кількості пар елементів, які задовольняють умову $Ai > Bi$.

16. Задано масив A з $N = 40$ елементів. Навести алгоритм та програму визначення суми і кількості елементів масиву A, які задовольняють умову $Ai > E$ при $E = -10$.

17. Задано масив A з $N = 22$ елементів. Навести алгоритм та програму формування масиву B з елементів масиву A, які задовольняють умову $Ai > E$ при $E = 5$.

18. Задано масив A з $N = 30$ елементів. Навести алгоритм та програму формування масиву B з перших 8 позитивних елементів масиву A.

19. Задано масив A з $N = 20$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L < \text{Ai} \le M$, де $L = 2$ та $M = 10$.

20. Задано масиви A і B по $N = 11$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai > Bi$, то $Ci = Ai + Bi$; інакше – $Ci = Ai – Bi$.

21. Задано масив A з $N = 50$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L > Ai > M$, де $L = 8$ та $M = 28$.

22. Задано масиви A і B по $N = 10$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai < Bi$, то $Ci = Bi – Ai$; інакше – $Ci = Ai + Bi$.

Зміст звіту

1. Постановка задачі для отриманого варіанта.

2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.

3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій та з детальним коментарем і описом роботи.

4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.

5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.3.1. Проаналізувати масиви *A* та *B*, в кожному з яких по 10 елементів типу WORD. Необхідно визначити кількість елементів масиву *A*i, що задовольняють умову $A_i \le B_i$, та використати передачу параметрів через таблицю адрес. Таку програму наведено у лістингу 3.3.1.

Лістинг 3.3.1. Передача параметрів через таблицю адрес:

```
title CopyRight by Rysovaniy A. N. ; передача параметрів через табл. адрес
      .386                ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
      .data                                   ; директива визначення даних
       A1 DW 0,1,2,3,4,5,6,7,8,9      ; збереження в масиві пам'яті з ім'ям A1
                                        ; чисел, кожне у 16-розрядну комірку
       B1 DW 9,8,7,6,5,4,3,2,1,0
       N1 DW 10
       _TA DD 4 DUP (0)
.code                              ; директива початку коду програми
_start:                          ; мітка початку програми з ім'ям _start
      mov _TA,offset A1                ; завантажити адресу масиву A1
      mov _TA+4,offset B1              ; завантажити адресу масиву B1
      mov _TA+8,offset N1      ; завантажити адресу лічильника N1
      mov ebx,[_TA]            ; завантажити адресу початку таблиці адрес
    call prog1  ;
    ret                      ; повернення управління ОС
      prog1 proc
      mov cx,[ebx+40]                ; лічильник = 0Ah (10)
      movzx ecx,cx           ; заповнення старшої частини регістра нулями
      mov si,[ebx]                   ; вибрати 1-й елемент масиву A1
      mov di,[ebx+20]                ; вибрати 1-й елемент масиву B1
      xor edx,edx            ;
  m1: cmp si,di                      ; порівняння елементів масивів
       jl  m2                ; перейти, якщо менше
       jmp m3                ; перейти в протилежному випадку
  m2:  inc dx                ; підрахувати кількість елементів
  m3:  add si,2              ; підготовка наступного елемента масиву A1
       add di,2              ; підготовка наступного елемента масиву B1
       loop m1               ; перейти на m1, якщо CX. ≠ 0
       ret                   ; повернення з процедури
      prog1 endp                     ; закінчення процедури
      end _start             ; закінчення програми з ім'ям _start
```

Стан програми після її виконання наведено на рис. 3.3.1.

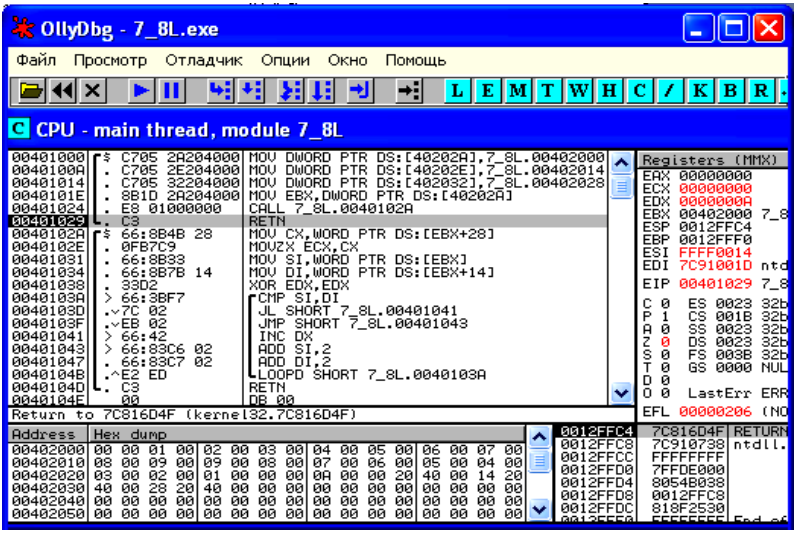


Рис. 3.3.1. Стан програми з лістингу 3.1 після її виконання

У цій програмі для того щоб заповнити нулями старшу частину лічильника, використано команду movzx ecx,cx, хоча можна попереду заповнити нулями цей регістр, а потім завантажити його необхідним числом.

У розглянутій програмі використовується одна процедура.

Головну програму, з якої викликаються інші процедури, також можна оформити у вигляді процедури. Таку програму, що обчислює суму та різницю двох цілих чисел з використанням трьох процедур та непрямого виклику двох додаткових, розглянуто в лістингу 3.3.2.

Лістинг 3.3.2. Програма обчислення суми та різниці двох цілих чисел з використанням непрямого виклику процедур:

```
        title CopyRight by Rysovaniy A. N.
        .686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам’яті та угоди ОС Windows
    option casemap: none                ; відмінність малих та великих літер
.data                               ; директива визначення даних
        tbl label dword         ; привласнення tbl подвійного слова
DD sub1 ; резервування під sub1 в пам’яті комірок
                                        ; розміром у подвійне слово (32 розряда)
DD sub2 ; резервування під sub2 комірок розміром у подвійне слово
        op1  DD -29             ; –29 = FFFF FFE3h
        op2  DD 59              ;   59 = 0000 003Bh
```

```
 res DD 2 DUP(0) ; резервування комірок розміром у два подвійних слова
    .code                          ; директива початку коду програми
_start:                            ; мітка початку програми з ім'ям _start
      _main proc                                          ; головна процедура
         lea   ESI, tbl                          ; занесення в ESI адреси tbl
         mov  [ESI], offset sub1          ; занесення адреси процедури sub1
         mov  [ESI+4], offset sub2 ;
         call  dword ptr [ESI]             ; непрямий виклик процедури sub1
         call  dword ptr [ESI+4]           ; непрямий виклик процедури sub2
         lea   EAX, res                 ; занесення адреси результату до EAX
         ret                            ; повернення управління ОС
      _main endp                                          ;   закінчення   головної
процедури
      sub1 proc
         clc                         ; очищення прапорця перенесення
         mov EAX, op1
         adc EAX, op2
         mov res, EAX
         ret                           ; повернення з процедури
      sub1 endp
      sub2 proc
         clc                         ; очищення прапорця перенесення
         mov EAX, op1
         sbb EAX, op2
         mov res+4, EAX
         ret                           ; повернення з процедури
      sub2 endp                        ; закінчення процедури
      end _start                       ; закінчення програми з ім'ям _start
```

Мітка start при виконанні декількох процедур є обов'язковою. Значення комірок пам'яті до моменту виконання програми наведено на рис. 3.3.2.

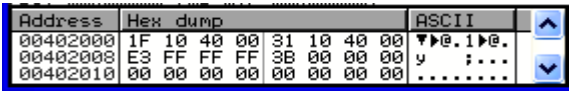


Рис. 3.3.2. Значення комірок пам'яті до моменту виконання програми

Значенню sub1 привласнюється адреса 0040101F початку цієї процедури, значенню sub2 – адреса 00401031 початку відповідної процедури.

У цій програмі для простоти не застосовуються Windows API-функції. Стан програми після виконання останньої команди основної програми (ret) наведено на рис. 3.3.3.

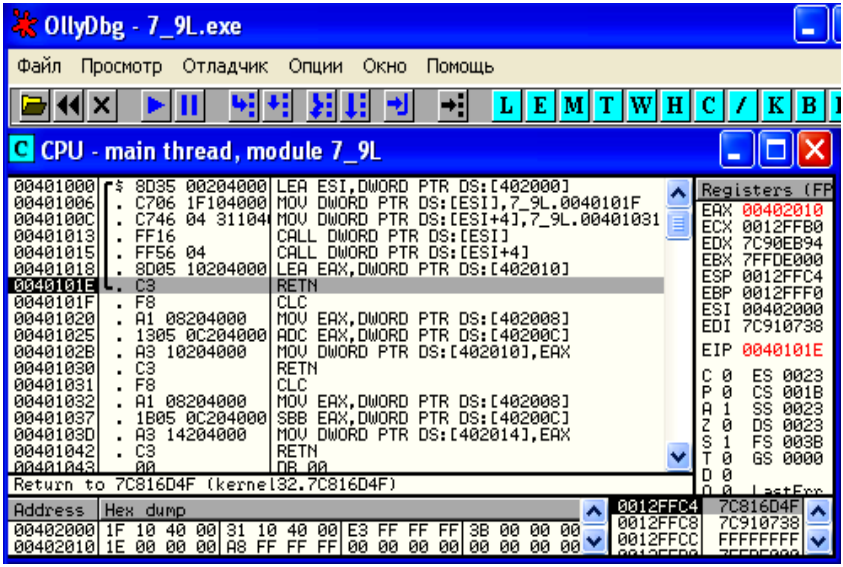


Рис. 3.3.3. Стан програми з лістингу 3.2 після виконання

Програма, що розглядається, складається з трьох процедур: 1) головної – з ім'ям _main; 2) такої, що виконує додавання, – з ім'ям sub1; 3) такої, що виконує віднімання, – з ім'ям sub2. Процедура sub1 обчислює суму чисел i1 та i2 та розташовує результат у молодше подвійне слово змінною res. Процедура sub2 виконує віднімання тих самих чисел та розміщення результату у старше подвійне слово змінної res. Головна процедура _main викликає процедури за адресою, яка знаходиться у регістрі ESI, а регістр ESI отримує цю адресу з таблиці tbl, яка вміщує відповідну адресу. За командою lea ESI, tbl в ESI фактично заноситься адреса процедури sub1.

3.4. Лабораторна робота 4
API-ПОДІБНІ ПРОЦЕДУРИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм, написаних мовою асемблера для програмування API-подібних функцій для МП платформи x86.

Вхідний контроль знань

Написати та налагодити програму згідно з отриманим варіантом та оформити її у вигляді процедури з параметрами. Результат записати у пам'ять.

1. Виконати операцію логічного додавання дня та року свого народження.
2. Виконати операцію логічного порівняння місяця та року свого народження.
3. Виконати зсув дня свого народження ліворуч на 2 розряди.
4. Перемножити день, місяць та рік свого народження.
5. Поділити рік свого народження на місяць народження.
6. Виконати зсув дня свого народження ліворуч на 4 розряд.
7. Скласти день, місяць та рік свого народження.
8. Виконати зсув року свого народження праворуч на 2 розряди.
9. Виконати операцію логічного перемноження місяця та року свого народження.
10. Виконати операцію логічного додавання за модулем місяця та року свого народження.

Постановка задачі

Згідно з номером студента в групі вибрати варіант завдання та написати на асемблері програму обчислення одного з виразів:

1. $2d/c - cd$;	8. $2ab - 8c/b$;	15. $d/3c - 15ac$;	22. $2b - e/22b$;
2. $2a - e/2c$;	9. $8d - 9d/c$;	16. $2d/3c - 16cd$;	23. $ab/3a - 23b$;
3. $cb/3a + 3a$;	10. $3e/b + 10c$;	17. $2a/7 - d/17e$;	24. $d/4a - 24d/c$;
4. $d/3b - d/4c$;	11 $2d/3b - 11c$;	18. $2ab - 18c/d$;	25. $e/8b + 25ac$;
5. $e/3c + 5ac$;	12. $3b - 12c/d$.	19. $8d/b - 19d/c$;	26. $4d/4a - 26cd$;
6. $2d/3c - 6cd$;	13. $cb/3a - 13a$;	20. $3d/b + 20e$;	27. $8a/7 - d/27b$;
7. $2a/7 - c/7e$;	14. $e/4b - d/14c$;	21. $2d/a - 21bd$;	28. $4ac - 28c/b$,

де a – номер студента за списком у навчальній групі;

b – номер навчальної групи;
c – *pp* – дві останні цифри свого року народження;
e – *pppp* – чотири цифри свого року народження;
d – *ддмм*:
дд – день свого народження;
мм – місяць свого народження.

Результат обчислення виразу зберегти в пам'яті. Навести значення та порядок розміщення даних у пам'яті.

Зміст звіту

1. Постановка задачі для отриманого варіанта завдання.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем.
3. Лістинг програми з детальним коментарем до кожної команди.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.4.1. Написати на асемблері програму обчислення виразу

$$a \times b + c,$$ де $a = 4$, $b = 5$, $c = 6$,

оформити її у вигляді API-подібної функції та зберегти результат виконання виразу у пам'яті.

Лістинг 3.4.1. Програма обчислення прикладу 3.4.1:

```
title CopyRight by Rysovaniy A. N.                    rysov@rambler.ru
.386                        ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none                  ; відмінність малих та великих літер
includelib \masm32\lib\kernel32.lib  ; виклик бібліотеки kernel32.lib
ExitProcess proto :DWORD                          ; прототип процедури
OperDigs proto :WORD, :WORD, :WORD      ; прототип процедури
.data                                          ; директива визначення даних
res dw 2 dup(0)                  ; резервування комірок для результату
.code                                  ; директива початку програми
_start:                                ; мітка початку програми з ім'ям _start
invoke OperDigs,4,5,6                  ; виклик директиви з параметрами
invoke ExitProcess,0                   ; виклик директиви з параметрами
OperDigs proc arg1:WORD,arg2:WORD,arg3:WORD
   mov ax,arg1                 ; ax := arg1 (число 4)
   mul arg2                    ; dx,ax = ax × arg2
   add ax, arg3                ; ax := ax + arg3 (число 6)
   jc m1                       ; перейти, якщо є перенесення
```

```
        jmp m2              ; безумовний перехід на збереження результату
m1: inc dx                  ; dx + 1, якщо перенесення є
m2: mov res,ax                      ; запам'ятовування в пам'яті
    mov res+2,dx                    ; запам'ятовування в пам'яті
    ret                             ; повернення з процедури
OperDigs endp                       ; закінчення процедури OperDigs
end _start                          ; закінчення програми з ім'ям _start
```

У зв'язку з тим, що числа вибрані невеликі, то для їх визначення вибрано слово WORD. Команди розгалуження jc та jmp необхідні для випадку, коли використовуються числа, які могуть дати переповнення розрядної сітки.

Порядок розміщення даних у пам'яті та стан налагоджувача OllyDbg з програмою з лістингу 3.4.1 наведено на рис. 3.4.1.

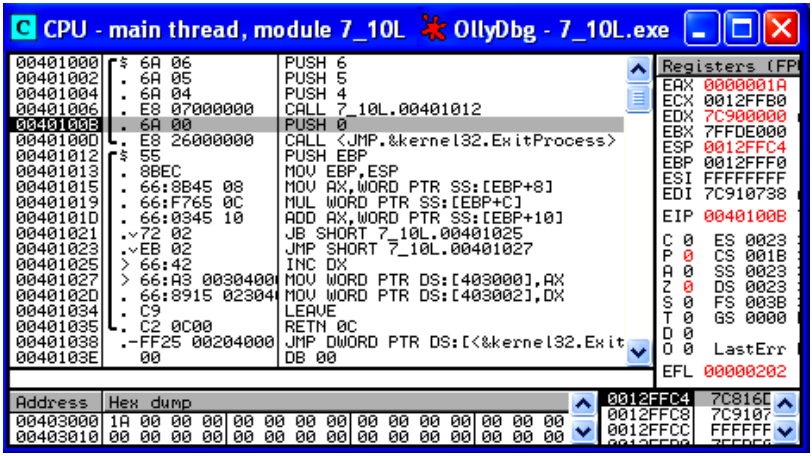


Рис. 3.4.1. Порядок розміщення даних у пам'яті та стан налагоджувача OllyDbg з програмою з лістингу 3.4.1

Параметри процедури OperDigs розміщуються в стеці за порядком справо наліво (спочатку – число 6 := arg3, потім – число 5 := arg2, а на остаток – число 4 := arg1). *Директива invoke замінює дві команди: CALL та PUSH 0, де PUSH 0 – код успішного завершення, який повертається операційній системі.* При виконанні команди CALL 7_10L.00401012 після аргументів записується код повернення в процедуру, а вже потім передається управління командам процедури.

Результат виконання операції прикладу розташовано у двох регістрах: EAX = 001A, та EDX = 0000.

3.5. Лабораторна робота 5
ЗОВНІШНІ ПРОЦЕДУРИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм, написаних мовою асемблера з використанням ***зовнішніх процедур*** для МП платформи x86.

Постановка задачі

Згідно з номером студента в групі вибрати варіант завдання та написати на асемблері програму обчислення одного з виразів:

1. $(a - bc) + e/d$;	9. $c - ad + b/e$;	17. $(c/e + 2a)b$;
2. $(ce + a)b - 2d/a$;	10. $(a - ed)b - d/b$;	18. $(a - bc)/b$;
3. $(a - cb)/b + 3d$;	11. $(a - b)ab - d/e$;	19. $8c - ab - b/19e$;
4. $8d - 4d/a$;	12. $(b - 2d)/(12a - c)$;	20. $(a - cd) - a/b$;
5. $5ac - 5b/c$;	13. $ae + b - b/c$;	21. $(a - b)ab - d/e$;
6. $(a - b/c) + 2ed$;	14. $8ad - b - d/a$;	22. $(a - 2d)/(a - c)$;
7. $2(c/e + a)b$;	15. $a - 5ac - b/c$;	23. $ab + c - b/c$;
8. $(a - c/b)/8b$;	16. $2(a - b/c) + ed$;	24. $8ab - d/a$.

Вимоги до програм

1. Передбачити ситуацію, при якій числа будуть змінюватися від мінімального до максимального значень (урахувати можливі переповнення розрядної сітки та позики).

2. Для перших десяти за списком студентів змінні повинні мати розмір ***байта***, для других десяти за списком студентів – розмір ***слова***, а для третіх десяти за списком студентів – розмір ***подвійного слова***.

3. Навести програму з використанням ***зовнішніх*** процедур.

4. Указати діапазон зміни змінних.

Зміст звіту

1. Постановка задачі для отриманого варіанта завдаання.

2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.

3. Лістинг програм: головної та зовнішньої процедури з детальним коментарем та описом роботи.

4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання виразу.

5. Короткий опис виконання програми.

6. Висновки за результатами роботи.

Приклад 3.5.1. Хай потрібно обчислити вираз *ab – c/d* , де *a, b, c, d* – цілі числа розміром у подвійне слово.

Обчислення виконаємо за допомогою однієї процедури, яка розташована у файлі 7_9_1.asm, що обчислює *ab – c/d*. В головній програмі виконується тільки виклик зовнішньої процедури.

Складемо схему виконання прикладу (рис. 3.5.1).

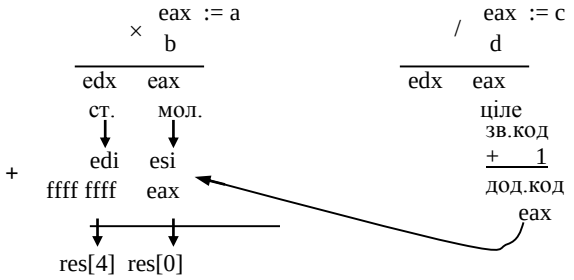


Рис. 3.5.1. Схема виконання прикладу 3.5.1

У зв'язку з тим, що в прикладі необхідно виконати віднімання операцій, а значення результату віднімання не відоме (*ab* може бути мінше, чим *c/d*), то спочатку результат *c/d* перетворюється в додатковий код, а вже потім виконується віднімання. Остаточній результат також буде в додатковому коді. Остачу ділення можна не запам'ятовувати. Згідно зі схемою складемо головну та додаткову програми. Головну програму з іменем 7_9_1.asm наведено у лістингу 3.5.2.

Лістинг 3.5.2. Головна програма обчислення виразу *ab – c/d*:

```
title CopyRight by Rysovaniy A. N.
.386                    ; директива визначення типу мікропроцесора
.model flat                     ; задання лінійної моделі пам'яті
Extern _abcd:proc               ; указання на зовнішню процедуру
Public _a, _b,_c, _d, _res      ; доступ для інших модулів
.data                           ; директива визначення даних (ab  –  c/d)
_a   dd  1                      ; запис у 32-розрядну комірку з ім'ям _a
_b   dd  2                      ; запис у 32-розрядну комірку з ім'ям _b
_c   dd  7                      ; запис у 32-розрядну комірку з ім'ям _c
```

```
    _d   dd  2              ; запис у 32-розрядну комірку з ім'ям _d
    _res  dd 3 dup(0)       ; комірки для збереження результату
.code                                ; директива початку програми
  _start:                    ; мітка початку програми з ім'ям _start
     call _abcd                      ; виклик процедури
    ret                              ; повернення управління ОС
   end _start          ; директива закінчення програми з ім'ям _start
```

Як команда асемблера програма з лістингу 5.2 складається з однієї команди call _abcd. Всі інші команди наведено в іншому файлі з іменем 7_9_2.asm, який містить процедуру обчислення виразу *ab – c/d* (лістинг 3.5.3).

Лістинг 3.5.3. Текст процедури з іменем _abcd:

```
    title CopyRight by Rysovaniy A. N.
    .386              ; директива визначення типу мікропроцесора
   .model  flat             ; задання лінійної моделі пам'яті
    public _abcd
    extern _a:dword, _b:dword, _c:dword,_d:dword,_res:dword
.code                            ; директива початку програми
  _abcd  proc                    ;  ab - c/d
    mov  eax,_a    ; пересилання з комірки пам'яті з ім'ям _a в eax
    mul _b                       ; edx, eax := eax × _b
    mov esi,eax ; збереження молодшої частини результату множення
    mov edi,edx                  ; збереження старшої частини
    mov eax,_c                   ; підготовка до ділення
    div _d                       ; eax, edx := eax/_d ; в edx – остача
     not eax                     ; інверсія (зворотній код цілого числа)
     add eax,1                   ; додатковий код цілого числа
     add eax,esi                 ; ціла молодша частина
mov _res[0],eax ; запам'ятовування молодшої цілої частини у пам'яті
     adc edi,0ffffffffh          ; ціла старша частина
mov _res[4],edi  ; запам'ятовування старшої цілої частини у пам'яті
     ret
     _abcd  endp
     end                         ; директива закінчення програми
```

Для виконання програми зручно створити командний bat-файл зі змістом:

```
ml.exe  /c  /coff   7_9_1.asm  7_9_2.asm
link.exe /subsystem:console 7_9_1.obj  7_9_2.obj
pause
start 7_9_1.exe
```

Після того, як такий командний bat-файл створено необхідно його виконати (двічі натиснути на нього). Процес отримання exe-файлу наведено на рис. 3.5.2.

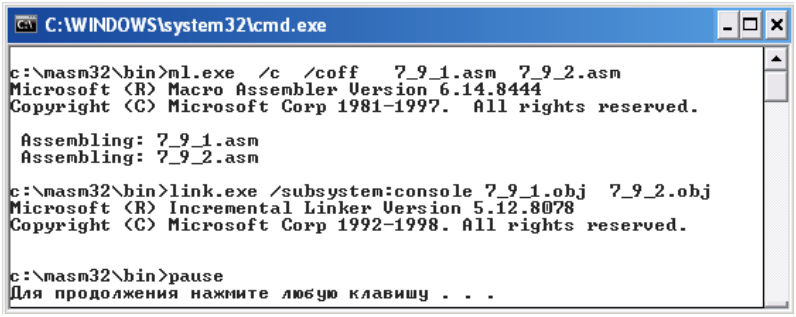


Рис. 3.5.2. Процес отримання exe-файлу

Результат виконання програми в додатковому коді можна побачити на рис. 3.5.3 у вікнах налагоджувача.

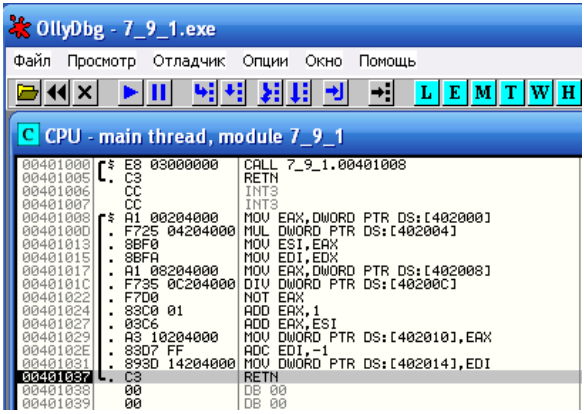


Рис. 3.5.3. Процес виконання програми та одержання результату

3.6. Лабораторна робота 6
ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням масивів, написаних мовою асемблера для програмування розгалужених процесів МП платформи x86 із застосуванням API-функцій під Win32 та **введенням-виведенням** даних на екран.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати варіант завдання та написати на асемблері програму обчислення одного з прикладів з ***введенням та виведенням*** даних. При цьому обов'язково використати спрощені віконця функцію з повідомленнями: в одному віконці вивести своє прізвище та номер своєї навчальної групи; при натисканні на другу кнопку – вивести повне завдання із звуковим повідомленням. Використання інших кнопок спрощеного віконця виведення повідомлень – за бажанням.

Варіанти завдань

1. Задано масив A з $N = 5$ елементів. Навести алгоритм та програму визначення суми елементів масиву A, для яких біти 0,1 та 4 збігаються.

2. Задано масив A з $N = 6$ елементів. Навести алгоритм та програму визначення суми елементів масиву A, для яких біти 2 та 7 збігаються.

3. Задано масиви A і B з $N = 5$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо у елементів масивів Ai та Bi біти 0, 1 та 2 збігаються, то $Ci = Ai – Bi$.

4. Задано масив A з $N = 7$ елементів. Навести алгоритм та програму формування масиву B з елементів масиву A, у яких 0, 2 та 5 біти мають нулі.

5. Задано масиви A і B з $N = 8$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо у елементів Ai та Bi біти 0, 1 та 2 збігаються, то $Ci = Ai + Bi$.

6. Задано масиви A і B з $N = 4$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai + Bi \le 0$, то $Cj = Bi$.

7. Задано масив A з $N = 5$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \le Ai < M$, де $L = -3$ та $M = 15$.

8. Задано масив A з $N = 6$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L < Ai < M$, де $L = 4$ та $M = 18$.

9. Задано масив A з $N = 7$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \le$ Ai $\le M$, де $L = -5$ та $M = 19$.

10. Задано масив A з $N = 8$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \ge Ai < M$, де $L = 6$ та $M = 20$.

Зміст звіту

1. Постановка задачі для конкретного варіанта завдання.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.6.1. Для функції $Y = 40X + 15$ одержати перше значення, що перевищує 512, починаючи з $X = 2$. Значення аргументу та функції вивести на екран.

Алгоритм вирішення прикладу 6.1 наведено на рис. 3.6.1.

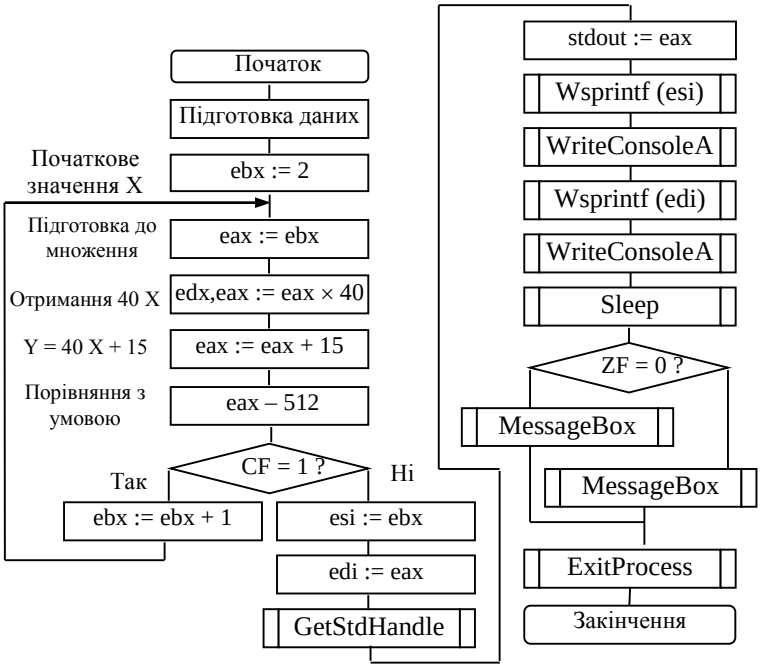


Рис. 3.6.1. Алгоритм розв'язання прикладу 3.6.1

Лістинг 3.6.1. Програма виконання прикладу 3.6.1:

```
      title CopyRight by Rysovaniy A. N.
      .386                    ; директива визначення типу мікропроцесора
.model  flat, stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\\user32.inc   ; файли інтерфейсу …
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
     includelib \masm32\lib\user32.lib
     includelib \masm32\lib\kernel32.lib
; Для функції У=40Х+15 одержати перше значення, при якому
   ; Y > 512, ; починаючи з X = 2. Значення аргументу та функції
     ; записати в комірки пам'яті. Вивести результат на екран
BSIZE equ 15                    ; задання реальної кількості байтів
.data                           ; директива визначення даних
       x dd 40          ; резервування 32-розрядної комірки пам'яті для x = 40
       y dd ?       ; резервування 32-розрядної комірки пам'яті для змінної Y
buf db BSIZE dup(?)                ; резервування пам'яті для буфера
```

```
frmt db "%d",0                    ; задання перетворення одного символу
stdout DWORD ?                    ; резервування в пам'яті 32-розрядної комірки
cWritten DWORD ?  ; резервування 32-розрядної комірки пам'яті
                                  ; з ім'ям cWritten для адреси символів виведення
     st1 db "       y = 40X + 15",0
     st2 db "Автор: Рысованый А.Н.",0
     st3 db " ЛР Организация ввода-вывода",0
     st4 db "Программа выполнена успешно",0
     st6 db "До следующей лабораторной работы !!!",0
.code                             ; початок сегмента  даних
start:                            ; мітка початку програми
   mov ebx,2                  ; початкове значення аргументу X
m1: mov eax,ebx                   ; підготування до множення
   mul x                      ; виконання частини функції Y (40X)
   add eax,15                 ; виконання всієї функції Y = 40X + 15
   cmp eax,512                ; порівняння з умовою
   jnc exit                   ; перейти, якщо Y > 128
   inc ebx                    ; збільшити лічильник аргументу X
   jmp m1                     ; перейти при  Y < 128
exit:
      mov esi,ebx             ; збереження X
      mov edi,eax                 ; збереження Y
invoke GetStdHandle, STD_OUTPUT_HANDLE    ; отримання дескриптора
  mov stdout, eax             ; збереження одержаного дескриптора у пам'яті
invoke  wsprintf, \           ; API-функція перетворення числа
 ADDR buf, \ ; адреса буфера, куди буде записана послідовність символів
    ADDR frmt, \                  ; адреса рядка перетворення формату
      esi                         ; регістр, вміст якого перетворюється
invoke WriteConsoleA, stdout,\ ; функція виведення на екран та дескриптор
       ADDR buf, \                ; адреса початку повідомлення
          BSIZE, \                ; розмір повідомлення
             ADDR cWritten, 0        ; адреса, де зберігається число символів
invoke wsprintf, ADDR buf, ADDR frmt,edi    ; функція перетворення edi
invoke WriteConsoleA, stdout, \ ; виведення на екран та дескриптор
  ADDR buf, \                 ; адреса початку повідомлення
   BSIZE,  \                  ; розмір повідомлення
    ADDR cWritten, 0              ; адреса, де зберігається число символів
invoke Sleep, 2000d               ; API-функція затримки зображення
invoke MessageBox,0 \ ; виведення вікна консолі та ідентифікатор вікна
 addr st2, \                  ; адреса рядка, який містить  текст повідомлення
  addr st1, \             ; адреса рядка, який містить  заголовок повідомлення
   MB_OKCANCEL                        ; вигляд діалогового вікна
      xor eax,1
      jz a1
invoke MessageBox, NULL, addr st4, addr st3, MB_ICONINFORMATION
      jmp a2
```

```
a1:  invoke MessageBox, NULL, addr st6, addr st3, MB_OK
a2:
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
        end start             ; директива закінчення програми з ім'ям start
```

Наприкінці програми показано використання API-функції MessageBox. Останній її параметр повертає до регістра eax або 1 (якщо OK), або 2 (якщо CANCEL). За допомогою команди xor (додавання за mod2), якщо значення регістра eax дорівнювало 1, прапорець ZF встановиться в 1. У такому випадку виводиться одне повідомлення, у іншому випадку – друге повідомлення.

Ліворуч наведено змінну *X*, а праворуч – *Y*. Є декілька варіантів підпису. Найбільш простим є варіант, при якому можна використати директиви

```
frmt1 db "x = %d",0
frmt2 db "y = %d",0
```

При іншому варіанті програма суттєво модернізується. Спочатку, як наведено у лістингу 6.1, замість функції wsprintf напишемо підпрограму, яка переводить число у десяткову систему числення (діленням на 10), а потім додаємо константу 48D = 30h = '0' (лістинг 3.6.1).

На рис. 3.6.2 наведено результат виконання лістингу 3.6.1.

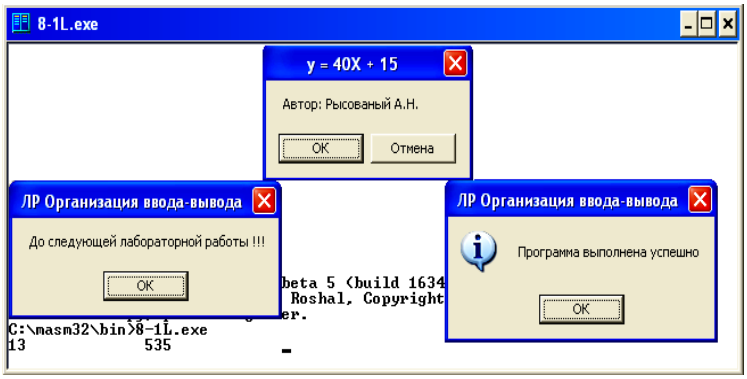


Рис. 3.6.2. Результат виконання лістингу 3.6.1

Приклад 3.6.2. Задано масиви *A* і *B* з *N* = 6 елементів. Навести алгоритм та програму формування масиву *C* за таким правилом: якщо у елементів *Ai* та *Bi* біти 0 та 7 збігаються, то *Ci* = *Ai* + *Bi*.

Алгоритм розв'язання прикладу 3.6.2 наведено на рис. 3.6.3.

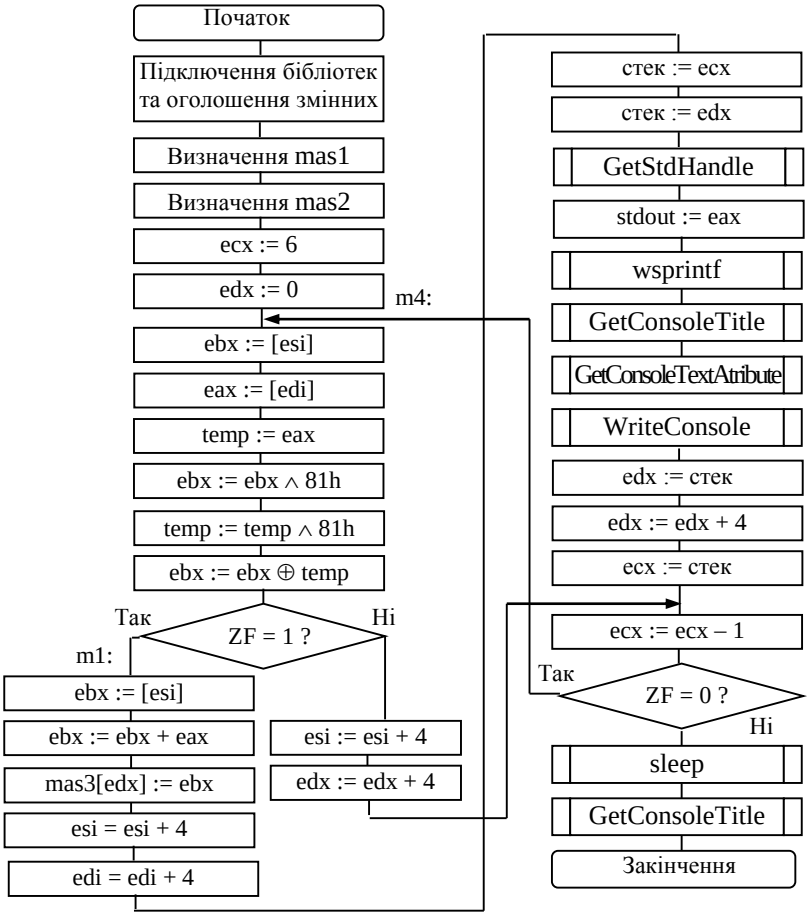


Рис. 3.6.3. Алгоритм розв’язання прикладу 3.6.2

Особливістю алгоритмів з трьома та більше масивами є те, що для таких задач не достатньо регістрів загального призначення.

Цю нестачу можна виправити використанням непрямої адресації та (чи) комірок пам’яті для тимчасового збереження даних. У програмі для запису даних у третій масив використовується змінна в пам’яті mas3 зі зміщенням, яке розташоване в регістрі edx.

Вирівнювання адрес комірок у масивах виконується за допомогою команд

esi := esi + 4;
edi := edi + 4;
edx := edx + 4.

Лістинг 3.6.2. Програма виконання прикладу 3.6.2:

```
        title CopyRight by Rysovaniy A. N.
; масиви A і B з N = 6 елементів. Сформувати масив C:
; якщо у  Ai та Bi  біти 0 та 7 збігаються, то Ci = Ai + Bi.
        .686                    ; директива визначення типу мікропроцесора
        .model  flat, stdcall  ; задання лінійної моделі пам'яті та угоди ОС
        option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include  \masm32\include\\user32.inc   ; файли інтерфейсу …
include\masm32\include\kernel32.inc ;файли систем. функцій застосувань…
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
        BSIZE equ 15                        ; задання реальної кількості байтів
.data                                       ; директива визначення даних
mas1 DD 0, 81h, 33, 81h, 0FFh, 5   ; резервування пам'яті для mas1
mas2 DD 0, 81h, 0, 99h, 0FFh, 6     ; резервування пам'яті для mas2
temp dd ?                          ; резервування пам'яті для змінної temp
mas3 dd 6 dup(0) ;
stdout DWORD ?                ; резервування в пам'яті 32-розрядної комірки
buf db BSIZE dup(?)           ; резервування пам'яті для буфера
frmt db "%d",0                ; задання перетворення одного символу
cWritten DWORD ?      ; резервування 32-розрядної комірки пам'яті
                              ; з ім'ям cWritten для адреси символів виведення
mytitle    db "laboratory work", 0
color dd 40h or 20h or 80h ; фон червоний + зелений +  підвищ. яскравості
.code                                       ; початок сегмента даних
  start:                                    ; мітка початку програми
        lea esi,mas1                        ; записати адресу початку масиву mas1
        lea edi,mas2                        ; записати адресу початку масиву mas2
        mov ecx,6                           ; лічильник кількості пар чисел
        xor edx,edx                 ; початкове значення зміщення в масиві mas3
m4:  mov ebx, [esi]                   ; переслати число з масиву mas1
        mov eax,[edi]                  ; переслати число з масиву mas2
        mov temp,eax               ; перезбереження для подальшого додавання
        and ebx,81h                   ; логічне множення числа з mas1 на маску
        and temp,81h                  ; логічне множення числа з mas2 на маску
        xor ebx,temp                    ; mas1i ⊕ mas2i
        jz m1                           ; перейти, якщо 0
    add esi,4            ; підготування до читання наступного елемента з mas1
    add edi,4            ; підготування до читання наступного елемента з mas2
        jmp m3 ;
```

```
m1:  mov ebx,[esi]                    ; читання елемента з mas1
       add ebx,eax                    ; формування елемента mas3
       mov mas3[edx],ebx              ; запис в mas3
   add esi,4        ; підготування до читання наступного елемента з mas1
   add edi,4        ; підготування до читання наступного елемента з mas2
       push ecx                       ; підготовка до виклику API-функції
       push edx                       ; підготовка до виклику API-функції
invoke GetStdHandle, STD_OUTPUT_HANDLE    ; отримання дескриптора
   mov stdout, eax              ; збереження одержаного дескриптора у пам'яті
invoke  wsprintf, \              ; API-функція перетворення числа
 ADDR buf, \  ; адреса буферу, куди буде записана послідовність символів
  ADDR frmt, \                   ; адреса рядка перетворення формату
   mas3[edx]                     ; комірка пам'яті, вміст якої перетворюється
invoke SetConsoleTitle, offset mytitle
invoke SetConsoleTextAttribute, stdout, color
invoke  WriteConsoleA, \  ; API-функція виведення в консоль
  stdout, \                    ; дескриптор стандартного пристрою виведення
   ADDR buf, BSIZE \  ; адр. початку повідомлення та розмір повідомлення
    ADDR cWritten, 0          ; адреса, де зберігається число символів
      pop edx                 ; встановлення edx
       add edx,4              ; підготування зміщення для запису в mas3
       pop ecx                ; встановлення ecx
m3:
      dec ecx                 ; зменшення лічильника пар чисел
      jnz m4                  ; перейти, якщо в лічильнику не нуль
invoke Sleep, 2000d           ; API-функція затримки зображення
invoke  ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
      end start               ; директива закінчення програми з ім'ям start
```

Результат виконання програми наведено у вікні налагоджувача OllyDbg на рис. 3.6.4.

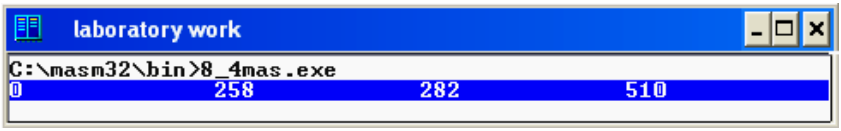


Рис. 3.6.4. Результат виконання програми з лістингу 3.6.2

Програма виконує виведення рядка результату у консоль, яка має свої колір фону та назву. Індивідуальну назву консолі забезпечує API-функція SetConsoleTitle. Колір фону задається API-функцією SetConsoleTextAttribute.

Приклад 3.6.3. Задано масив A з $N = 4$ елементів. Навести програму визначення суми елементів масиву A, для яких біти 0 та 5 збігаються.

Лістинг 3.6.3. Програма виконання прикладу 3.6.3:

```
        title CopyRight by Rysovaniy A. N.
        .686      ; директива визначення типу мікропроцесора
.model flat, stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
        option casemap:none             ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include    \masm32\include\\user32.inc   ; файли інтерфейсу …
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
        BSIZE equ 15            ; задання реальної кількості байтів
.data                           ; директива визначення даних
buf db 10 dup(?),0              ; місце у вікні для виведення повідомлення
        ifmt db "Задано массив из 4-х элементов:",0dh,0ah,\
        "10   12   13   14",0dh,0ah,0ah,\
"Сумма элементов массива, для которых биты 0 и 5 совпадают: =
%d",0dh,0ah,0ah,\
        "Автор программы:  Рысованый А.Н., НТУ ХПИ",0
        titl1 db "Сумма элементов главной диагонали",0 ; назва віконця
        mas1 dw 10, 12, 13, 14                  ; масив mas1 слів
        len equ ($-mas1)/ 2             ; обчислення кількості  слів в mas1
        sum dw 0                        ; комірка для збереження результату
.code                                   ; початок сегмента-даних
 _st:                                   ; мітка початку програми
        mov ecx,len                     ; кількість байтів
        lea esi,mas1                    ; початкова адреса масиву mas1
m1:   mov ax,[esi]                      ; в ах заноситься елемент масиву
        bt ax,0                 ; вибір нульового біта
        setc bh                 ; якщо cf=1, то встановлення 1 в bh
        bt ax,5                 ; вибір п'ятого біта
        setc bl                 ; якщо cf = 1, то встановлення 1 в bl
        cmp bh,bl               ; порівняння бітів
        jne m2                  ; якщо не дорівнює, то перейти на m2
        add sum,ax              ; додавання вибраних елементів масиву
m2:  add esi,2          ; збільшення адреси mas1 для вибірки нового числа
        dec ecx                 ; зменшення лічильника чисел у масиві mas1
        jnz m1                  ; перейти на мітку m1 , якщо не нуль
        movzx edx,sum  ; збереження результату з розширенням розрядності
invoke wsprintf,ADDR buf,ADDR ifmt,edx     ; функція перетворення edx
invoke MessageBox,  ; API-функція виведення спрощеного вікна консолі
          NULL, ADDR buf, ADDR titl1, MB_OK
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
        end  _st                ; директива закінчення програми з ім'ям _st
```

Рядок len equ ($-mas1)/2 доцільніше записувати як

len equ ($-mas1)/**type** mas1,

щоб самостійно не розраховувати кількість байтів, які вміщуються у той або інший формат даних. Наприклад, якщо це dw, то кількість байтів у цьому форматі – 2; якщо це dd, то кількість байтів у цьому форматі – 4 й т.ін. Оператор **type** сам підставляє необхідну кількість байтів.

Результат виконання програми з лістингу 3.6.3 наведено на рис. 3.6.5.

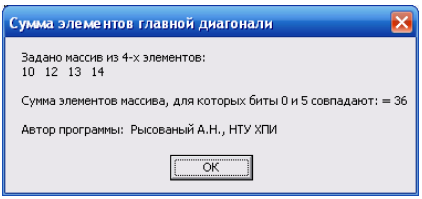


Рис. 3.6.5. Результат виконання програми з лістингу 3.6.3

Результатом виконання програми з лістингу 8.15 є сума трьох чисел: 10, 12, 14, для яких біти 0 та 5 збігаються.

Приклад 3.6.4. Задано масив A з $N = 100$ елементів (констант типу WORD). Сформувати масив з індексів елементів масиву A, у яких 1,3 і 5 біти містять одиниці.

Для налагодження програми зручно використовувати невеликий масив з конкретними даними. Після налагодження програми масив може бути визначений декількома варіантами.

Варіант1:

```
mas1 dw 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
   21,22,23,24,25,26,27,28,29,30,  й так далі до 99.
```

Варіант2:

```
buf dw 10 dup (1)
      dw 20 dup (0)
      dw 70 dup (255),0        ; буфер виведення повідомлення
```

Лістинг 3.6.4. Програма виконання прикладу 3.6.4:

```
title Рысованый А.Н.                                rysov@rambler.ru
.386          ; директива визначення команд мікропроцесора
.model  flat,stdcall ; задання лінійної моделі пам'яті
option casemap:none ; відмінність рядкових та прописних літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\kernel32.inc; файли систем. функцій застосувань
include \masm32\include\\user32.inc   ; файли інтерфейсу …
includelib \masm32\lib\user32.lib
```

```
        includelib \masm32\lib\kernel32.lib
        .data                                   ; директива визначення даних
  mas1 dw 10,42,58,6,7,8 ; резервування в області пам'яті з іменем mas1
                ; комірок розміром у байт для чисел цього масиву
len equ ($-mas1)/ type mas1   ; обчислення кількості слів масиву mas1
        memRes dd 6 dup(0),0
        _title db "Результаты решения программы",0
        info db "Задан массив A из N = 6 элементов.",0Ah,0Dh,
"Сформировать массив B из индексов элементов массива A, ", 0Ah, 0Dh,
        "у которых 1,3 и 5 биты содержат единицы.",0Ah,0Dh,0Dh
        buf dd len dup (?),0          ; буфер виведення повідомлення
        ifmt  db "Ответ: mas3 =  %d  %d  %d  %d  %d  %d",0
.code                                   ; директива початку сегмента команд
_start:                         ; мітка початку програми
        mov ecx, len            ; лічильник кількості байтів у масиві mas1
        mov edx, 0              ; лічильник кількості елементів mas1[5,3,1] = 1
        mov ebx,0 ;
        lea esi, mas1                   ; занесення адреси масиву mas1 в esi
        begin: mov eax, [esi]           ; занесення елемента  mas1  в  eax
        bt eax,1                        ; виділення 1-го біта в eax
        jc a1                           ; якщо CF = 1, то перейти на a1
        jmp _exit               ; а якщо CF = 0, то перейти на мітку _exit
a1:   bt eax,3                  ; виділення 3-го біта в eax
        jc a2                   ; якщо CF = 1, то перейти на a2
        jmp _exit               ; а якщо CF = 0, то перейти на мітку _exit
a2:   bt eax,5                  ; виділення 5-го біта в eax
        jnc _exit               ; якщо CF = 0, то перейти на мітку _exit
m3:  mov memRes[ebx],edx   ; переписати номер елемента в пам'ять
        add ebx,4 ; підготовка адреси для запису нового номера елемента
_exit:  inc edx                 ; збільшення лічильника елементів mas1[0] = 0
        add esi,2          ; підготовка адреси для зчитування нового елемента
        dec ecx            ; перевірка лічильника кількості байтів у масиві mas1
        jnz begin          ; якщо в ecx не нуль, то перейти  до наступного циклу
invoke  wsprintf, \                     ; API-функція перетворення чисел
  ADDR buf, \     ; адреса буф., куди буде записана послідовність символів
   ADDR ifmt, \             ; адреса рядка перетворення формату
 memRes[0],memRes[4],memRes[8],memRes[12],memRes[16],memRes[20] ;
        invoke MessageBox, 0, addr info, addr _title, MB_ICONINFORMATION
        invoke ExitProcess,0
        end  _start         ; директива закінчення програми з ім'ям _start
```

Результат виконання програми з лістингу 3.6.4 наведено на рис. 3.6.6.

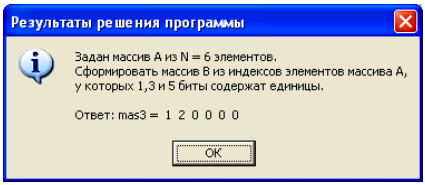


Рис. 3.6.6. Результат виконання програми з лістингу 3.6.4

Як лічильник елементів, у яких біти 1, 3 та 5 дорівнюють одиницям, використано регістр edx. У зв'язку з тим, що функція wsprintf використовує 32-розрядні дані в комірках пам'яті memRes, то для розміщення номерів елементів в пам'яті відбувається збільшення адреси на 4 байти командою

```
add ebx,4
```

Для виведення результату зарезервовано 6 комірок. Однак при наведених даних тільки числа 42 та 58 (перший та другий елементи) мають одиниці в 1, 3 та 5 розрядах.

3.7. Лабораторна робота 7
ФАЙЛИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм зі створенням файлів та запису в них даних.

Зміст звіту

1. Постановка задачі для конкретного варіанта завдання.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Програма та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Постановка задачі

Завдання 1

Згідно з останньою цифрою номера студента в групі вибрати варіант завдання та написати програму ***з виведенням даних у файл та в функцію*** **MessageBox.** Додати своє прізвище та ініціали в файл, який був отриманий у результаті виконання завдання. Парні номери роблять запис у кінець файлу, а непарні – з потокової позиції вказівника файлу. Визначити розмір створеного файлу. В спрощене вікно вивести свої прізвище та ініціали, номер навчальної групи, e-mail, завдання та результати його виконання. Використати функцію **SHFileOperation**.

1. Проаналізувати масив даних з 15 елементів. Підрахувати і зберегти кількість елементів масиву, якщо їх значення менше або більше 132, та кількість елементів масиву, значення яких дорівнюють 132. Вивести відповідні повідомлення.

2. Проаналізувати масив даних з 10 елементів. Додавати елементи масиву доти, поки значення суми не перевищить 512. Зберегти номер елемента, на якому відбулося переповнення. Якщо сума елементів не досягла значення 512, то видати відповідне повідомлення.

3. Проаналізувати масив даних з 16 елементів. Елементами масиву є числа 32, 64, 96 і 128. Підрахувати та вивести на екран кількість повторень кожного елемента.

4. Проаналізувати масив даних з 12 елементів. Створити масив, до якого входять елементи першого масиву, що дорівнюють 128. Перервати виконання програми, якщо буде знайдено 5 елементів зі значенням 128. Вивести відповідні повідомлення.

5. Проаналізувати 2 масиви, що складаються з 15 елементів кожен. Підрахувати кількість елементів першого масиву, що мають рівні значення в другому масиві. Вивести відповідні повідомлення.

6. Проаналізувати масив даних з 14 елементів. Підрахувати кількість елементів, значення яких дорівнює 55h. Рахунок перервати, якщо кількість елементів перевищить 3. Вивести відповідні повідомлення.

7. Проаналізувати масив даних з 15 елементів. Елементами масиву є числа 10, 20, 30 і 180. Підрахувати кількість повторень кожного елемента. Вивести відповідні повідомлення.

8. Для функції $Y = 40X + 65$ знайти перше значення аргументу, при якому значення функції перевищить 1024. Початкове значення аргументу $X = 20$. Вивести відповідні повідомлення.

9. Для функції $Y = 40X + 10$ одержати перше значення, що перевищує 512, починаючи з $X = 1$. Вивести значення аргументу та функції.

10. Для функції $7X + 85$ знайти перше значення аргументу, при якому молодші цифри результату виконання функції дорівнюють 155. Вивести результат виконання функції та його аргумент.

Завдання 2

Згідно з останньою цифрою номера студента в групі вибрати варіант завдання та написати програму:

1. Визначення версії програми.
2. Визначення дати останньої зміни файлу.
3. Визначення дати останнього доступу до файлу.
4. Визначення дати створення файлу.
5. Визначення типу файлу.
6. Визначення властивостей файлу.
7. Визначення дати створення файлу
8. Видалення каталога з підкаталогами.
9. Визначення файлів у певній директорії.
10. Визначення шляху до особистої програми.
11. Копіювання директорії.

При написанні програми можна використати посилання на сайт http://www.desksoft.ru/ з вибором ікони сайта з підписом DRKB.

Приклад 3.7.1. Проаналізувати масив даних з 16 елементів. Елементами масиву є числа 32, 64, 96 і 128. Підрахувати та вивести на екран кількість повторень кожного елемента.

Програма з лістингу 3.7.8 виводить у спрощене віконце та створює файл із записом кількості чисел, які збіглися за умовами програми.

Лістинг 3.7.1:

```
    .386                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
    option casemap:none             ; відмінність великих та малих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\\user32.inc   ; файли інтерфейсу …
include \masm32\include\kernel32.inc; файли систем. функцій застосувань
include \masm32\include\\masm32.inc
    includelib \masm32\lib\user32.lib
    includelib \masm32\lib\kernel32.lib
    includelib \masm32\lib\masm32.lib
.data                       ; директива визначення даних
    _mas dd 32, 64, 32, 128, 96, 96, 64, 128, 64, 64, 32, 64, 96, 128, 128, 32
    len equ ($-_mas)/ type _mas          ; визначення кількості чисел
    _n1 dd ?                             ; комірка для збереження _n1
    _n2 dd ?                             ; комірка для збереження _n2
    _n3 dd ?                             ; комірка для збереження _n3
    _n4 dd ?                    ; комірка для збереження _n4
    const1 dd 32                ; привласнення змінній значення 32
    const2 dd 64                ; привласнення змінній значення 64
    const3 dd 96                ; привласнення змінній значення 96
    const4 dd 128               ; привласнення змінній значення 128
    fName db "10_.txt", 0               ; комірки для ім'я файлу
fHandle DWORD ?                 ; резервування в пам'яті 32-розрядної
            ; комірки з ім'ям fHandle для дескриптора  збереження файлів
cWritten  DWORD  ?          ; резервування 32-розрядної комірки пам'яті
                          ; з ім'ям cWritten для адреси символів виведення
msg1 db "Кількість елементів:", 0ah, 0dh,\
 " 32 - %d",0ah,0dh,\
  " 64 - %d",0ah,0dh,\
   " 96 - %d",0ah,0dh,
    "128 - %d",0
    titl db "Результат програми",0
.code                           ; директива початку сегмента даних
  start:                        ; мітка початку програми
    lea esi, _mas               ; занесення адреси   mas
    mov ecx, len                ; лічильник чисел
```

```
cycle2:
      mov eax,[esi]                    ; занесення числа
  .IF eax==const1                      ; якщо число дорівнює 32
      inc _n1                          ; додати 1 у лічильник число 32
  .ELSEIF eax==const2                  ;
      inc _n2                          ; додати 1 у лічильник число 64
  .ELSEIF eax==const3                  ;
      inc _n3                          ; додати 1 у лічильник число 96
  .ELSE
      inc _n4                          ; додати 1 у лічильник число 32
  .ENDIF
      add esi,4h ; переведення вказівника на наступний елемент масиву
    loop cycle2
next:
invoke CreateFile, ADDR fName, ; адреса імені файлу з символами
 GENERIC_WRITE,                        ; запис у файл
  0, NULL,                              ; параметри багатозадачності
   CREATE_ALWAYS,                       ; знищити та створити новий файл
    FILE_ATTRIBUTE_ARCHIVE, 0
      mov   fHandle, eax           ; запам'ятовування дескриптора пристрою
invoke WriteFile, fHandle,         ; запис у файл за дескриптором пристрою
 ADDR _n1,                      ; адреса області пам'яті, що зберігає символи
  16,                           ; кількість символів
   ADDR cWritten, 0      ; адреса для кількості записаних у файл символів
      invoke  CloseHandle, fHandle              ; закрити дескриптор файлу
      mov eax, _n1;
      mov ebx, _n2;
      mov ecx, _n3;
      mov edx, _n4;
      invoke wsprintf, ADDR msg1,ADDR msg1,eax, ebx, ecx, edx
      invoke MessageBox,0,ADDR msg1,ADDR titl,MB_ICONINFORMATION ;
invoke  ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
      end start            ; директива закінчення програми з іменем start
```

У програмі функція wsprintf перетворює числа в символи відразу всіх чотирьох регістрів, які потім виводяться функцією MessageBox у спрощене віконце (рис. 3.7.1).

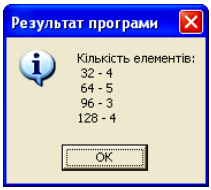


Рис. 3.7.1. Результат виконання
програми з лістингу 3.7.1

Лістинг 3.7.2. Програма запуску файлу з іменем 14_1.exe:

```
        .386
        .model flat, stdcall
        option casemap :none
         include \masm32\include\windows.inc
        include \masm32\macros\macros.asm
        uselib kernel32, user32, shell32
.data
        oper db "c:\masm32\bin\14_1.exe",0
        open db "open",0
.code
start:
        invoke ShellExecute, 0, addr open, addr oper,0,0, SW_SHOWNORMAL
        invoke ExitProcess, 0
        end start
```

3.8. Лабораторна робота 8
ОБРОБКА ДАНИХ У СПІВПРОЦЕСОРІ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням команд **співпроцесора**, написаних мовою асемблера для програмування МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера в групі вибрати варіант завдання та написати на асемблері програму обчислення одного з виразів з використанням дійсних чисел та виведенням їх на екран. При цьому обов'язково використати ***спрощені віконця*** з повідомленнями: в одному віконці вивести своє прізвище та номер навчальної групи; при натисканні на другу кнопку спрощеного віконця – вивести умову завдання. Використання інших кнопок спрощеного віконця виведення повідомлень – за бажанням.

Кожне завдання виконати двома способами:

1) з виконанням команд співпроцесора;

2) з використанням вбудованих функцій masm32.

Методичні рекомендації

Для налагодження остаточного варіанта програми спочатку необхідно налагодити частину програми з отриманням одного результату та його виведенням. На другому етапі – отримання масиву результатів. На третьому – виведення всього масиву.

Завдання 1. Дослідити виконання арифметичних операцій співпроцесора. Результат вивести на екран функцією MessageBox з горизонтальним (у рядок) розташуванням результатів.

1. Обчислити 6 значень функції $Y_n = 25x^3 - 2{,}1$ (x змінюється з кроком 0.2).

2. Обчислити 5 значень функції $Y_n = 7x^3/(2x^2 + 1{,}6)$ (x змінюється від 1 з кроком 4).

3. Обчислити 4 значення функції $Y_n = 37/(2x^2 + 7{,}3)$ (x змінюється від 1 з кроком 2).

4. Обчислити 5 значень функції $Y_n = 5{,}1x^2 + 5{,}3$ (x змінюється від 4,7 з кроком 3). Результат округлити до найближчого цілого числа та вивести на екран.

5. Обчислити 6 значень функції $Y_n = 4x/(x + 5)$ (x змінюється від 3 з кроком 1,25). Результат округлити до цілого числа.

6. Обчислити 4 значення функції: $Y_n = 125/(3x^2 - 1{,}1)$ (x змінюється від 3 з кроком 1,5). Результат округлити в меншу сторону.

7. Обчислити 4 значення функції $Y_n = 2{,}5x^2 - 3{,}2$ (x змінюється від 4 з кроком 1).

8. Обчислити 3 значення функції $Y_n = 25x^2 + 2{,}1$ (x змінюється від 3 з кроком 2,5).

9. Обчислити 4 значення функції $Y_n = 150/(x^2 - 7)$ (x змінюється від 2 з кроком 3,1).

10. Обчислити 6 значень функції $Y_n = 256/(3x^2 + 31)$ (x змінюється від 2 з кроком 3).

Завдання 2. Дослідити виконання операцій порівняння співпроцесора.

1. Знайти перше значення аргументу функції $Y = 7(x + 0{,}3)$, при якому молодші цілі цифри результату виконання функції дорівнюватимуть 15 (x змінюється від 2 з кроком 3,5).

2. Знайти ціле значення аргументу, при якому функція $Y = 10/(x^3 + 1{,}7)$ стане менше 0,3 (x змінюється від 2 з кроком 2,5).

3. Знайти значення x, при якому функція $Y = 3^x/4 - 6$ дорівнюватиме 12,5.

4. Знайти значення x, при якому виконується функція $8\text{arctg}0{,}1 + \text{arctg}x = \pi/4$.

5. Визначити номер (x) елемента функції $x_n = 2^x + 5$, при якому сума елементів перевищить 12 000.

6. Знайти значення x, при якому функція $\lg(2^x + 3)$ буде більше 2,5.

7. Знайти ціле значення аргументу, при якому функція $Y = 5{,}6^x/3x^2$ перевищить 125.

8. Знайти ціле значення аргументу, при якому функція $Y = 2^x + 30$ перевищить 100. Р

9. Знайти перше значення аргументу функції $Y = 9(x^2 + 0{,}6)$, при якому молодші цілі цифри результату виконання функції дорівнюватимуть 12 (x змінюється від 3 з кроком 5,5).

10. Знайти ціле значення аргументу, при якому функція $Y = 2^x/5 + 4$ перевищить 400.

Завдання 3. Дослідити виконання команд співпроцесора з логарифмічними і показовими функціями. Результат вивести на екран функцією MessageBox з вертикальним (у стовпчик) або горизонтально-вертикальним розташуванням результатів з підписом кожного елемента результату.

1. Обчислити 5 значень функції $Y = 5{,}2 * \ln(\sin x)$ (x змінюється в градусах з кроком 1,5).
2. Обчислити 4 значення функції $Y = 5^x + \cos x$ (x змінюється від 0,4 з кроком 0,15).
3. Обчислити 6 значень функції $Y = 4{,}5 \lg(\operatorname{tg} x)$ (x змінюється в градусах з кроком 10).
4. Обчислити 6 значень функції $Y = 12^x - \sin x$ (x змінюється з кроком 0,05).
5. Обчислити 8 значень функції $Y = 3{,}3 * \log_2(x^2 + 1)$ (x змінюється з кроком 0,3).
6. Обчислити 4 значення функції $Y = 5{,}1(\sin x)$ (x змінюється в градусах з кроком 8,5).
7. Обчислити 6 значень функції $Y = \lg(x^2 + \sqrt{x})$ (x змінюється з кроком 0,3).
8. Обчислити 5 значень функції $Y = 4{,}3(x^2 + 1)$ (x змінюється з кроком 1,7).
9. Обчислити 6 значень функції $Y = 6{,}2 \lg(\cos x)$ (x змінюється у градусах з кроком 1,5).
10. Обчислити 7 значень функції $Y = 3{,}1 + 2/\cos x$ (x змінюється у градусах від 10 з кроком 8).

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу.
3. Лістинг програми з детальним коментарем та описом роботи.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.8.1. Обчислити 3 значення функції $Y_n = 5x^2/1{,}1$ (x змінюється від 1 з кроком 1). Результат розмістити в пам'яті та вивести на екран.

Алгоритм розв'язання прикладу 3.8.1 наведено на рис. 3.8.1.

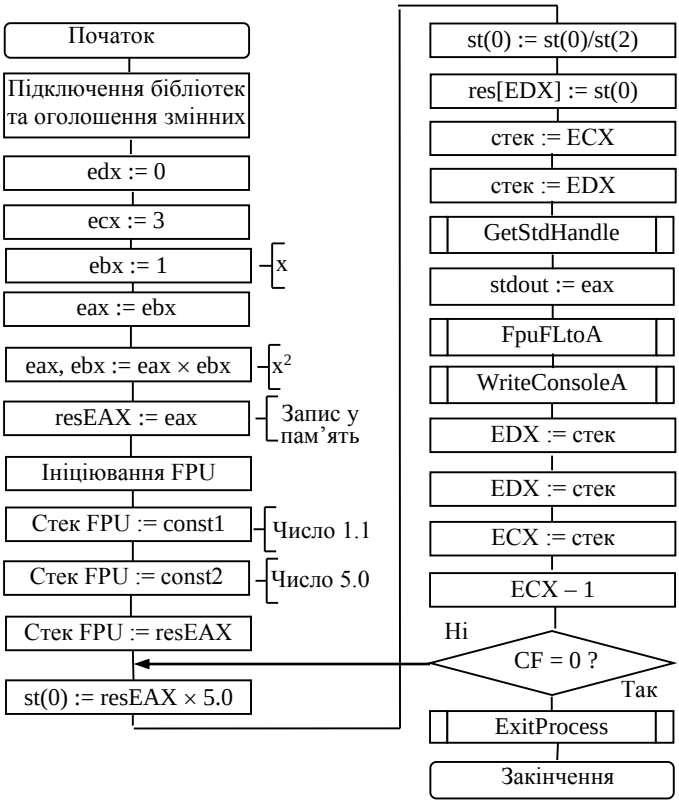


Рис. 3.8.1. Алгоритм розв'язання прикладу 3.8.1

У зв'язку з тим, що необхідно вивести три числа, для розділу комірок використовується додатковий елемент (регістр EDX). Спочатку цей регістр дорівнює нулю. Співпроцесор записує в пам'ять 80-розрядні числа (десять байтів). Тому в циклі виконується команда

```
add edx,0Ah
```

для виконання зміщення відносно попередньої комірки, яка потім буде адресуватися як `res[edx]`.

У прикладі при отриманні x^2 використовуються цілі числа. Тому необхідно виконати команду

```
fild dword ptr resEAX.
```

Якщо виконати команду `fld …` , то в даному випадку результат буде з похибкою.

А оскільки API-функції при своєму виконанні змінюють регістри, то значення тих регістрів, які потрібні при виконанні програми, необхідно зберігати (найкраще – в стеку).

Лістинг 3.8.1. Програма виконання прикладу 3.8.1:

```
      title CopyRight by Rysovaniy A. N.        Обчислення Yn = 5x2 / 1,1
      .686                          ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті  та угоди ОС Windows
      option casemap:none                   ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\kernel32.inc ; систем. функції застосувань…
include \masm32\include\fpu.inc
      includelib \masm32\lib\user32.lib
      includelib \masm32\lib\kernel32.lib
      includelib \masm32\lib\fpu.lib
      BSIZE equ  30                                 ; розмір буфера
  .data                                             ; директива визначення даних
  const1  DWORD 1.1           ; визначення в 32-розрядній комірці числа 1.1
  const2  DWORD 5.0           ; визначення в 32-розрядній комірці числа 5.0
resEAX DWORD 0       ; 32-розрядна комірка для тимчасового результату
res TBYTE 3 dup (0)    ; три 80-розрядні комірки для результатів
stdout DWORD ?               ; резервування в пам'яті 32-розрядної комірки
                        ; з ім'ям stdout для збереження дескриптора виведення
cWritten  DWORD  ?           ; резервування 32-розрядної комірки пам'яті
                       ; з ім'ям cWritten для адреси символів виведення
buf BYTE BSIZE dup (?); кількість байтів для запису символів
  .code                              ; директива початку коду програми
  _st:                                 ; мітка початку програми
      xor edx,edx                   ; початкове значення (для зміщення)
      mov ecx,3                     ; лічильник циклів
      mov ebx,1               ; початкове значення x
m1: mov eax,ebx                        ; підготування до піднесення в степінь
      mul ebx                 ; піднесення в степінь
      mov resEAX,eax      ; збереження x2 в пам'яті
    finit                     ; ініціювання співпроцесора
      fld dword ptr const1              ; st(0) := 1.1
      fld dword ptr const2              ; st(0) := 5.0, st(1) := 1.1
      fild dword ptr resEAX             ; st(0) := resEAX, st(1) := 5.0, st(2) := 1.1
      fmul st,st(1)                     ; st(0) := resEAX x 5.0
      fdiv st,st(2)                     ; st(0) :=  st(0)/st(2)
      fstp res[edx]                   ; збереження результату в комірці res[edx]
      push ecx                                ; підготовка до виклику API-функції
      push edx                                ; підготовка до виклику API-функції
  invoke GetStdHandle, STD_OUTPUT_HANDLE ; отримання дескриптора
    mov stdout, eax           ; збереження одержаного дескриптора у пам'яті
invoke  FpuFLtoA, \             ; функція перетворення 80-розрядного числа
```

```
  ADDR res[edx], \                    ; адреса числа, що відображається
    10, \                             ; кількість десяткових знаків після коми (10)
     ADDR buf, \              ; адреса буфера для символів, які перетворяться
SRC1_REAL or SRC2_DIMM                ; операнд SRC1в пам'яті
                                      ;  операнд SRC2 з параметром DIMM
 invoke  WriteConsoleA, \             ; API-функція виведення на екран
  stdout, \              ; дескриптор стандартного пристрою виведення
     ADDR buf, \                      ; адреса початку повідомлення
       BSIZE, \                       ; розмір повідомлення
         ADDR cWritten, 0         ; адреса, де зберігається число символів
       pop edx                        ; встановлення edx
       add edx,0Ah                    ; підготовка до вибору наступної комірки
       inc ebx                        ; збільшення x
       pop ecx                        ; встановлення ecx
     loop m1                          ; перейти, якщо не нуль
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
       end _st                ; директива закінчення програми
```

На рис. 3.8.2 наведено результат виконання лістингу 3.8.1.

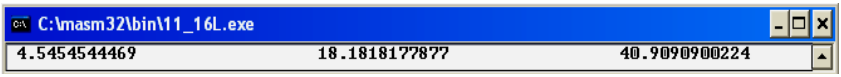


Рис. 3.8.2. Результат виконання лістингу 3.8.1

При налагодженні програми в налагоджувачі слід приділити увагу тому, як виглядають та виконуються API-функції (рис. 3.8.3).

При наведеній послідовності виконання команд виведення відбувається в один рядок. Якщо поставити відповідні параметри у функції WriteConsoleA, то можна переводити виведення в декілька рядків.

Для того щоб самостійно не рахувати кількість байтів, яку займає змінна, наприклад, в команді add edx,0Ah, простіше використати команду add edx, TYPE res.

Для виведення чисел з пам'яті використовується функція FpuFLtoA, яка перетворює **80-розрядні** числа в символи. Тому й комірки пам'яті треба визначати як 80-розрядні, наприклад:

res **TBYTE** 3 dup (0).

Крім того, в параметрах функції треба вказати, що дані виводяться з пам'яті. Для цього служить параметр SRC1_**REAL**.

Наведена програма з лістингу 3.8.1 може бути написана виключно за допомогою вбудованих функцій masm32. Таку програму наведено в лістингу 3.8.2.

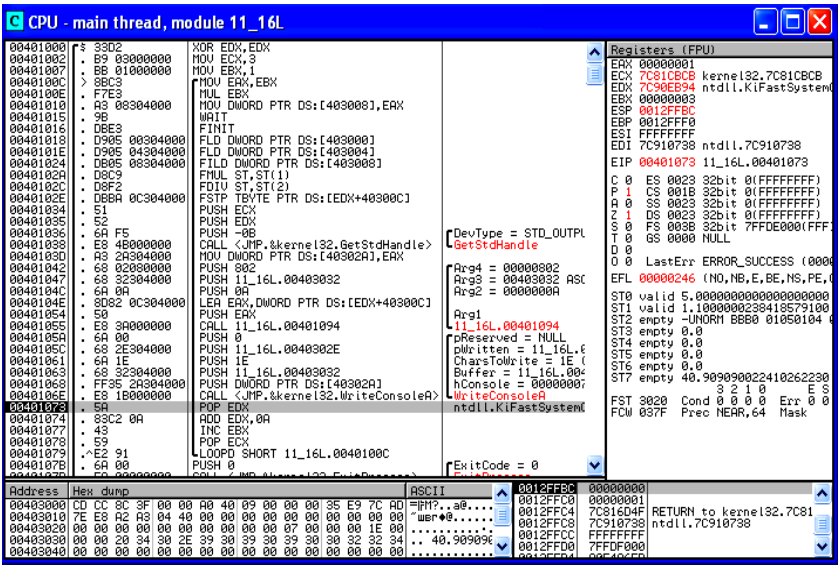


Рис. 3.8.3. Програма з лістингу 3.8.1 у вікнах налагоджувача OllyDbg

Лістинг 3.8.2. Програма виконання прикладу 3.8.1 з використанням вбудованих функцій masm32:

```
        title CopyRight by Rysovaniy A. N.          Обчислення Yn = 5x^2 / 1,1
; використання вбудованих функцій masm32
        .686                        ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
        option casemap:none                 ; відмінність малих та великих літер
include  \masm32\include\windows.inc ; файли структур, констант …
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
        include    \masm32\include\fpu.inc
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
        includelib \masm32\lib\fpu.lib
        BSIZE equ  30                       ; розмір буфера
.data                                       ; директива визначення даних
const1  DT 1.1                      ; визначення в 32-розрядній комірці числа 1.1
const2  DT 5.0                      ; визначення в 32-розрядній комірці числа 5.0
res TBYTE 3 dup (0)                     ; три 80-розрядні комірки для результатів
stdout DWORD ?                      ; резервування в пам'яті 32-розрядної комірки
                        ; з ім'ям stdout для збереження дескриптора виведення
cWritten  DWORD  ?                  ; резервування 32-розрядної комірки пам'яті
```

```
                                    ; з ім'ям cWritten для адреси символів виведення
buf BYTE BSIZE dup (?) ; кількість байтів для запису символів
.code                                    ; директива початку коду програми
_st:                                     ; мітка початку програми
      xor edx,edx                        ; початкове значення (для зміщення)
      mov ecx,3                          ; лічильник циклів
      mov ebx,1
m1: mov eax,ebx                          ; початкове значення x
  invoke FpuMul, eax, ebx, 0, SRC1_DIMM or SRC2_DIMM or DEST_FPU
invoke FpuMul, ADDR const2, 0, 0, SRC1_REAL or SRC2_FPU or DEST_FPU
invoke FpuDiv, 0,ADDR const1,ADDR res[edx],
                                    SRC1_FPU or SRC2_REAL or DEST_MEM
      push ecx                       ; підготовка до виклику API-функції
      push edx                       ; підготовка до виклику API-функції
invoke GetStdHandle, STD_OUTPUT_HANDLE ; отримання дескриптора
   mov stdout, eax                   ; збереження одержаного дескриптора у пам'яті
invoke  FpuFLtoA, \                  ;  функція перетворення 80-розрядного числа
  ADDR res[edx], \                   ; адреса числа, що відображається
   10, \                         ; кількість десяткових знаків після коми (10)
    ADDR buf, \                  ; адреса буфера для символів, які перетворяться
    SRC1_REAL or SRC2_DIMM             ; операнд SRC1в пам'яті
                                ;  операнд SRC2 з параметром DIMM (число)
invoke  WriteConsoleA, \             ; API-функція виведення на екран
  stdout, \                  ; дескриптор стандартного пристрою виведення
   ADDR buf, \                   ; адреса початку повідомлення
    BSIZE, \                         ; розмір повідомлення
    ADDR cWritten, 0                 ; адреса, де зберігається кількість символів
      pop edx                        ; встановлення edx
      add edx,0Ah                ; підготовка до вибору наступної комірки
      inc ebx                                ; збільшення x
      pop ecx                                ; встановлення ecx
      dec ecx
      jnz m1                             ; перейти, якщо не нуль
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
      end _st                            ; директива закінчення програми
```

Приклад 3.8.2. Обчислити 6 значень функції: Y = cos *x*, де *x* змінюється в градусах від 10 з кроком 8.

Лістинг 3.8.3. Програма виконання прикладу 3.8.2:

```
      title Rysovaniy A.N.                        rysov@rambler.ru
; Y = cosx, x змінюється в градусах від 10 з кроком 8.
      .686                  ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
      option casemap:none         ; відмінність малих та великих літер
      include \masm32\include\windows.inc ; файли структур, констант …
```

```
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
        include \masm32\include\fpu.inc
        include \masm32\include\user32.inc       ; для MessageBox
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
        includelib \masm32\lib\fpu.lib
        BSIZE equ  30                    ; розмір буфера
.data                                            ; директива визначення даних
        const  DWORD 0.01745         ; (π = 3,14)/180 – градуси в радіанах
        i1  DWORD 10                          ; початкове значення градусів
        st1 db "         y=cosx",0
        st2 db "Автор: Рысованый А.Н.",0
         st3 db "        Прощание",0
         st4 db "До свидания",0
          st6 db "До следующей лабораторной работы !!!",0
        res TBYTE 6 dup (0)  ; шість 80-розрядних комірок для результатів
stdout DWORD ?       ; резервування в пам'яті 32-розрядної комірки
                ; з ім'ям stdout для збереження дескриптора виведення
cWritten  DWORD  ?    ; резервування 32-розрядної комірки пам'яті
                        ; з ім'ям cWritten для адреси символів виведення
buf BYTE BSIZE dup (?); кількість байтів для запису символів
.code                                    ; директива початку коду програми
  start:                          ; директива початку коду програми
        xor edx,edx              ; початкове значення (для зміщення)
        mov ecx,6                ; лічильник циклів
      finit                          ; ініціювання співпроцесора
@@: fild dword ptr i1      ; завантаження в st(0) змінної i1 (градусів)
        fmul dword ptr const ; st(0) := st(0) × const
        fcos                          ; st(0) := cos(i1) в радіанах
        fstp res[edx]               ; збереження результату в комірці res[edx]
        push ecx                          ; підготовка до виклику API-функції
        push edx                          ; підготовка до виклику API-функції
   invoke GetStdHandle, STD_OUTPUT_HANDLE  ; отримання дескриптора
      mov stdout, eax           ; збереження одержаного дескриптора у пам'яті
invoke  FpuFLtoA, \          ; функція перетворення 80-розрядного числа
  ADDR res[edx], \           ; адреса числа, що відображається
     10, \                     ; кількість десяткових знаків після коми (10)
     ADDR buf, \              ; адреса буфера для символів, які перетворяться
SRC1_REAL or SRC2_DIMM    ; адреса 80-розрядного числа в пам'яті
                                                          ; та саме число
 invoke  WriteConsoleA, \    ; API-функція виведення на екран
  stdout, \                ; дескриптор стандартного пристрою виведення
     ADDR buf, \                    ; адреса початку повідомлення
       BSIZE, \                     ; розмір повідомлення
         ADDR cWritten, 0    ; адреса, де зберігається число символів
        pop edx                  ; встановлення edx
```

```
        add edx,0Ah             ; підготовка до вибору наступної комірки
        add i1,8                ; збільшення градусів на  крок дискретизації
        pop ecx                         ; встановлення ecx
        loop @b                         ; перейти, якщо не нуль
invoke MessageBox, \   ; API-функція виведення спрощеного вікна консолі
  NULL, \               ; hwnd – ідентифікатор вікна
     addr st2, \        ; адреса рядка, яка містить  текст повідомлення
       addr st1, \      ; адреса рядка, яка містить  заголовок повідомлення
          MB_YESNO or MB_ICONINFORMATION   ; вигляд діалогового вікна
       xor eax,6          ; виділення розрядів коду повернення кнопки YES
       jz m1
invoke MessageBox, NULL, addr st6, addr st3, MB_ICONWARNING
       jmp m2
m1:
invoke MessageBox, NULL, addr st4, addr st3, MB_ICONERROR
m2:
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
       end start        ; директива закінчення програми з іменем start
```

У цій програмі команда з унікальною міткою @@: fild означає, що до неї звертається у циклі команда, яка розташовується після неї в тексті та має символ з літерою @b. Такі мітки можна змінити на звичайні.

Результат виконання програми з лістингу 3.8.3 у вигляді вікна наведено на рис. 3.8.4.

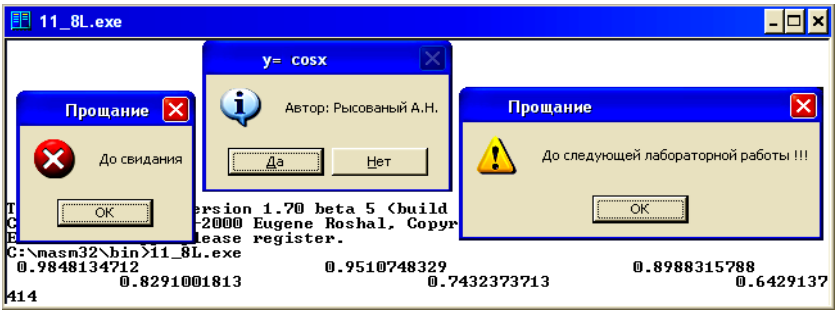


Рис. 3.8.4. Вікно результату виконання програми з лістингу 3.8.3

Константи NULL та MB_OK, використані у функції Messagebox, покращують читабельність коду. До них можна безпосередньо звертатися в програмі.

Оператор addr використовується для передачі адреси мітки функції. Він дійсний тільки в контексті директиви invoke.

Приклад 3.8.3. Обчислити 3 значення функції $Y_n = 25x^2 + 2,1$ (x змінюється від 3 з кроком 2,5). Результат розмістити в пам'яті та вивести на екран.

Лістинг 3.8.4:

```
title CopyRight by КІТ-29б Вернидуб А. В., НТУ "ХПІ"
; Обчислити 3 значення функції Yn = 25x^2 + 2,1
                                                    ; (x змінюється від 3 з кроком 2,5)
include \masm32\include\kernel32.inc ; файли систем. функцій застосувань
      include C:\masm32\include\\user32.inc   ; файли інтерфейсу …
      include C:\masm32\include\fpu.inc
      includelib C:\masm32\lib\kernel32.lib
      includelib C:\masm32\lib\user32.lib
      includelib C:\masm32\lib\fpu.lib
.data
ttl  db " Обчислити 3 значення функції Yn = 25^2 + 2,1 (x змінюється від 3 з
кроком 2,5)",0
            CrLf equ 0A0Dh
           _y1 dt 0.0
           _y2 dt 0.0
           _y3 dt 0.0
           _x DWORD 3.0
           _op1 DWORD 25.0
           _op2 DWORD 2.1
           _zero DWORD 0.0
           _step DWORD 2.5
           info db "       y1 = "
           _res1 db 14 DUP(0),10,13
           db  "       y2 = "
           _res2 db 14 DUP(0),10,13
           db  "       y3 = "
           _res3 db 15 DUP(0),0
      .code
      _start:
            finit
            mov ecx, 3
      m1:   fld _x
           fmul _x
           fmul _op1
           fadd _op2
           fld _x
           fadd _step
           fstp _x
      loop m1
           fstp _y3
           fstp _y2
```

```
        fstp _y1
invoke FpuFLtoA,offset _y1,10,offset _res1,SRC1_REAL or SRC2_DIMM
        mov word ptr _res1 + 14, CrLf
invoke FpuFLtoA,offset _y2,10,offset _res2,SRC1_REAL or SRC2_DIMM
        mov word ptr _res2 + 14, CrLf
invoke FpuFLtoA,offset _y3,10,offset _res3,SRC1_REAL or SRC2_DIMM
invoke MessageBox, 0, offset info, ADDR ttl, MB_ICONINFORMATION
invoke  ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
    end _start ; закінчення програми
```

Спрощене вікно результату виконання програми з вертикальним виведенням чисел з лістингу 3.8.4 наведено на рис. 3.8.5.

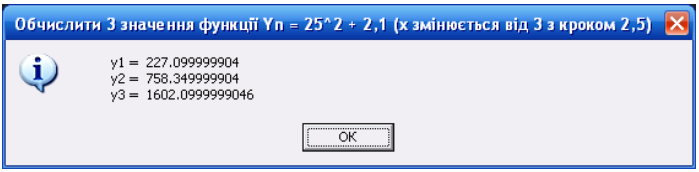


Рис. 3.8.5. Спрощене вікно результату виконання програми з лістингу 3.8.4

Приклад 3.8.4. Знайти ціле значення аргументу, при якому функція $Y = 10/(x^2 + 1{,}4)$ стане менше 0,2 (x змінюється від 3 з кроком 2,5). Результат розмістити в пам'яті та вивести відповідні повідомлення.

Лістинг 3.8.5:

```
    title CopyRight by Rysovaniy A. N.  and Beletsky K.A., НТУ "ХПІ", КІТ - 37a
        .686                            ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
        option casemap:none             ; відмінність малих та великих літер
        ; y = 10/(x^2 + 1.4)< 0.2, x  =3, s_x = 2.5
include\masm32\include\windows.inc ; файли структур, констант …
include\masm32\include\fpu.inc
 include\masm32\include\kernel32.inc;файли систем. функцій застосувань…
        include\masm32\include\\user32.inc   ; файли інтерфейсу …
        includelib\masm32\lib\fpu.lib
        includelib\masm32\lib\kernel32.lib
        includelib\masm32\lib\user32.lib
.data                                   ; директива визначення даних
        y    dd ?
        x    dd 3.0
        c1   dd 0.2
        c2   dt 1.4
        c3   dd 10
        c4   dt 2.5
```

```
      temp  dd ?
      st1  db "y=10/(x^2 + 1.4)", 0
      st2  db "y =              x =", 0 ;
.code                         ; директива початку коду програми
_st:                          ; директива початку коду програми
          xor edx, edx          ; комірка для зміщення при виведенні у віконце
          finit               ; ініціювання співпроцесора
          fld c4              ; st(0) = 2.5
 cycle:
          fild c3                       ; st(0) = 10
          fld x                         ; st(0) = x
          fld x                         ; st(0) = x
          fmulp st(1), st               ; st(0) = x^2
          fld c2                        ; st(0) = 1.4
          faddp st(1), st               ; st(0) = x^2 + 1.4
          fdivp st(1), st               ; st(0) = 10/(x^2 + 1.4)
          fst y                         ; занесення результату в пам'ять
          fcomp c1                      ; порівняння з умовою завдання
          fstsw ax                      ; занесення та
          sahf                          ; встановлення прапорців
          jb m1                         ;
          fld x                         ; занесення x = 3.
          fadd st, st(1)                ; збільшення кроку на 2.5
          fstp x                        ; збереження нового x
          jmp cycle                     ;
 m1:
          fld y                         ;
          add edx, 4                ; зміщення в st2 на 4  байти ("y =")
invoke FpuFLtoA, NULL, 10, ADDR st2[edx], SRC1_FPU or SRC2_DIMM
          add edx, 0dh              ; зміщення на 13 знакомісць під "y"
      ; mov byte ptr [st2[edx]], 20h   ; символ пропуску
      ; inc edx                       ; підготування адреси для запису 0dh
          mov byte ptr [st2[edx]], 0dh ; повернення курсора в початок рядка
          inc edx                     ; збільшення індексу адреси
          mov byte ptr [st2[edx]], 0ah ; переведення рядка
          inc edx                     ;
          fld x                       ;
          add edx, 4                  ;
 invoke FpuFLtoA, NULL, 2, ADDR st2[edx], SRC1_FPU or SRC2_DIMM
invoke MessageBox, NULL, ADDR st2, ADDR st1, MB_ICONWARNING
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
      end _st                       ; закінчення програми
```

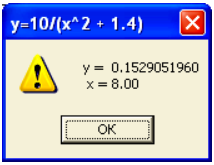


Рис. 3.8.6. Результат виконання програми з лістингу 3.8.5

Результат виконання програми з лістингу 3.8.5 наведено на рис. 3.8.6.

Особливістю програми є те, що виведення чисел результату виконане шляхом урахування адрес позицій всіх символів.

Приклад 3.8.5. Знайти значення x, при якому функція $Y = 2^x + 12 \times 2^{-x}$ дорівнюватиме 9,5. Результат розмістити в пам'яті та вивести відповідні повідомлення.

Для написання програми з початковою функцією зроблені такі перетворення:

$$t + 12\ (1/t) = 9{,}5;$$
$$t^2 - 9{,}5t + 12 = 0.$$

Останній вираз є квадратним рівнянням, для розв'язання якого в програмі обчислюється дескримінант кореня.

Лістинг 3.8.6:

```
   title CopyRight by Rysovaniy A. N. and Nezhurina Irina, НТУ "ХПІ", КІТ-27а
      .686                          ; директива визначення типу мікропроцесора
.model flat, stdcall    ; задання лінійної моделі пам'яті та угоди ОС Windows
      option casemap:none      ; відмінність малих та великих букв
      include C:/masm32/include/kernel32.inc ; бібліотеки, що підключаються
      include C:/masm32/include/\user32.inc   ; файли інтерфейсу …
      include C:/masm32/include/fpu.inc
      includelib C:/masm32/lib/kernel32.lib
      includelib C:/masm32/lib/user32.lib
      includelib C:/masm32/lib/fpu.lib
.data                                 ; директива визначення даних
      CrLf  equ  0A0Dh                ; перехід рядка і повернення каретки
      _x1 dt 0.0                      ; аргумент функції
      _x2 dt 0.0                      ; приріст аргументу
      _t1 dt 0.0                      ; тимчасові
      _t2 dt 0.0                        ; змінні
      _b dd -9.5                        ; службові
      _c dd 12.0                       ; константи
      _c2 dd 2.0                        ; для розв'язання
      _c4 dd 4.0                         ; квадратного рівняння
 info db "x1 = "              ; підпис першого числа результату
  buff1 db 14 DUP (0), 10, 13         ; рядок для виведення
   db "x2 = "                         ; підпис другого числа результату
    buff2 db 15 DUP (0), 0            ; рядок для виведення
      titl db "Y = 2^x + 12 2^-x", 0    ; назва спрощеного вікна
```

```
.code                                   ; директива початку коду програми
_st:                                    ; мітка початку програми
      finit                             ; ініціювання співпроцесора
; знаходимо дискримінант D = b*b - 4*a*c (a = 1,тому не пишемо)
      fld _c4                 ; st(0) <- _c4
      fmul _c                 ; st(0) < - st(0) * _c
      fld _b                  ; st(0) <- st(1)<- st(0), st(0) <- _b
      fmul _b                 ; st(0) < - st(0) * _x => st(0) = b*b
      fsub st, st(1)          ; st(0) = st(1) - st(0)
      fsqrt                   ; st(0) <- квадратний корінь від st(0)
; знаходження коренів від рівняння
      ; t1 = [-b - sqrt(D)] / 2*a              t2 = [-b + sqrt(D)] / 2*a
      fld _b                  ; st(1) <- st(0), st(0) <- _b
      fchs                    ; st(0) <- st(0) * (-1) команда заміни знака числа
      fld st                  ; st(2) <- st(1), st(1) <- st(0), st(0) <- st(0)
      fsub st, st(2)          ; st(0) = st (0) - st(2)
      fld _c2                 ; st(0) <- _c2, st(1) <- st(0), st(2) <- st(1)...
      fdiv                    ; st(0) <- st(1) / st(0)
      fstp _t1                ; збереження першого кореня
      fadd st, st(1)          ; віднімання
      fld _c2                 ; завантаження дільника
      fdiv                    ; ділення
      fstp _t2                ; збереження другого кореня
; 2^x = t,  x = ?
      ; x = log2 (t)
; для t1
      fld1                    ; st(0) <- 1
      fld _t1                 ; st(1) <- st(0), st(0) <- _t1
      fyl2x                   ; st(0) <- st(1) * log2 [st(0)]
      fstp _x1                ; збереження результату
; для t2
      fld1                    ; st(0) <- 1
      fld _t2                 ; st(1) <- st(0), st(0) <- _t2
      fyl2x                   ; st(0) <- st(1) * log2 [st(0)]
      fstp _x2                ; збереження результату
      invoke FpuFLtoA,       ; переведення числа в текстовий рядок
       offset _x1,             ; вказівник на _x1
        12,                     ; кількість цифр після коми у числі
          offset buff1,         ; вказівник на рядок результату
           SRC1_REAL or SRC2_DIMM ; 1-е число – в пам'яті, 2-е – число
      mov word ptr buff1 + 14, CrLf                  ; переведення рядка
invoke FpuFLtoA,offset _x2,12,offset buff2,
   SRC1_REAL or SRC2_DIMM       ; переведення числа у текстовий рядок
invoke MessageBox,0,offset info,offset titl,0      ; виведення повідомлення
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
      end _st                           ; закінчення програми
```

Спрощене вікно результату виконання програми з лістингу 3.8.6 наведено на рис. 3.8.7.

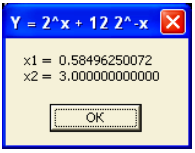


Рис. 3.8.7. Спрощене вікно результату
виконання програми з лістингу 3.8.6

Приклад 3.8.6. Обчислити значення функції <ln[sqrt(a+x*x)]> для 24 значень змінної "x" та порівняння їх із заданою константою. Якщо значення перевищує її – видати вікно з помилкою.

Лістинг 3.8.7:

```
        .686                ; директива визначення типу мікропроцесора
        .model flat,stdcall                 ; задання лінійної моделі пам’яті
        option casemap:none                 ; відмінність малих та великих літер
        include C:\masm32\include\windows.inc   ; підключення
        include C:\masm32\include\kernel32.inc  ;       бібліотек
        include C:\masm32\include\\user32.inc   ; файли інтерфейсу …
        include C:\masm32\include\fpu.inc       ;
        includelib C:\masm32\lib\kernel32.lib   ;
        includelib C:\masm32\lib\user32.lib     ;
        includelib C:\masm32\lib\fpu.lib        ;
.data                       ; директива визначення даних
_temp dt 0     ; резервування пам’яті для очищення стека співпроцесору
_st0 db "ERROR",0
_st1 db "<Летучка 4 - запрограмувати ln(sqrt(a+x*x)) для 24 значень x > ",0
_buf db "Один з результатів обчислень функції <ln(sqrt(a+x*x))>  перевищує
заданий кордон",0
_st2 db 300 dup(?),0                ; резервування буфера
_x dd 954.6,1.0,54.1,3.0,7.8,11.7,6.0,547.98,32.0,87.9,9.8,1.8,54.7,3.06,7.86,
11.75,6.67,547.9,332.07,89.9,21.9,246.09,3.7,8.0
_a dd 39.0
_rez dt 24 dup(0)                   ; резервування пам’яті
_len dd 24                          ; кількість елементів результату
_c dd 7.0 ; константа, з якою порівнюються результати
.code                               ; початок сегмента даних
_start:                             ; мітка початку програми
        lea edi,_rez     ; завантаження початкової адреси _rez у пам’яті
        lea esi,_x       ; завантаження початкової адреси масиву _x у пам’яті
```

```
      mov ecx,_len            ; завантаження лічильника у ecx
       finit                  ; ініціювання співпроцесору
       xor edx,edx            ; індекс адреси
m1:
  fld dword ptr [esi] ; завантаження першого елемента масиву _x у стек
   fmul dword ptr [esi]       ; st=X*X
   fadd _a                    ; st=X*X+a
   fsqrt                      ; st=sqrt(X*X+a)
   invoke FpuLnx,0,0,SRC1_FPU or DEST_FPU  ; st=ln[sqrt(a+x*x)
fld dword ptr _c  ; st=c,st(1)=ln[sqrt(a+x*x)
     clc                      ; очищення ознаки Carry
     fcomi st(0),st(1)        ; порівняння значення функції та константи
     jc error  ; перехід на error, якщо значення функції більше ніж константа
     fcmovnb st(0),st(1)   ; повернення на вершину стека  значення функції
fstp tbyte ptr [edi]       ; запис значення функції за адресою е EDI
fstp tbyte ptr _temp            ; очищення другої комірки стека
invoke FpuFLtoA,ADDR [edi],7,
addr _st2[edx],SRC1_REAL or SRC2_DIMM ; перетворення з точністю 7
  add edx,0ah                   ; підготування EDX
  mov byte ptr _st2[edx],0dh    ; переведення курсора у початок рядка
  mov byte ptr _st2[edx],20h    ; запис знака пропуску
  inc edx                      ; збільшення індексу адреси
mov byte ptr _st2[edx],20h   ; запис другого знака пропуску
  inc edx                      ; збільшення індексу адреси
  cmp ecx,7                    ; перевірка лічильника,
  jz m2                        ;      якщо 7, то перехід у переведення рядка
  cmp ecx,13                   ; перевірка лічильника,
  jz m2                        ;      якщо 13, то перехід у переведення рядка
  cmp ecx,19                   ; перевірка лічильника,
  jz m2                        ; якщо 19, то перехід у переведення рядка
  jmp m3                       ; перехід далі в основну програму
m2:                           ; початок процесу переведення рядка
  add byte ptr _st2[edx],0ah         ; підготування EDX
  mov byte ptr _st2[edx],0ah         ; переведення рядка
  inc edx                      ; збільшення індексу адреси
mov byte ptr _st2[edx],0ah ; повторне переведення рядка
  inc edx                      ; збільшення індексу адреси
m3:
   add edi,10          ; збільшення адреси у EDI (оскільки змінна DT = 10)
   add esi,4           ; збільшення адреси у ESI (оскільки змінна DD = 4)
   dec ecx             ; зменшення лічильника на 1
   jnz m1              ; перехід на m1, якщо ECX не 0
invoke MessageBox,0,addr _st2,\
    addr _st1,MB_ICONINFORMATION+150000h;
    jmp fin
error:
```

```
 invoke MessageBox,NULL,addr _buf,addr _st0,MB_ICONERROR+95000h;
fin:
  invoke ExitProcess,0        ; очищення ресурсів
      end _start                ; директива закінчення програми з ім'ям _start
```

Спрощені вікна результату виконання програми з лістингу 3.8.7 наведено на рис. 3.8.8 та рис. 3.8.9.

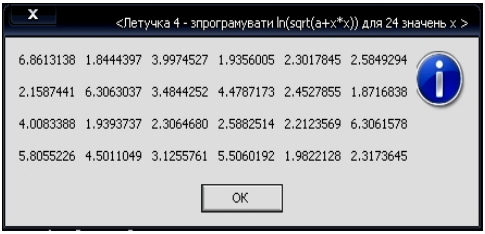


Рис. 3.8.8. Спрощене вікно результату виконання програми з лістингу 3.8.7

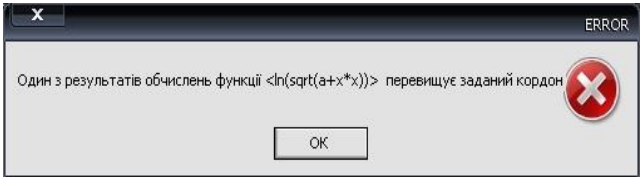


Рис. 3.8.9. Спрощене вікно виведення повідомлення про помилку

3.9. Лабораторна робота 9
РЯДКИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм, написаних мовою асемблера з використанням команд обробки рядків та API-функцій під Win32 для програмування МП платформи x86.

Постановка задачі

Згідно з цифрою номера студента в групі вибрати свій варіант та написати програму з використанням **команд обробки рядків**, з обов'язковим виведення у **файл** та й у **спрощене віконце**. Дані, які виводяться, необхідно підписати, вивести умову завдання та довідку про автора програми.

1. Задано текст, що складається з 6 слів по 8 символів, розділених пропуском. Переставити слова в тексті так, щоб кожне наступне слово починалося з тієї букви, на яку закінчилося попереднє. Перше слово залишити на місці.

2. Задано послідовність букв у вигляді прізвища студента. Розставити їх за абеткою.

3. Задано текст у вигляді прізвища, імені та по батькові, а також e-mail студента. Стиснути текст, залишивши між словами по одному пропуску.

4. Задано текст із 20 символів. Визначити кількість повторень поєднань "*ОР*" в тексті і замінити такі повторення символом "!"

5. Задано текст із 40 символів. Визначити кількість слів, які містять більше 4-х символів. Слова розділяються одним пропуском.

6. Задано текст із 8 слів з різною кількістю символів. У словах з парним номером змінити порядок букв на зворотний.

7. Побудувати послідовність з n ($n \le 30$) символів такого вигляду: *АББВВВГГГГДДДДДЕЕЕЕЕЕ...*

8. Задано текст із 32 символів, що складається із слів, розділених одним пропуском. Визначити кількість слів і кількість букв у кожному слові.

9. Задано текст із 34 символів, що складається із слів, розділених одним пропуском. Визначити кількість слів, в яких буква *Е* зустрічається більш ніж 2 рази.

10. Задано послідовність із 37 символів. Визначити частоту повторення кожного символу.

11. Задано 8 слів по 6 символів. На початку кожного слова записано номер з двох символів. Розставити слова за збільшенням номерів.

12. Задано текст із 15 слів, розділених пропуском. Визначити кількість слів, в яких буква *А* зустрічається більше 3-х разів.

13. Задано текст із 10 слів, розділених пропусками (одним і більше). Визначити кількість слів, що містять більше 4 символів.

14. Задано текст із 8 слів, розділених пропуском. Визначити кількість повторень букви *Е* в кожному слові.

15. Задано текст із 12 слів, розділених пропуском. Розставити слова відповідно до латинського алфавіту.

16. Задано текст із 26 символів. Визначити кількість різних символів і частоту їх повторень.

17. Задано текст, що складається з 5 слів по 7 символів. Розставити слова відповідно до російського алфавіту.

18. Задано текст, що складається з 10 слів, розділених деякою кількістю пропусків. Визначити кількість букв *А* в кожному слові.

19. Задано текст, що складається з 8 слів по 5 символів. Визначити кількість голосних букв у кожному слові.

20. Задано текст, що складається з 4 слів по 8 символів. Визначити кількість різних букв у кожному слові.

21. Задано текст, що складається з 7 слів по 5 символів. Видалити слова, що містять більше 3-х букв *О*.

22. Задано текст, розділений на слова пропусками. Змінити порядок букв у словах на протилежний.

23. Задано текст із 25 символів. Знайти слова, порядок букв в яких зворотний по відношенню до першого слова. (Слова розділені пропусками).

24. Задано текст із 18 символів: 3 слова по 6 символів. Здійснити кільцевий зсув кожного слова вліво: 1-го – на 1 символ; 2-го – на 2 символи; 3-го – на 3 символи.

25. Побудувати послідовність вигляду *АББССС* із 30 символів відповідно до заданої таблиці, що містить символ і кількість його повторень. Наприклад: *А* – 1 раз; *Б* – 2 рази; *С* – 3 рази і т.ін.

26. Задано текст з 30 символів, які можуть бути поділені пропусками та розділовими знаками. У кожному слові рядка замінити першу букву на останню, другу на передостанню і т.ін.

27. Ввести довільний символьний рядок. Розділити його на два рядки: рядок голосних і рядок приголосних. Перший рядок вивести на екран, а другу – у файл.

Зміст звіту

1. Постановка задачі для конкретного варіанта.

2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.

3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій та з детальним коментарем і описом роботи.

4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.

5. Короткий опис виконання програми.

6. Висновки за результатами роботи.

Приклад 3.9.1. Задано текст, розділений на слова пропусками. Змінити порядок букв у словах на протилежний.

Лістинг 3.9.1:

```
      .686                    ; директива визначення типу мікропроцесора
.model  flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none           ; відмінність малих та великих літер
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\\user32.inc   ; файли інтерфейсу …
includelib\masm32\lib\kernel32.lib
includelib\masm32\lib\user32.lib
      BSIZE equ 30                  ; задання реальної кількості байтів
.data                               ; директива визначення даних
      mas1 db 'abc def ghk lmn'
      mas2 db 15 dup(0)
cWritten DWORD ?          ; резервування 32-розрядної комірки пам'яті
                            ; з ім'ям cWritten для адреси символів виведення
stdout DWORD ?            ; резервування 32-розрядної комірки пам'яті
                    ; з ім'ям stdout для збереження дескриптора виведення
.code                               ; директива початку коду програми
_start:                             ; мітка початку програми з ім'ям _start
      cld                           ; напрям – вверх
      lea edi,mas1                  ; адреса першого масиву в edi
      mov esi, 0
      mov edx,0
      mov al,' '                    ; заносимо в al пропуск
      mov ecx,21
      push ecx
      jmp m2
m4:  mov ecx,4                      ; лічильник слів
m1: mov ah, [ebx]                   ; заносимо в ah останній символ масиву
```

```
        mov mas2[edx],ah                ; зберігаємо цей символ
        inc edx                         ; збільшуємо індекс
        dec ebx                         ; перехід на наступний елемент
        cmp [ebx],esi
        dec ecx
        jnz m1                  ;
m2:     pop ecx
        repne scasb             ; шукаємо пропуск у масиві
        mov ebx,edi             ; зберігаємо в ebx місце, де був знайдений пропуск
        dec ebx
        push ecx
        dec ecx
        jnz m4
invoke GetStdHandle, STD_OUTPUT_HANDLE   ; одержання дескриптора
        mov stdout,eax                  ; збереження одержаного дескриптора
invoke  WriteConsoleA, \                ; API-функція виведення на екран
 stdout, \                              ; дескриптор стандартного пристрою виведення
  ADDR mas2,\                           ; адреса рядка, яка містить  текст повідомлення
   sizeof mas2+1, \     ; адреса рядка, яка містить  заголовок повідомлення
    ADDR cWritten, 0      ; адреса, де зберігається кількість символів
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
end _start                              ; директива закінчення програми
```

Результат виконання програми з лістингу 3.9.1 наведено на рис. 3.9.1.

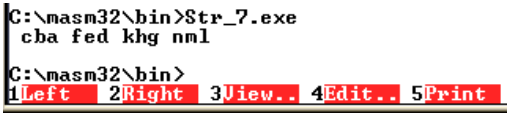


Рис. 3.9.1. Результат виконання програми з лістингу 3.9.1

Приклад 3.9.2. Побудувати послідовність вигляду ААААБББССССС із 40 символів відповідно до заданої таблиці, що містить символ і кількість його повторень.

Лістинг 3.9.2:

```
        title Рысованый А.Н.
      .686                      ;директива визначення типу мікропроцесора
      .model flat, stdcall              ;завдання лінійної моделі памяті
      option casemap:none   ;відмінність малих та великих літер
include \masm32\include\kernel32.inc ; файли систем. функцій застосувань
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\\user32.inc   ; файли інтерфейсу …
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
.data                           ; директива визначення даних
```

```
  info db "Масив = "                ; підпис числа для віконця результату
  mas BYTE 45 dup (?),0   ; визначення масиву
  c1 db 61h                         ; c1 := 61h = 97 – код літери “a”
  titl db " Результат ",0     ; назва віконця
.code                               ; директива початку коду програми
_start:                             ; мітка початку програми з ім’ям _start
  mov ebx,40       ; EBX := 28h
  mov edx,3        ; EDX := 3 – кількість початкових однакових літер
  lea edi,mas       ; занесення адреси початку рядка до EDI
m1:  mov al,c1                  ; al := c1
  mov ecx,edx                   ; ECX := EDX
  cld                           ; напрям – вверх
  rep stosb                     ; [di]<- al, заповнити [di] вмістом al
  mov ecx,edx                   ; ECX := EDX
  inc c1                        ; збільшення коду літери
  inc edx                       ; збільшення кількості повторень літери
  sub ebx,ecx      ; обчислення кількості повторень, які залишилися
  cmp ebx,0                     ; порівняння вмісту EBX з нулем
  jnl m1                        ; якщо не менше – перейти на m1
  jmp m2                        ; інакше – на m2
m2:
invoke MessageBox,0,\           ; API-функція виведення спрощеного віконця
    ADDR info,\                 ; адреса рядка, яка містить  текст повідомлення
     ADDR titl,\          ; адреса рядка, яка містить  заголовок повідомлення
      MB_ICONINFORMATION +180000h           ; вигляд діалогового вікна
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
       end _start                   ; директива закінчення програми
```

Результат виконання програми з лістингу 3.9.2 наведено на рис. 3.9.2.

Рис. 3.9.2. Результат виконання
програми з лістингу 3.9.2

У програмі початок перебору починається з коду літери *a*, а потім в циклі цей код збільшується, після чого підраховується кількість літер, яка не повинна перевищувати 40.

Приклад 3.9.3. Задані рядки A та B. Знайти перші незбіжні числа та переписати наступні числа рядка A в рядок C.

Лістинг 3.9.3:

```
        title Рысованый А.Н.                                rysov@rambler.ru
        .386            ; директива визначення команд мікропроцесора
        .model  flat,stdcall ; задання лінійної моделі пам'яті
        option casemap:none ; відмінність рядкових та прописних літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\kernel32.inc ; файли систем. функцій застосувань
include \masm32\include\\user32.inc   ; файли інтерфейсу …
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
.data                                   ; директива визначення даних
        mas1 dd 0Ah,0Bh,1,2,3,4         ; визначення масиву з ім'ям mas1
        mas2 dd 0Ah,0Ch,5,6,7,8         ; визначення масиву з ім'ям mas2
len equ ($-mas2)/ type mas2 ; обчислення кількості подвійних слів в mas2
        mas3 dd len DUP(?),0 ; резервування 32-розрядних комірок для mas3
        _title db "Результаты решения программы",0
info db "Заданы строки A и B. Найти первые несовпадающие числа", 0Ah, 0Dh,
"и переписать последующие числа строки A в строку C",0Ah,0Dh,0Dh
buf dd ?,0                              ; буфер виведення повідомлення
        ifmt  db "Ответ: mas3 =  %d  %d  %d  %d  %d  %d",0
.code                                           ; початок сегмента-даних
_st:                                            ; мітка початку програми
        mov ecx,len                     ; кількість слів у масиві
        lea esi, mas1           ; завантаження адреси масиву mas2
        lea edi, mas2           ; завантаження адреси масиву mas2
        cld                     ; напрям – вверх
        repne cmpsd             ; [esi] - [edi] поки не дорівнює
        jz ravno                ; перехід, якщо source = destination
        jmp exit
ravno:
        lea edi,mas3            ; завантаження адреси масиву mas3
        dec ecx                 ;
        add esi,4               ; підготовка вибірки наступного числа
        rep movsd                       ; edi<- esi
invoke  wsprintf, \                     ; API-функція перетворення чисел
  ADDR buf, \     ; адреса буф., куди буде записана послідовність символів
   ADDR ifmt, \             ; адреса рядка перетворення формату
     mas3,mas3[4],mas3[8],mas3[12],mas3[16],mas3[20] ;
invoke MessageBox, NULL, addr info, addr _title, MB_ICONINFORMATION
exit:
        invoke ExitProcess,0
        end _st      ; директива закінчення програми з ім'ям _st
```

У програмі порівнюються два масиви: A та B. При порівнянні чисел mas1: 0Bh та mas2: 0Ch виявляється, що числа не дорівнюють один

одному. Тому слід наступні числа рядка A (1, 2, 3, 4) переписати в рядок C (mas3). Для цього використовується команда add esi,4 збільшення адреси числа mas1.

Для того щоб числа при виведенні функцією MessageBox не розташовувалися один біля одного, то у рядку форматування даних

ifmt db "Ответ: mas3 = %d %d %d %d %d %d",0

зроблено по два пропуски (рис. 3.9.3).

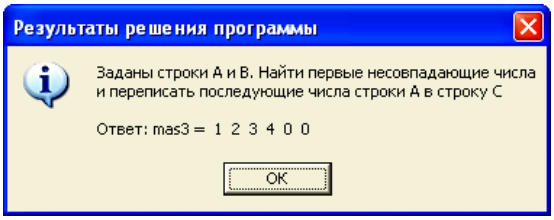


Рис. 3.9.3. Результат виконання програми з лістингу 3.9.3

3.10. Лабораторна робота 10
МАКРОСИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм для програмування макросів, написаних мовою асемблера для МП платформи x86 з організацією виведення результату на екран з використанням API-функцій під Win32.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати на асемблері програму обчислення одного з виразів з використанням макросів та виведенням їх на екран. При цьому обов'язково використати ***спрощені віконця*** з повідомленнями: в одному віконці вивести своє прізвище, номер своєї навчальної групи та своєї фотографії в якості іконки; при натисканні на другу кнопку спрощеного віконця – вивести умову завдання. Використання інших кнопок спрощеного віконця виведення повідомлень – за бажанням.

Завдання 1

1. $(2a/b) + a/3{,}1b$;
2. $1{,}1(x - a) + 2^{10}/(x - a)$;
3. $x + a - 4{,}5/(x + a)$;
4. $3{,}5(a - b) - (a - b)/5{,}1$;
5. $2x - 3{,}3 + 8{,}5(2x - 3{,}3)$;
6. $(1{,}1ab - 3)/ab$;
7. $2{,}3(a - b) + 2{,}3(a - b)/a$;
8. $5{,}6(ab + d)/[5{,}6(ab - d)]$;
9. $(de - e)/[2{,}2(de - e)]$;
10. $(a/2{,}3b) - (2{,}3a/b)$.

Завдання 2. Самостійно придумати завдання та написати програму з обов'язковим використанням макросів masm32 за варіантом згідно з останньою цифрою номера студента в групі. Використання інших макросів – за бажанням.

1. Макроси запитів.
2. Макроси файлового введення-виведення.
3. Макроси розміщення пам'яті.
4. Консольні макроси.
5. Макроси рядкових команд.
6. Макроси динамічних рядкових масивів.
7. Макроси шляхів та директорій.

8. Макроси розподілення пам’яті.
9. Макроси бібліотек.
10. Макроси перетворення.

Вимоги до програм

Програма повинна містити введення та виведення дійсних чисел на екран монітора з використанням API-функцій.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем.
3. Лістинг програми з детальним коментарем до кожної команди.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.10.1. Написати програму з використанням макросів для обчислення виразу

$$a(a - b) - (a - b)$$

та слів розміром у байт.

Програму без виведення результату на монітор наведено в лістингу 3.10.1.

Лістинг 3.10.1:

```
title CopyRight by Rysovaniy A. N.     RYSOV@RAMBLER.RU
.686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам’яті та угоди ОС Windows
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\macros\macros.asm
uselib kernel32, fpu
  mSubB macro _a,_b          ;; макрос з ім’ям mSubB для (a – b)
    mov al,_a                 ;; занесення змінної a
    sub al,_b                 ;; віднімання a – b
    mov res1,al               ;; збереження результату у пам’яті
  endm                        ;; закінчення макросу
@ macro c0,c1,c2,c3 ;; макрос для запису команд у рядок
c0
 c1
  c2
   c3
```

```
   endm  ;; закінчення макросу @
.data                                   ; директива визначення даних
       _a  db 6    ; збереження в комірці пам'яті розміром в байт операнда 6
       _b  db 5    ; збереження в комірці пам'яті розміром в байт операнда 5
       res1 db 0           ; резервування пам'яті для результату res1
       buf dd ?,0              ; буфер виведення повідомлення
ifmt  db "    Уравнение: ", 0dh,0ah,"Y = a(a – b) - (a – b)", 0dh,0ah,0ah, ;
   "При a = 6, b = 5, Y = %d",0dh,0ah,0ah, "Автор программы:",0  ;
       titl db "Результат решения уравнения",0 ; назва спрощеного вікна
.code                                   ; директива початку програми
   _start:                              ; мітка початку програми з ім'ям _start
       ; виконання [a x mSubB]
 @<mSubB [_a],[_b]>,<mov al,_a>,<mul res1>,<mov bx,ax>
       ; виконання  [- mSubB]
 @<mSubB [_a],[_b]>,<movzx ax,res1>,<sub bx,ax>,<movzx ebx,bx>
invoke wsprintf, ADDR buf,\ ; перетворення числа та адр. буф. з символами
  ADDR ifmt, ebx     ; адреса рядка перетворення та регістр з результатом
invoke MessageBox, 0,addr buf,\ ; hwnd та адр. буфера текст повідомлення
 addr titl, MB_ICONINFORMATION ; адреса заголовка вікна та вигляд вікна
invoke ExitProcess, 0 ; повернення управління ОС та визволення ресурсів
       end  _start          ; директива закінчення програми з іменем start
```

У зв'язку з тим, що після перемноження регістрів al та bl результат зберігається у регістрі ax розміром у слово, виникає необхідність у подальшому зберіганні результату розміром 16 розрядів. Через цю причину використовується команда розширення розрядності movzx.

У наведеній програмі на екран виводиться вміст регістра ebx. Причому API-функції потребують, щоб число, яке виводиться, мало розрядність 32. Інакше буде генерована помилка при трансляції. Щоб помилок не виникало, використано команду збільшення розрядності регістра до 32 (movzx ebx,bx).

Для виведення чисел на екран за допомогою API-функцій необхідно, щоб результат був розташований у 32-розрядному регістрі. Тому в програмі використано команду movzx, яка розширює результат до 32-розрядного формату. Результат виконання програми наведено на рис. 3.10.1.

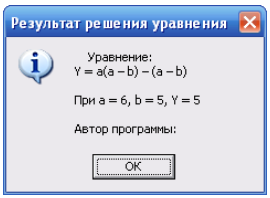


Рис. 3.10.1. Результат виконання програми з лістингу 3.10.1

При налагодженні програми не слід копіювати команди з редактора Word, тому що він вставляє у текст службові символи, деякі з яких є невидимими потім навіть у блокноті. Слід зразу використовувати блокнот чи інші текстові редактори, призначені для таких цілей.

Приклад 3.10.2. Написати програму з використанням макросів для обчислення виразу

$$1{,}1(x - a) + 210/(x - a).$$

Лістинг 3.10.2:

```
Title Вернидуб А. В., НТУ "ХПІ", КІТ-29б
.686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\macros\macros.asm
uselib kernel32,user32, fpu
    mSub macro x, a         ; макрос з ім`ям mSub
    fld x                       ; завантаження x
    fld a                       ; завантаження a
    fsub                        ; x-a                       ;; 2*x - 3.3
    endm                    ; закінчення макросу
.data                               ; директива визначення даних
    st1 db "1,1(x – a) + 2^10/(x – a), де x=2.7, a=3.2  ",0
    buf db "  Виконала студентка групи КІТ-29б Вернидуб Анна", 10,13
    st2 db "     Результат:    "
    sum dt 0
    x dd 2.7
    a dd 3.2
    res dt ?
    const1 dd 1.1
    const2 dd 2
.code                               ; директива початку коду програми
 Begin:                             ; мітка початку програми
    mov ecx,8
    finit                           ; ініціювання FPU
    fild const2
    fild const2
 _1: fmul st(0),st(1)
   loop _1
    fmulp st(1),st
    fld const1
    fadd
    mSub [x], [a]       ; виклик макросу
    fmul
```

```
fstp res                    ; збереження результату
invoke FpuFLtoA,ADDR res, 2, ADDR sum, SRC1_REAL or SRC2_DIMM
invoke MessageBox, NULL, addr buf, addr st1, MB_ICONINFORMATION
invoke ExitProcess, 0
end Begin
```

Результат виконання програми з лістингу 3.10.2 наведено на рис. 3.10.2.

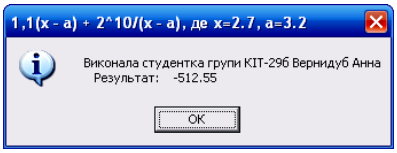


Рис. 3.10.2. Результат виконання програми з лістингу 3.10.2

3.11. Лабораторна робота 11
ДИРЕКТИВИ УМОВНОГО АСЕМБЛЮВАННЯ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням директив умовного асемблювання, написаних мовою асемблера для МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати програму з використанням **директив умовного асемблювання та дійсних чисел** з обов'язковим виведенням результатів у спрощене вікно (вікна) та додати прізвище та ініціали про автора програми, свій e-mail, варіант завдання та умову цього завдання.

Використати функцію MessageBox разом із графічними позначками цієї функції та вказати прізвище автора програми.

1. Задано масив A з $N = 5$ елементів. Навести алгоритм та програму визначення суми елементів масиву A, для яких біти 2 та 10 збігаються. Вивести відповідні повідомлення.

2. Проаналізувати масив даних mas1 з 6 елементів. Підрахувати і зберегти в комірках пам'яті кількість елементів масиву, якщо їх значення менші або більші від першого елемента та дорівнюють останньому. Вивести відповідні повідомлення.

3. Задано масив A з $N = 7$ елементів. Навести алгоритм та програму формування масиву B з елементів масиву A, у яких 0, 2 та 5 біти мають нулі.

4. Проаналізувати масив даних з 5 елементів. Додавати елементи масиву доти, поки значення суми не перевищить 512 та не буде менше 510. Зберегти номер елемента, на якому відбулося переповнення. Якщо сума елементів не досягла значення 512, то видати відповідне повідомлення.

5. Проаналізувати масив даних з 6 елементів. Елементами масиву є числа 32 та 128. Підрахувати та вивести на екран кількість елементів, які за значенням знаходяться у середині цього діапазону.

6. Проаналізувати масив даних A з 5 елементів. Створити масив B, до якого входять елементи першого масиву, що задовольняють умову $-1 \le B_i > 128$. Перервати виконання програми, якщо буде знайдено 5 елементів зі значенням 12. Вивести відповідні повідомлення.

7. Проаналізувати 2 масиви, що складаються з 6 елементів кожен. Підрахувати кількість елементів першого масиву, що мають рівні значення в другому масиві. Вивести відповідні повідомлення.

8. Проаналізувати масив даних з 4 елементів. Підрахувати кількість елементів, значення яких дорівнює 55h та рахунок перервати, якщо кількість таких елементів перевищить 3. Вивести відповідні повідомлення.

9. Задано масиви *A* і *B* по *N* = 6 елементів. Навести алгоритм та програму формування масиву *C* за таким правилом: якщо у елементів *Ai* та *Bi* біти 2, 3 та 8 збігаються, то *Ci* = *Ai* + *Bi*. Вивести відповідні повідомлення.

10. Проаналізувати масив даних з 7 елементів. Підрахувати кількість елементів, які знаходяться у діапазонах 10–20 та 30–40. Вивести відповідні повідомлення.

12. Задано масив *A* з *N* = 7 елементів. Структура масиву *A: X*1, *Y*1; *X*2, *Y*2; … . Навести алгоритм та програму визначення кількості пар, для яких виконується умова *Xi* ≤ *Yi*. Вивести відповідні повідомлення.

13. Задано масив *A* з 5 елементів. Навести алгоритм та програму визначення елементів масиву *A*, які задовольняють умову $6 \le A_i < 3A$, та розмістити їх у новому масиві. Вивести відповідні повідомлення.

14. Задано масиви *A* і *B* по *N* = 6 елементів. Навести алгоритм та програму формування масиву *C* за таким правилом: якщо *Ai* – *Bi* ≤ 0, то *Cj* = *Ai*.

15. Задано масив *A* з *N* = 5 елементів. Навести алгоритм та програму визначення суми та кількості елементів масиву *A*, які задовольняють умову *Ai* ≥ *E* при *E* = – 11. Вивести відповідні повідомлення.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.11.1

Проаналізувати масив даних з 10 елементів розміром у подвійне слово. Додавати елементи масиву доти, поки значення суми не перевищить 15. Зберегти номер елемента, на якому відбулося переповнення. Видати повідомлення на екран монітора (текстове або/чи числове).

Блок-схему алгоритму виконання прикладу наведено на рис. 3.11.1.

Розв'язання прикладу 3.11.1 з використанням директив високого рівня, але поки без виведення даних, наведено у лістингу 3.11.1.

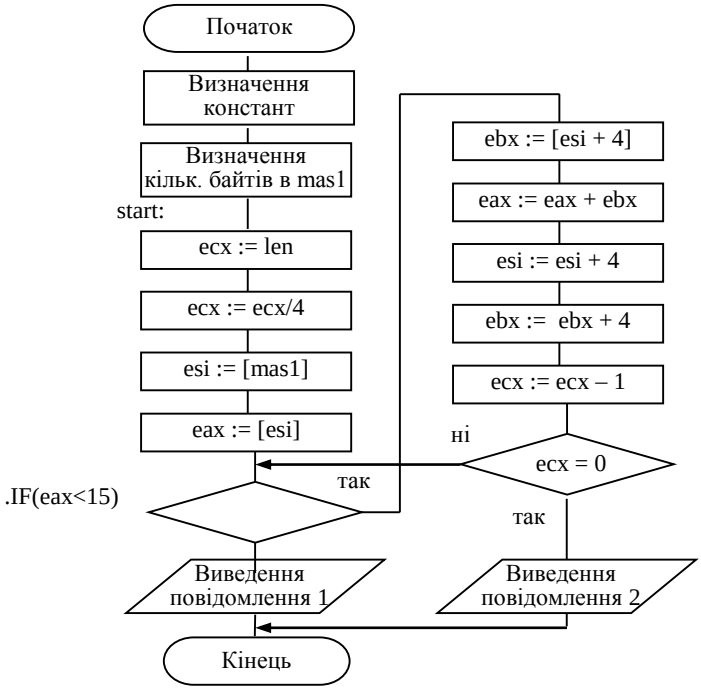


Рис. 3.11.1. Блок-схема алгоритму виконання прикладу 3.11.1

Лістинг 3.11.1:

```
        title CopyRight by Rysovaniy A. N.                rysov@rambler.ru
        .386                ; директива визначення типу мікропроцесора
.model flat, stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
        option casemap:none         ; відмінність малих та великих літер
.data                                       ; директива визначення даних
        mas1 dd 1,4,5,4,6,7,3,8,2,10      ; масив чисел
        len equ $-mas1                  ; визначення кількості байтів у масиві
.code                           ; директива початку сегмента даних
_start:                          ; мітка початку програми з ім'ям _start
        mov ecx,len             ; завантаження лічильника
        shr ecx,2               ; визначення кількості подвійних слів
```

```
    lea esi, mas1        ; завантаження адреси початку масиву
    mov eax, [esi]       ; завантаження числа
m1:
  .IF(eax<15)            ;  умова
    mov ebx,[esi+4]              ; завантаження нового числа з масиву
    add eax,ebx                  ; додавання елементів
    add esi,4                    ; розрахунок адреси нового числа
    add ebx,4                    ; розрахунок адреси наступного числа
    loop m1                      ; перейти на m1, якщо ecx = 0
  .ENDIF                         ; закінчення директиви високого рівня
    ret                                  ; повернення управління ОС
    end _start         ; директива закінчення програми з ім'ям _start
```

У наведеній програмі при виникненні необхідності додавання двох чисел одного масиву слід використати додатковий регістр (для зберігання адреси другого числа). Таким регістром вибрано регістр ebx. Завантаження в цей регістр значення адреси другого числа виконується командою mov ebx,[esi+4].

У зв'язку з тим, що за завданням вибрано формат регістрів у 32 розряди (4 байти), для підготовки зчитування нових слів виконуються команди

```
add esi,4
add ebx,4.
```

Вікна налагоджувача після налагодження програми з лістингу 3.11.1 наведено на рис. 3.11.2.

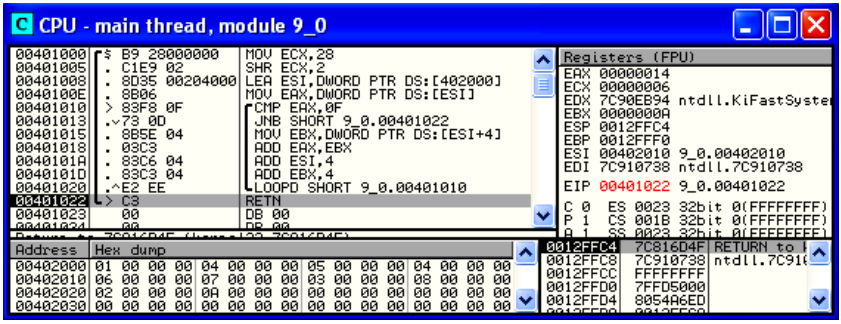


Рис. 3.11.2. Вікна налагоджувача після налагодження програми

Для виконання другої частини роботи необхідно організувати виведення даних на екран. Для цього доцільно використати API-функцію WriteConsoleA, фрагмент застосування якої може бути таким:

```
    …
    BSIZE     equ 15
   .data                                   ; директива визначення даних
     stdout dd ? ; дані для дескриптора виведення, що не визначені
     buf    db BSIZE dup(?)          ; резервування пам'яті для буфера
     frmt    db  "%d",0          ; задання перетворення одного символу
 cWritten    dd  ?              ; комірки пам'яті для адрес  символів виведення
   . . .
   .code                                   ; директива початку сегмента даних
   _start:                                 ; мітка початку програми з ім'ям _start
   . . .
invoke GetStdHandle, STD_OUTPUT_HANDLE ; отриманого дескриптора
     mov stdout, eax              ; запам'ятовування отриманого дескриптора
   . . .
invoke  wsprintf, \                        ; API-функція перетворення числа
 ADDR buf, \ ; адреса буфера, куди буде записана послідовність символів
  ADDR frmt, \                  ; адреса рядка перетворення формату
   edx                          ; регістр, вміст якого перетворюється
invoke  WriteConsoleA, \       ; API-функція виведення на екран
  stdout, \                  ; дескриптор стандартного пристрою виведення
    ADDR buf,  BSIZE,\ ; адреса повідомлення та розмір повідомлення
       ADDR cWritten, 0  ; адреса, де зберігається кількість символів
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
     end _start                ; директива закінчення програми з іменем _start
```

У цій програмі використовуються файли user32.inc і user32.lib, що зберігають інформацію про функцію **wsprintf**. Кількість параметрів цієї функції поки дорівнює трьом:

1. Параметр ADDR buf – це адреса буфера, куди буде записана послідовність символів. Пам'ять для буфера виділяється рядком buf db BSIZE dup(?), який резервує кількість байтів, заданих змінною BSIZE. Про резервування байтів пам'яті говорить слово dup – скорочене від англійського слова *duplication* (повторення). Знак питання в дужках після dup свідчить про те, що значення байтів наперед не визначене.

Розмір буфера, позначений ім'ям BSIZE, є реальним числом, яким асемблер замінить змінну BSIZE та яке задається рядком BSIZE equ 15.

2. Параметр ADDR ifmt – це адреса рядка формату, яка задає тип перетворення. Цей рядок складається з символів і завжди завершується

нульовим байтом. Рядок “%d”,0 задає перетворення одного цілого числа в послідовність символів, а рядок “%d%d”,0 – перетворення двох чисел.

3. Параметр edx указує функції wsprintf на те, що число, яке потребує перетворення в послідовність символів, знаходиться, наприклад, у регістрі edx.

Процедура WriteConsoleA виводить ці символи на екран. За першим параметром цієї функції (stdout) перетворене у символ число виводиться через консоль, за другим (параметр ADDR buf) указується адреса початку повідомлення, за третім (BSIZE) – розмір повідомлення, за четвертим (ADDR cWritten) – адреса ділянки пам’яті, де процедура WriteConsoleA зберігає кількість виведених на екран символів.

Приклад 3.11.2. Проаналізувати масив даних mas1 з 6 елементів. Підрахувати і зберегти в комірках пам’яті кількість елементів масиву, якщо їх значення менші або більші від першого елемента та дорівнюють останньому.

Лістинг 11.2:

```
    .686                        ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам`яті та угоди ОС Windows
    option casemap:none         ; відмінність малих та великих літер
    include C:\masm32\include\mcdsrv32.inc
    include C:\masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\kernel32.inc ; файли систем. функцій застосувань
    include C:\masm32\include\\user32.inc   ; файли інтерфейсу …
    includelib C:\masm32\lib\mcdsrv32.lib
    includelib C:\masm32\lib\kernel32.lib
    includelib C:\masm32\lib\user32.lib
    IDI_ICON EQU 1001
.data
    _min dd 0
    _max dd 0
    _toj dd 0
    _st1 db "ЛР7 Директиви умовного асемблювання",0
    _st2 dd ?,0
    _mas1 dd 4,3,44,12,22,4
    _El1 dd 4
    ifmt db "Исходный массив:",0dh,0ah,\
    "04  03  44  12  22  04",0dh,0ah,0ah,\
    "Количество елементов, равных первому = %d",0dh,0ah,0ah,\
    "Количество елементов меньших, чем первый = %d",0dh,0ah,0ah,\
    "Количество елементов больших, чем первый = %d",0dh,0ah,0ah,\
    "Автор:  Гладких А.А., КІТ-39",0
    params MSGBOXPARAMS <>
```

```
.code
_nach:
      mov ecx,5
      lea esi,_mas1
      add esi,4
m1:     mov ebx,_El1
      mov eax,dword ptr [esi]
      .IF eax==ebx
      add _toj,1
      .ELSEIF eax > ebx
      add _max,1
      .ELSEIF eax < ebx
      add _min,1
      .ENDIF
      add esi,4
     loop m1
      invoke wsprintf,ADDR _st2,ADDR ifmt,_toj,_min,_max
        mov params.cbSize,SIZEOF MSGBOXPARAMS
        mov params.hwndOwner,0
        invoke GetModuleHandle,0
        mov params.hInstance,eax
        mov params.lpszText,offset _st2
        mov params.lpszCaption,offset _st1
        mov params.dwStyle,MB_USERICON
        mov params.lpszIcon,IDI_ICON
        mov params.dwContextHelpId,0
        mov params.lpfnMsgBoxCallback,0
        mov params.dwLanguageId,LANG_NEUTRAL
        invoke MessageBoxIndirect,ADDR params
      invoke ExitProcess,0                    ; закінчення програми
      end _nach
```

Результат виконання програми наведено на рис. 3.11.3.

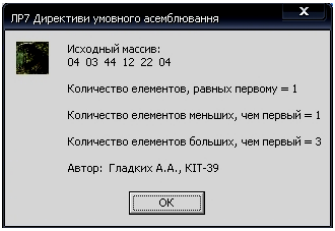


Рис. 3.11.3. Результат виконання програми з лістингу 3.11.2

Приклад 3.11.3. Задано масиви *A* і *B* по *N* = 6 елементів. Навести алгоритм та програму формування масиву *C* за таким правилом: якщо у елементів *Ai* та *Bi* біти 4 та 9 збігаються, то *Ci* = *Ai* + *Bi*. Вивести відповідні повідомлення.

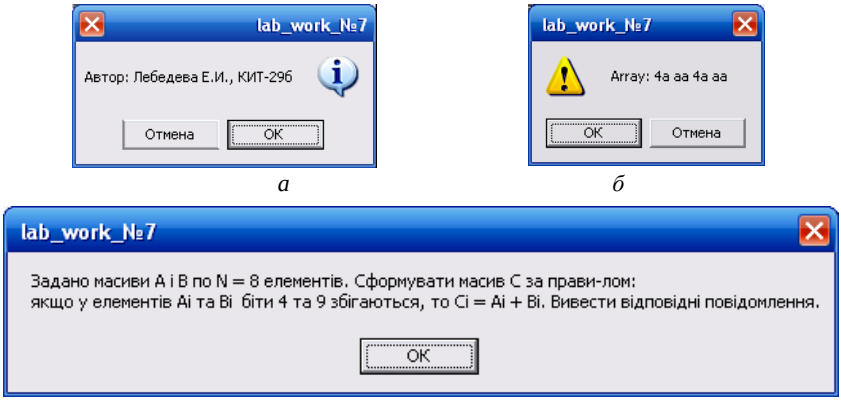
Лістинг 3.11.3:

```
      .686                          ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам`яті та угоди ОС Windows
      option casemap:none           ; відмінність малих та великих літер
include C:\masm32\include\windows.inc ; файли структур, констант …
include\masm32\include\kernel32.inc ; файли систем. функцій застосувань
include C:\masm32\include\\user32.inc  ; файли інтерфейсу …
includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib
.data                               ; директива визначення типу мікропроцесора
  mas1 dd 2 dup(25h, 147h, 55h, 0A4h) ; резервування пам`яті для mas1
  mas2 dd 2 dup (25h, 34h, 55h, 155h) ; резервування пам`яті для mas2
  temp dd 0                         ; резервування пам`яті для змінної temp
  mas3 dd 8 dup(0)
  stdout DWORD ?                    ; резервування 32р. комірки
  buf db BSIZE dup(?)               ; резервування пам`яті для буфера
  frmt db "%x", 0                   ; задання перетворення числа
cWritten DWORD ? ; резервув. 32-розр. комірки для адреси виведення
  mt db " lab_work_№7", 0     ; назва віконця
  ifmt db "Автор: Лебедева Е.И., КИТ-29б", 0
ifmt1 db "Задано масиви A і B по N = 8 елементів. Сформувати масив C за
правилом:", 0ah, 0dh,
"якщо у елементів Ai та Bi  біти 4 та 9 збігаються, то Ci = Ai + Bi. Вивести
відповідні повідомлення.", 0ah, 0dh, 0
ifmt2 db "Array: %x %x %x %x " , 0  ; формат виведення  mas3

.code
_start:                     ; мітка початку програми
     lea esi, mas1              ; записати адресу початку масиву mas1
     lea edi, mas2              ; записати адресу початку масиву mas2
     mov ecx, 8                 ; лічильник кількості пар чисел
     xor edx, edx               ; edx = 0
m4: mov ebx, [esi]     ; помістити число з mas1
      mov eax, [edi]       ; помістити число з mas2
      mov temp, eax      ;  збереження для подальшого складання
```

```
      and ebx, 108h   ; логічне множення числа з першого масиву на маску
      and temp, 108h   ; логічне множення числа з другого масиву на маску
      sub ebx, temp        ; EBX:=ebx-temp
     .IF ebx==0            ; перевірка різниці на рівність 0
        mov ebx, [esi]
        add ebx, eax       ; формування елемента mas3
        mov mas3[edx], ebx  ; запис в mas3
        add esi, 4         ; просування по першому масиву
        add edi, 4         ; просування по другому масиву
        dec ecx
      .IF ecx!=0           ; перевірка регістра ecx на нерівність 0
       jmp m4              ; якщо регістр ecx не дорівнює 0, то перейти на m4
      .ENDIF               ; закінчення директиви ecx!=0
    .ELSE                 ; альтернативний блок коду
       add esi, 4         ; просування по першому масиву
       add edi, 4         ; просування по другому масиву
       add edx, 4
       dec ecx
      .IF ecx!=0          ; перевірка регістра ecx на нерівність 0
       jmp m4             ; якщо регістр ecx не дорівнює 0, то перейти на m4
      .ENDIF              ; закінчення директиви ecx!=0
   .ENDIF                 ; закінчення директиви ebx==0
     invoke Sleep, 1000d
invoke MessageBox, NULL, addr ifmt, addr mt, \
  MB_OKCANCEL+MB_ICONINFORMATION +180000h
     xor eax, 1
     jz a1                              ; якщо натиснуто ОК
       jmp  a2                          ;  інакше – “Отмена”
a1:
     invoke Beep, 01234, 300            ; звуковий сигнал
invoke wsprintf, ADDR buf, ADDR ifmt2, mas3, mas3[4], mas3[8], mas3[12]
invoke MessageBox, NULL, addr buf, addr mt, \
   MB_OKCANCEL +MB_ICONEXCLAMATION
a2:
       invoke Beep, 02578, 200
       invoke MessageBox, NULL, addr ifmt1, addr mt, MB_OK
a3:
     invoke ExitProcess, 0
     end _start ; закінчення програми з ім’ям _start
```

Результат виконання програми наведено на рис. 3.11.4.



а *б*

с

Рис. 3.11.4. Результат виконання програми з лістингу 3.11.3: *а* – перше вікно програми; *б* – вікно програми при натисканні на першому вікні кнопки OK; *с* – останнє вікно

3.12. Лабораторна робота 12
ДВОВИМІРНІ МАСИВИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм для програмування двовимірних масивів, написаних мовою асемблера для МП платформи x86 з організацією виведення результату на екран з використанням API-функцій під Win32.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати програму з ***введенням*** та ***виведенням*** даних на екран монітора:

1. Ввести двовимірний масив A(*N*,*N*). Скласти алгоритм і програму підрахунку середнього арифметичного значення двовимірного масиву. Знайти відхилення від середнього значення в елементів першого рядка.

2. Ввести двовимірний масив A(*N*,*N*). Скласти алгоритм і програму підрахунку середнього арифметичного значення двовимірного масиву. Обчислити відхилення від середнього значення для всіх елементів двовимірного масиву.

3. Ввести двовимірний масив A(*N*,*N*). Скласти алгоритм і програму заміни всіх негативних елементів на середнє арифметичне значення елементів двовимірного масиву.

4. Скласти алгоритм і програму знаходження кількості рядків двовимірного масиву A(*N*,*N*), кількість негативних елементів у яких більше M.

5. Ввести двовимірний масив розміром 7×4. Знайти найбільший елемент двовимірного масиву. Видалити рядок з максимальним елементом.

6. Ввести двовимірний масив розміром 3×4. Поміняти стовпець з максимальним елементом на перший стовпець двовимірного масиву.

7. Ввести двовимірний масив розміром 4×7. Знайти максимальний елемент двовимірного масиву, розташований нижче за побічну діагональ.

8. Ввести двовимірний масив розміром 5×4. Знайти найменший елемент двовимірного масиву. Перенести рядок, що містить цей елемент, в кінець.

9. Ввести двовимірний масив розміром 6×4. Знайти максимальний елемент двовимірного масиву. Перенести рядок, що містить цей елемент, в кінець.

10. Ввести двовимірний масив розміром 7×4. Знайти мінімальний елемент двовимірного масиву. Переставляючи рядки і стовпці, добитися того, щоб цей елемент опинився в правому нижньому кутку.

11. Створити матрицю B[1..*N*,1..*M*] з дійсних чисел. З кожного рядка матриці визначити числа, яких немає в наступному рядку, та записати їх в одновимірний масив.

12. Скласти алгоритм обчислення суми елементів двовимірного масиву A(1.. *N*, 1..*M*), розташованих вище за головну діагональ.

13. Скласти алгоритм обчислення кількості непарних елементів у кожному рядку двовимірного масиву A(1.. *N*, 1..*M*).

14. Скласти алгоритм пошуку максимального значення в двовимірному масиві A(*N*,*M*).

Зміст звіту

1. Постановка задачі для конкретного варіанта.

2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.

3. Лістинг програми та коментарі до всіх команд.

4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.

5. Короткий опис виконання програми.

6. Висновки за результатами роботи.

Приклад 3.12.1. Знайти максимальне значення в рядках двовимірного масиві A(*N*,*M*). Вивести ці числа функцію MessageBox.

Лістинг 3.12.1:

```
…
.data           ;
    mas1 dd 10,33,3,64,6,5,82,
         7,80,6,78,8,15,4
    len equ ($-mas1)/4/2    ; визначення чисел в одному рядку
    max1 dd 0       ; змінна для максимального числа першого рядка
    max2 dd 0       ; змінна для максимального числа другого рядка
    titl db "Результат программы",0
buf dd 0,0
```

```
 ifmt db "Условие: найти максимальное значение ",0dh,0ah,\
  "в строках двумерных массивов:",0dh,0ah,\
   "10   33   3   64   6   5   82",0dh,0ah,\
   "  7   80   6   78   8  15    4",0dh,0ah,0ah,\
   "Максимальное значение в первой строке:    %d",0dh,0ah,\
   "Максимальное значение во второй строке:  %d",0

.code                              ;
  _start:
        mov ecx,len       ; кількість чисел в одному рядку
        lea esi, mas1       ; завантаження адреси початку масиву
        mov eax, [esi]          ; завантаження числа
m1:

   .IF(eax >= max1)   ; порівняння числа з першого рядка з максимальним
        mov max1, eax
        add esi,4              ; розрахунок адреси нового числа
        mov eax,[esi]
        loop m1                  ; пересилка нового числа

   .ELSE       ; иначе
        add esi,4                ; розрахунок адреси нового числа
        mov eax,[esi]
       loop m1

   .ENDIF
         mov ecx,len              ; кількість чисел в одному рядку
  m2:

    .IF(eax >= max2)  ; порівняння числа з другого рядка з максимальним
       mov max2, eax
        add esi,4                       ; розрахунок адреси нового числа
        mov eax,[esi]
        loop m2

    .ELSE   ; иначе
        add esi,4                       ; розрахунок адреси нового числа
        mov eax,[esi]
       loop m2

    .ENDIF
       invoke wsprintf, ADDR buf, ADDR ifmt, max1, max2
       invoke MessageBox, 0, addr buf, addr titl, MB_ICONEXCLAMATION
        invoke ExitProcess,0
        end _start
```

Результат виконання програми наведено на рис. 3.12.1.

У зв’язку з тим, що числа в масиві всі позитивні, то для знаходження максимального на початку обирається нульове число та з ним порівнюється наступне. На наступному етапі вже це число є максимальним й з ним порівнюється інше, й так далі.

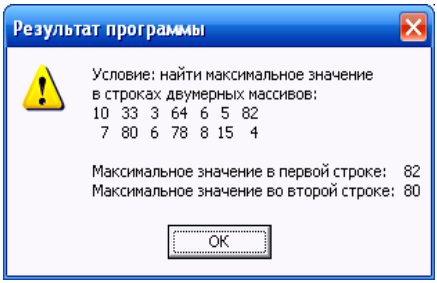


Рис. 3.12.1. Результат виконання програми

3.13. Лабораторна робота 13
СТРУКТУРИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням структур та API-функцій під Win32, написаних мовою асемблера МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант завдання та написати на асемблері програму обчислення одного з виразів з виведенням свого завдання, отриманих даних та довідку про автора програми функцією MessageBoxIndirect з відображенням **своєї фотографії** як іконки.

Завдання 1

1. Задано матрицю 3×5. Виконати транспонування цієї матриці. Результат виконання програми вивести у вікно консолі.

2. Задано матрицю 6×6. Визначити суму елементів під головною діагоналлю. Результат виконання програми вивести у вікно консолі.

3. Задано матрицю 3×4. Визначити рядок з максимальною сумою позитивних елементів. Результат виконання програми вивести у вікно консолі.

4. Задано матрицю 4×5. Визначити мінімальний елемент кожного стовпця. Результат виконання програми вивести у вікно консолі.

5. Задано матрицю 5×7. Визначити максимальний елемент кожного парного рядка. Результат виконання програми вивести у вікно консолі.

6. Задано матрицю 4×6. Визначити суму елементів кожного стовпця. Результат виконання програми вивести у вікно консолі.

7. Задано матрицю 3×6. Визначити рядок з максимальною сумою елементів. Результат виконання програми вивести у вікно консолі.

8. Задано матрицю 3×5. Визначити рядок з мінімальною сумою елементів. Результат виконання програми вивести у вікно консолі.

9. Задано матрицю 3×6. Визначити елементи, кратні 3, в кожному рядку і помістити на їх місце елемент, номер якого збігається з номером рядка. Результат виконання програми вивести у вікно консолі.

10. Задано матрицю 4×6. Визначити суму негативних елементів кожного рядка і помістити її на місце першого елемента. Результат виконання програми вивести у вікно консолі.

Завдання 2

1. Задано послідовність структур. Структура містить поля: назва автомобіля, порядковий номер, ім'я власника, кількість порушень. Обчислити кількість власників, у яких більше трьох порушень.

2. Задано послідовність структур. Структура містить поля: ім'я студента, стипендія, середня оцінка, вік. Обчислити середній вік студентів.

3. Задано послідовність структур. Структура містить поля кредитної картки: номер, прізвище власника, розмір грошової суми і ступінь захисту. Обчислити кількість кредитних карток з грошовою сумою більше $300 і ступенем захисту більше 10.

4. Задано послідовність структур. Структура містить поля (відповідно до комп'ютера): серійний номер, ціна, назва, прізвище власника, розмір монітора в дюймах. Обчислити середню ціну 26-дюймових моніторів.

5. Задано послідовність структур. Структура містить поля за характеристиками студентських груп: назва групи, успішність (середня цілочислова оцінка), номер курсу. Вивести на екран назву групи з максимальною успішністю.

6. Задано послідовність структур. Структура містить поля з торгової діяльності комп'ютерної фірми: назва товару (комп'ютер, модем, гвинт), ціна (у доларах), кількість проданих одиниць. Обчислити прибуток фірми (підсумовування "кількості проданих одиниць помножену на ціну").

7. Задано послідовність структур. Структура містить поля з торгової діяльності комп'ютерної фірми: назва товару (комп'ютер, модем, гвинт), ціна, кількість проданих одиниць. Обчислити прибуток по кожному товару (прибуток = кількість проданих одиниць, помножена на ціну).

8. Задано послідовність структур. Структура містить поля з зарплати викладачів: прізвище викладача, назва курсу (який веде викладач), посада, зарплата. Обчислити прізвище самого низькооплачуваного викладача.

9. Задано послідовність структур. Структура містить поля про дані податкової інспекції: назва фірми, прибуток, розмір оплачуваних податків, кількість штрафів. Обчислити кількість фірм, дохід яких перевищує середній дохід фірм.

10. Задано послідовність структур. Структура містить поля: назва автомобіля, порядковий номер, ім'я власника, кількість порушень. Вивести на екран ім'я власника з максимальною кількостю порушень.

11. Задано послідовність структур. Структура містить поля: назва автомобіля, порядковий номер, ім'я власника, кількість порушень. Вивести на екран ім'я власника з мінімальним числом порушень.

12. Задано послідовність структур. Структура містить поля: ім'я студента, стипендія, середня оцінка, вік. Обчислити середню оцінку студентів.

13. Задано послідовність структур. Структура містить поля: назва автомобіля, порядковий номер, ціна автомобіля, рік випуску, розмір податків. Обчислити кількість автомобілів з останнім (максимальним) роком випуску.

14. Задано послідовність структур. Структура містить поля даних про комп'ютер: серійний номер, ціна, назва, прізвище власника, розмір монітора в дюймах. Обчислити середню ціну комп'ютера.

15. Задано послідовність структур. Структура містить поля за характеристиці студентських груп: назва групи, успішність (середня цілочислова оцінка), номер курсу. Вивести на екран назву групи з мінімальною успішністю.

16. Задано послідовність структур. Структура містить поля за торговою діяльності комп'ютерної фірми: назва товару (комп'ютер, модем, гвинт), ціна (у доларах), кількість проданих одиниць. Обчислити загальну кількість проданих одиниць.

17. Задано послідовність структур. Структура містить поля з зарплати викладачів: прізвище викладача, назва курсу (який веде викладач), посада, зарплата, адреса. Вивести на екран адресу максимально оплачуваного викладача.

18. Задано послідовність структур. Структура містить поля з зарплати викладачів: прізвище викладача, назва курсу (який веде викладач), посада, зарплата, адреса. Обчислити середню, мінімальну і максимальну зарплату викладачів.

19. Задано послідовність структур. Структура містить поля за даними податкової інспекції: назва фірми, прибуток, розмір оплачуваних податків, кількість штрафів. Вивести на екран назву фірми з максимальною кількістю штрафів.

20. Задано послідовність структур. Структура містить поля за даними податкової інспекції: назва фірми, прибуток, розмір оплачуваних податків, кількість штрафів. Вивести на екран назву фірми з максимальним прибутком.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.13.1. Задано матрицю 2 × 4. Визначити суму всіх негативних елементів матриці. Результат виконання програми вивести у вікно консолі.

Лістинг 3.13.1. Програма виконання прикладу 3.13.1:

```
      title CopyRight by Rysovaniy A. N., rysov@rambler.ru
      .386                    ; директива визначення типу мікропроцесора
.model  flat, stdcall  ; задання лінійної моделі пам'яті  та угоди ОС
      option casemap:none              ; відмінність малих та великих літер
    include \masm32\include\windows.inc   ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib kernel32, user32;
      DATE1 STRUCT      ; тип даних СТРУКТУРА з іменем DATE1
      elem1 dd ?            ; ім'я першого поля структури
      elem2 dd ?            ; ім'я другого поля структури
      elem3 dd ?            ; ім'я третього поля структури
      elem4 dd ?            ; ім'я четвертого поля структури
      DATE1 ENDS
.data                                 ; директива визначення даних
        str1 DATE1 <1,-1,-2,3>              ; структура з іменем str1
        str2 DATE1 <0,-2,-1,-3>             ; структура з іменем str2
      titl1 db " Работа с элементами структуры",0
      buf db 10 dup(?),0
ifmt db "Задана матрица:",0dh,0ah,\
 "1   -1   -2   3",0dh,0ah,\
 "0   -2   -1  -3",0dh,0ah,0ah,\
   "Результат сложения отрицательных элементов",0dh,0ah,\
   "   = %d                  ",0dh,0ah,0ah,\
    "Автор программы:             НТУ ХПИ",0
.code                                  ; директива початку сегмента даних
 start:                                ; мітка початку програми з ім'ям start
      xor edx,edx                           ; заповнювання нулями
      mov ebx,2                             ; завантаження кількості рядків
      lea esi,str1        ; завантаження адреси першого рядка структури
```

```
   m5: mov ecx,4              ; кількість елементів у рядку
   m3: mov eax,[esi]          ; завантаження елемента з рядка структури
       add eax,0              ; визначення ознак елемента
       js m1                  ; перейти на m1, якщо елемент негативний
       jmp m2                 ; безумовний перехід, якщо навпаки
   m1: add edx,eax            ; додавання негативних елементів рядка структури
   m2: add esi,4              ; підготовка адреси нового елемента
       loop m3                ; ecx := ecx – 1 та перехід на m3, якщо не нуль
       dec ebx                ; ebx := ebx – 1
       jz m4                  ; якщо ebx = 0 (z = 1), то перехід на закінчення
       lea esi,str2                        ; завантаження адреси нового рядка
       jmp m5                 ; перехід на новий цикл
   m4:
invoke  wsprintf, \                        ; API-функція перетворення числа
 ADDR buf, \; адреса буфера, куди буде записана послідовність символів
  ADDR ifmt, edx ; адр. рядка перетворення та регістр перетворення
invoke MessageBox, 0,  addr buf, \ ; адреса тексту повідомлення
 addr titl1, MB_ICONINFORMATION+180000h ; заголовок та вигляд вікна
invoke ExitProcess, 0  ; повернення управління ОС та вивільнення ресурсів
       end start           ; директива закінчення програми з іменем start
```

Результат виконання програми з лістингу 3.13.1 наведено на рис. 3.13.1.

У результаті виконання програми отримується число –9, яке заноситься до регістра edx, а потім виводиться у вікно консолі.

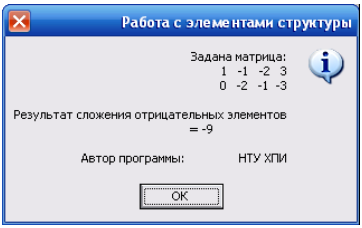


Рис. 3.13.1. Вікно з результатом виконання програми з лістингу 3.13.1

Приклад 3.13.2. Задано матрицю 6 × 6. Визначити суму елементів під головною діагоналлю. Результат виконання програми вивести у спрощене вікно.

Лістинг 3.13.2:

```
Title Вернидуб А. В., НТУ “ХПІ”, КІТ-29б
.686                    ; директива визначення типу мікропроцесора
```

```
      .model flat,stdcall ; задання лінійної моделі пам`яті та угоди ОС
      option casemap:none          ; відмінність малих та великих літер
    include \masm32\include\windows.inc ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib kernel32, user32;
  DATE1 STRUCT          ; тип даних СТРУКТУРА з іменем DATE1
  elem1 dd ?            ; ім'я першого поля структури
  elem2 dd ?            ; ім'я другого поля структури
  elem3 dd ?            ; ім'я третього поля структури
  elem4 dd ?            ; ім'я четвертого поля структури
  elem5 dd ?            ; ім'я п'ятого поля структури
  elem6 dd ?            ; ім'я шостого поля структури
  DATE1 ENDS
.data                   ; директива визначення даних
      str1 DATE1 <1,-1, 2, 3, 7, 8>     ; структура з іменем str1
      str2 DATE1 <1, 2,-1, 3, 5, 1>     ; структура з іменем str2
      str3 DATE1 <0, 2,-1, 3, 5, 7>     ; структура з іменем str3
      str4 DATE1 <0,-2, 1, 8, 0, 5>     ; структура з іменем str4
      str5 DATE1 <0,-2, 1, 2, 5, 3>     ; структура з іменем str5
      str6 DATE1 <7,-2, 1, 3,10, 8>     ; структура з іменем str6
    tmp dd 0
    st1 db "Лабораторна робота №8",0
    buf db 10 dup(?),0
ifmt db "Задано матрицю 6 на 6. Визначити суму елементів під головною
діагоналлю. ",10,13,10,13
    st2 db " Сума елементів під головною діагоналлю    %d",0
.code                   ; директива початку сегмента даних
  Begin:                        ; мітка початку програми з ім'ям Begin
    xor edx,edx
    lea esi,str2        ; визначення адреси початку масиву
    mov ecx,5           ; лічильник
Next:                   ; цикл
    mov eax,[esi]       ; занесення елемента масиву
    adc edx,eax         ; підрахунок суми елементів
   .IF (ecx==5)         ; умова
    lea esi,str3        ; визначення адреси початку масиву
    add esi,4           ; корекція позиції елемента
  .ELSEIF (ecx==4)      ; умова
    lea esi,str4        ; визначення адреси початку масиву
    add esi,8           ; корекція позиції елемента
  .ELSEIF (ecx==3)      ; умова
    lea esi,str5        ; визначення адреси початку масиву
    add esi,12          ; корекція позиції елемента
  .ELSEIF (ecx==2)      ; умова
    lea esi,str6        ; визначення адреси початку масиву
    add esi,16          ; корекція позиції елемента
```

```
    .ENDIF
      xor eax,eax
      loop Next              ; повернення до початку цикла
      mov tmp,edx            ; запис у пам'ять та підготовка до виведення
invoke wsprintf, ADDR buf,addr ifmt, tmp
invoke MessageBox,NULL,addr buf,addr st1,MB_OK
invoke ExitProcess,0
      end Begin
```

Результат виконання програми з лістингу 3.13.2 наведено на рис. 3.13.2.

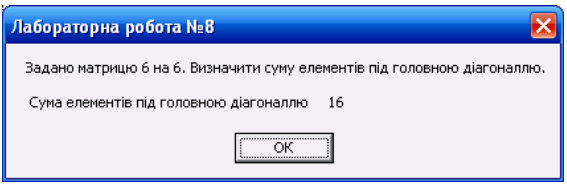


Рис. 3.13.2. Результат виконання програми з лістингу 3.13.2

Приклад 3.13.3. Задано послідовність структур. Структура містить поля: ім'я студента, стипендія, середня оцінка, вік. Обчислити середній вік студентів. Результат виконання програми вивести у спрощені вікна.

Лістинг 3.13.3:

```
      Title Вернидуб А. В., НТУ "ХПІ", КІТ-29б
      .686                        ; директива визначення типу мікропроцесора
      .model flat,stdcall         ; задання лінійної моделі пам`яті та угоди ОС
      option casemap:none   ; відмінність малих та великих літер
    include \masm32\include\windows.inc  ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib kernel32, user32;
  array STRUCT              ; тип даних  – структура
    _suc      db   ?            ; поле структури
    _course   db   ?            ; поле структури
    _name db 10 dup(?)    ; поле структури
    _step     db   ?
      array ENDS        ; закінчення даних СТРУКТУРА
.data           ; директива визначення даних
      max db 0
      sum dd ?
      first array  < 21, 5, "Иванов",70>     ; структура з іменем first
      second array < 35, 5, "Петров",50> ; структура з іменем second
      third array  < 21, 5, "Сидоров",30>     ; структура з іменем third
```

```
        fourth array < 35, 5, "Киселёв",60>     ; структура з іменем fourth

        date  db "Ім'я            Стипендія   Середній бал       Вік        ", 0ah, 0dh,
              "Иванов  К.В.       70          4.7            21 ", 0ah, 0dh,
              "Петров  М.В.       50          5              35", 0ah, 0dh,
              "Сидоров Р.Д.       30          3.2            15 ", 0ah, 0dh,
              "Киселёв Е.З.       60          4              35",0ah,0dh,0
        mas dd 8 dup(?),0
        st1 db "Обчислити середній вік студентів в групі", 0
        ifmt db "Середній вік студентів: %d ",0
.code
Begin:
        invoke MessageBox, NULL, addr  date, addr st1, MB_ICONQUESTION
        xor edx, edx
        mov ecx, 4
        lea esi, first
next:
        mov eax, [esi]
        add max,al
        add esi, 13
        loop next
        movsx ax,max
        movsx eax,ax
        mov ebx,4
        div ebx
        mov sum,eax
        invoke  wsprintf,ADDR mas, ADDR ifmt, sum
        invoke MessageBox, NULL, addr  mas, addr st1,
MB_ICONINFORMATION
        invoke ExitProcess, 0
        end Begin
```

Результат виконання програми з лістингу 3.13.8 наведено на рис. 3.13.3.

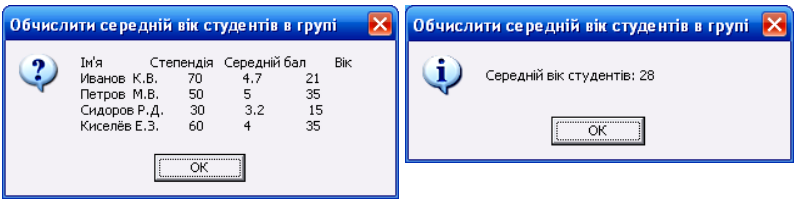


Рис. 3.13.3. Результат виконання програми з лістингу 3.13.3

3.14. Лабораторна робота 14
DLL-ФАЙЛИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням **Dll-файлів з використанням точки входу,** написаних мовою асемблера для програмування МП платформи x86. Дані, які виводяться, необхідно підписати. Вивести довідку про автора програми, номер варіанта, повністю все завдання та своєї e-mail.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати програму на асемблері обчислення одного з виразів:

1. Задано масив A з $N = 100$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L < Ai \le M$, де $L = 2$ та $M = 10$.

2. Задано масив A з $N = 20$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \le Ai < M$, де $L = 5$ та $M = 15$.

3. Задано масив A з $N = 30$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L < Ai < M$, де $L = 4$ та $M = 18$.

4. Задано масив A з $N = 40$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \le Ai \le M$, де $L = 5$ та $M = 20$.

5. Задано масив A з $N = 50$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \ge Ai > M$, де $L = 6$ та $M = 22$.

6. Задано масив A з $N = 60$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L > Ai \ge M$, де $L = 7$ та $M = 24$.

7. Задано масив A з $N = 70$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L > Ai > M$, де $L = 8$ та $M = 28$.

8. Задано масив A з $N = 80$ елементів. Навести алгоритм та програму визначення кількості елементів масиву A, які задовольняють умову $L \geq Ai \geq M$, де $L = 9$ та $M = 30$.

9. Задано масиви A і B по $N = 90$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai \leq Bi$, то $Ci = Ai \times Bi$; інакше – $Ci = Ai + Bi$.

10. Задано масиви A і B по $N = 100$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai \leq Bi$, то $Ci = Bi - Ai$; інакше – $Ci = Ai \times Bi$.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.14.1. Задано масиви A і B по $N = 9$ елементів. Навести алгоритм та програму формування масиву C за таким правилом: якщо $Ai \leq Bi$, то $Ci = Ai + Bi$; інакше – $Ci = Ai - Bi$.

Крок 1. У текстовому редакторі створимо програму з ім'ям lab10.asm. У цій програмі визначаються масиви чисел _A, _B, _C. Сама програма складається з набору частинних (крім ExitProcess) функцій, які розташовані в окремому файлі (lab10_dll.asm). Основна програма наведена в лістингу 3.14.1.

Лістинг 3.14.1: Основна програма з ім'ям lab10.asm:

```
title основна програма. Автор - Кутя Микита,  НТУ ХПІ, КІТ-28а
.386                    ; директива визначення типу мікропроцесора
.model  flat,stdcall ; задання лінійної моделі пам'яті  та угоди ОС Windows
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\\user32.inc   ; файли інтерфейсу …
include\masm32\include\kernel32.inc; файли систем. функцій застосувань…
; прототипи функцій бібліотеки, що підключаються:
  include lab10_dll.inc                          ; бібліотека констант
includelib \masm32\lib\kernel32.lib
includelib lab10_dll.lib   ; динамічна бібліотека, що підключається
.data
_A dd 8, 7, 6, 5, 4, 3, 2, 1       ; масив з початковими даними
```

```
        _B dd 1, 2, 3, 4, 5, 6, 7, 8       ; масив з початковими даними
        _C dd 8 dup(0)                     ; масив для занесення результату
_str db "Массив _A:",9, 8 dup(" %d "),10, 13, "Массив _B:",9, 8 dup(" %d "),\
10,13,"Массив _C:",9,8 dup(" %d "),0 ; перетворення ; 9 – символ табуляції
.code                                      ; директива початку сегмента даних
_start:
          invoke arrCompare, addr _A, addr _B, addr _C     ;формування
invoke arr2str,addr _A, addr _B, addr _C, addr _str      ; конвертація в рядок
invoke msgResult, addr _str        ; виведення вікна з результатами
invoke msgAuthor                   ; виведення відомостей про автора
invoke ExitProcess, 0              ; коректний вихід з програми
        end _start                                 ; закінчення програми з ім'ям start
```

Крок 2. У текстовому редакторі створимо файл з розширення .def. Цей файл lab10_dll.def буде таким, як у лістингу 3.14.2.

Лістинг 3.14.2. Файл lab10_dll.def:

```
LIBRARY lab10_dll     ; ім'я програми, яка включається до бібліотеки
EXPORTS arrCompare          ; ім'я процедури, яка викликається
EXPORTS arr2str             ; ім'я процедури, яка викликається
EXPORTS msgResult           ; ім'я процедури, яка викликається
EXPORTS msgAuthor           ; ім'я процедури, яка викликається
```

Крок 3. Створимо у текстовому редакторі програму з ім'ям lab10_dll.asm, в якій розкриємо всі окремі процедури. Така програма наведена в лістингу 3.14.3:

Лістинг 3.14.3. Програму з ім'ям lab10_dll.asm:

```
        title основна програма. Автор – Кутя Микита,  НТУ ХПІ, КІТ-28а
        .386                       ; директива визначення типу мікропроцесора
.model  flat,stdcall ; задання лінійної моделі пам'яті  та угоди ОС Windows
        option casemap:none        ; відмінність малих та великих літер
        include \masm32\include\windows.inc   ; файли структур, констант …
        include \masm32\macros\macros.asm
        uselib kernel32, user32;
public arrCompare
public arr2str
public msgResult
public msgAuthor
                ;;    прототипи функцій:    ;;
; функція порівняння двох масивів і заповнення третього:
arrCompare proto pmass_A:DWORD, pmass_B:DWORD,\
  pmass_res:DWORD
; Функція конвертації масиву чисел у символьний рядок. Аргументи –
                ; адреси масивів і рядка, що містить шаблон для форматування
```

```
arr2str proto pmass_A:DWORD,pmass_B:DWORD, pmass_C:DWORD, \
  pstr:DWORD
; функція виведення результату. Аргумент – рядок, що заздалегідь
                                        ; відформатований для вивведення:
msgResult proto pstr:DWORD
msgAuthor proto      ; функція виведення відомостей про автора

.data
hInst dd 0
_title_rez db "Результат:",0            ; заголовок вікна з результатом
_title_auth db "Об авторе",0   ; заголовок вікна відомостей про автора
                    ; вміст вікна відомостей про автора. Тут 9 – знак табуляції:
_str_auth db "Имя:", 9 ,"Кутя Никита", 10, 13, "Группа:",9,"КИТ-28а",0

.code

;   точка входу:
DllEntry proc instance:HINSTANCE, reason:DWORD, unused:DWORD
  mov eax, reason                ; перевірка причини виклику
  cmp eax, DLL_PROCESS_ATTACH
              ; якщо причина відрізняється від під'єднуваного процесу – вихід
  jnz _exit
  push instance    ;
  pop hInst          ; обов'язкове збереження хендлу програми
_exit:
    and eax, 1
    ret
DllEntry endp     ; закінчення процедури точки входу
; Тексти функцій:

msgAuthor proc
    invoke MessageBox, 0, addr _str_auth, addr _title_auth, MB_OK
    ret
msgAuthor endp

arrCompare proc pmass_A:DWORD, pmass_B:DWORD, pmass_res:DWORD
; збереження адрес аргументів функції
    mov esi, pmass_A
    mov edi, pmass_B
    mov edx, pmass_res
    mov ecx, 8        ; кількість елементів масиву –  8
m2:
; порівняння масивів відбувається за допомогою регістрів:
    mov eax, dword ptr[esi]
    mov ebx, dword ptr[edi]
; формування елементів третього масиву:
```

```
    .IF eax<= ebx
    add eax, ebx
    .ELSE
    sub eax, ebx
    .ENDIF
    mov dword ptr[edx], eax       ; результат зберігається в пам'яті
; перехід до наступного кроку ітерації:
    add esi, 4
    add edi, 4
    add edx, 4
    loop m2
  ret
arrCompare endp

                                        ; функція заповнення рядка з результатом.
                                  ; використовуються тільки останні два аргументи:
  arr2str proc pmass_A:DWORD,pmass_B:DWORD, pmass_C:DWORD, \
    pstr:DWORD
   mov ecx, 24  ; занесення загальної кількості елементів у трьох масивах
  ; перехід до останнього елемента  третього масиву:
      mov edi, pmass_C
      add edi,28 ; 8*4-4
  ; аргументи для wsprintf заносяться в стек з кінця:
  m1:
      push dword ptr[edi]
      sub edi, 4
      loop m1
      mov esi, pstr ; після занесення чисел заносимо в стек решту аргументів
   push esi          ; використовується один і той же рядок
   push esi
  call wsprintf              ;  виклик wsprintf
    ret
  arr2str endp
     msgResult proc pstr:DWORD
        invoke MessageBox, 0, pstr, addr _title_rez, MB_OK
        ret
     msgResult endp
end DllEntry
```

Крок 4. Створимо командний bat-файл для отримання **dll-файлу**, такого, як у лістингу 3.14.4. Для цього треба використати асемблерний файл **без мітки початку** програми.

Лістинг 3.14.4. Командний bat-файл з ім'ям, наприклад lab10_dll.bat, для створення файлу dll:

```
ml.exe /c /coff "lab10_dll.asm"
link.exe /DLL /DEF:lab10_dll.def "lab10_dll.obj"
```

Виконаємо lab10_dll.bat-файл. У результаті чого отримуються нові файли: lab10_dll**.dll**, lab10_ dll**.exp**, lab10_ dll**.lib** та lab10_ dll**.obj**.

Крок 5. Створимо ще один bat-файл з ім'ям, наприклад lab10.bat, для створення виконуючого exe-файлу:

```
ml.exe /c /coff "lab10.asm"
link.exe /subsystem:console "lab10.obj"
```

У результаті **виконання** файлу lab10.bat отримується файл lab10.exe, який й підключає приватну динамічну бібліотеку. Результат виконання програми з лістингу 3.14.1 наведено на рис. 3.14.1.

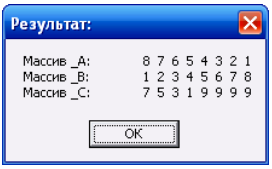


Рис. 3.14.3. Результат виконання програми з лістингу 3.14.1

3.15. Лабораторна робота 15
MMX-КОМАНДИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням команд MMX, написаних мовою асемблера для програмування МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати на асемблері програму обчислення одного з виразів з використанням цілих чисел за допомогою команд MMX, з виведенням результату обчислень в одне спрощене віконце за допомогою функції MessageBoxIndirect та прізвище й ініціали автора програми і свій e-mail.

Вхідний контроль знань

1. Виконати паралельне додавання 2-х масивів з 18 знакових даних розміром у байт.
2. Виконати паралельне віднімання 2-х масивів з 19 знакових даних розміром у байт.
3. Виконати паралельне пакування 2-х масивів з 10 знакових даних розміром у байт.
4. Виконати паралельне пакування 2-х масивів з 11 знакових даних розміром у слово.
5. Виконати паралельне пакування 2-х масивів з 10 знакових даних розміром у подвійне слово.
6. Виконати паралельне множення 2-х масивів з 11 знакових даних розміром у слово.
7. Виконати паралельну логічну операцію ”Виключне *АБО*” 2-х масивів по 15 знакових чисел.
8. Виконати командою pmulhw операцію над двома масивами.
9. Виконати командою pmullw операцію над двома масивами.
10. Виконати командою pmaddwd операцію над двома масивами.
11. Виконати паралельну логічну операцію *АБО* 2-х масивів з 14 знакових чисел.

12. Виконати паралельну порозрядну інверсію та множення командою pandn 2-х масивів з 13 знакових даних.

13. Виконати паралельне логічне множення 2-х масивів по 11 знакових чисел.

Завдання

1. Ввести з клавіатури два дійсних числа та виконати їх додавання. Якщо результат позитивний, то виконати паралельне порівняння за допомогою ММХ-команд 2-х масивів з 10 знакових цілих чисел розміром у байт. Якщо один масив більший від іншого, то виконати операцію

$$a - e/c - ab, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;\ e = 21{,}07;$$

інакше – виконати операцію ab.

2. Ввести з клавіатури два дійсних числа та виконати їх перемноження. Якщо результат менше 5, то виконати паралельне порівняння за допомогою ММХ-команд 2-х масивів по 10 знакових цілих чисел розміром у байт. Якщо один масив більший від іншого, то виконати операцію

$$(a - c)b + c/a, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;$$

інакше – виконати операцію $a - c$.

3. Ввести з клавіатури два дійсних числа та виконати їх додавання. Якщо результат негативний, то виконати паралельне порівняння за допомогою ММХ-команд 2-х масивів по 11 знакових цілих чисел розміром у байт. Якщо один масив більший від іншого, то виконати операцію

$$(a - e)b - d/b, \text{ де } a = 0{,}1;\ b = 1{,}05;\ c = 2{,}1;\ d = 3{,}2;$$

інакше – виконати операцію d/b.

4. Ввести з клавіатури два дійсних числа та виконати їх додавання. Якщо результат перевищує число 7, то виконати операцію паралельного логічного додавання за допомогою ММХ-команд над масивами цілих чисел. Якщо друге слово більше 55, то виконати операцію

$$a - e/b - de, \text{ де } a = 0{,}2;\ b = 8{,}05;\ c = 2{,}2;\ d = 3{,}3;$$

інакше – виконати операцію $a - e/b$.

5. Ввести з клавіатури два дійсних числа та виконати їх віднімання. Якщо результат позитивний, то виконати операцію паралельного логічного множення за допомогою ММХ-команд над масивами цілих чисел. Якщо друге слово більше 45, то виконати операцію

$$(b + d)/ba - c, \text{ де } a = 0{,}2;\ b = 7{,}05;\ c = 2{,}3;\ d = 3{,}4;$$

інакше – виконати операцію $b + d$.

6. Ввести з клавіатури два дійсних числа та виконати їх віднімання. Якщо результат негативний, то виконати операцію паралельного додавання pxor за модулем за допомогою ММХ-команд над масивами 11 цілих чисел. Якщо друге слово менше 15, то виконати операцію

$(b - dc)/a - c$, де $a = 0{,}2$; $b = 6{,}05$; $c = 1{,}3$; $d = 3{,}5$;
інакше – виконати операцію $b - dc$.

7. Ввести з клавіатури дійсне число та добути з нього корінь. Якщо результат менше 7, то виконати команду логічного зсуву ліворуч pslld масиву з 10 чисел та залежно від значення другого байта виконати операцію

$ae - d/c - b$, де $a = 1{,}2$; $b = 5{,}05$; $c = 0{,}3$; $d = 3{,}5$;
інакше – виконати операцію d/c.

8. Ввести з клавіатури дійсне число. Якщо число позитивне, то виконати команду арифметичного зсуву елементів даних праворуч psraw масиву з 9 чисел та залежно від значення другого байта виконати операцію

$(ab - d)/b - d$, де $a = 3{,}6$; $b = 9{,}17$; $c = 2{,}8$; $d = 3{,}9$;
інакше – виконати операцію $ab - d$.

9. Ввести з клавіатури дійсне число. Якщо число позитивне, то виконати команду логічного зсуву елементів даних праворуч psrlw масиву з 9 чисел та залежно від значення другого байта виконати операцію: (de – e)/(c – a), де $a = 3{,}1$; $b = 9{,}11$; $c = 2{,}3$; d = 5,5;
інакше – виконати операцію de – e.

10. Ввести з клавіатури дійсне число. Якщо число більше 10, то виконати паралельне пакування за допомогою MMX-команд 2-х масивів по 11 чисел та залежно від значення другого байта виконати операцію

$a/b - cd$, де $a = 4{,}5$; $b = 12{,}3$; $c = 2{,}5$; $d = 6{,}5$;
інакше – виконати операцію a/b.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій під Win32 та детальним коментарем і описом роботи.
4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.15.1. Виконати паралельне порівняння 2-х масивів по 8 знакових даних розміром у байт. Якщо старші половини масивів однакові, то виконати операцію

$a - b/4$, де $a = 0{,}2$; $b = 10{,}05$;
інакше – операцію не виконувати.

Програму обчислення прикладу наведено у лістингу 3.15.1.

Лістинг 3.15.1. Програма з виведенням результату у спрощене вікно:

```
 title CopyRight by Rysovaniy A. N.
 .686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
 .MMX                    ; директива визначення команд MMX
option casemap: none     ; відмінність малих та великих літер
include \masm32\include\windows.inc  ; файли структур, констант …
include \masm32\macros\macros.asm
uselib kernel32, user32, fpu;
.data                    ; директива визначення даних
  str1  DB -1, -2, 3, 4, 0, 1, 12, -8
  str2  DB -1, -2, 3, 4, 0, 1, 12, -8
  res  DB 8 DUP (5)
  a1 DWORD  0.2
  b1 DWORD  10.05
  c1 DWORD  4.
  res2 DD ?
st1 db "result",0        ; назва віконця
st2 db 10 dup(?),0       ; буфер чисел для виведення повідомлення
.code                    ; директива початку коду програми
 _st:                    ; мітка початку програми з ім'ям _st
  movq   MM0, qword ptr str1   ; копіювання 64-розрядного числа
  pcmpeqb MM0, qword ptr str2
  movq   qword ptr res, MM0    ; збереження результату
emms                     ; остання MMX-команда (для співпроцесора)
  add dword ptr res,0          ; порівняння старшої частини res
  jz m1                ; перехід на m1, якщо старша частина дорівнює нулю
finit                    ; ініціювання співпроцесора
  fld b1             ; завантаження у вершину стека ST(0) змінної b1
  fld c1             ; завантаження у вершину стека ST(0) змінної c1
  fdiv           ; st(0) := st(1)/st(0)
  fld a1             ; завантаження у вершину стека ST(0) змінної a1
  fsub st(0),st(1)         ; st(0) := st(0) – st(1)
  ;fst res2                ; можливе збереження у пам'яті
m1:
invoke FpuFLtoA, \       ; API-функція перетворення 80-розрядного числа
 0, 10,\  ; адреса числа та  кількість десяткових знаків після коми (10)
  ADDR st2, \        ; адреса буфера для символів, які перетворяться
  SRC1_FPU or SRC2_DIMM ; адреса 80-розрядного числа та саме число
invoke MessageBox, \     ; API-функція виведення вікна консолі
 NULL, \                 ; hwnd – ідентифікатор вікна
   addr st2, \           ; адреса рядка, яка містить  текст повідомлення
    addr st1, \ ; адреса рядка, яка містить  заголовок повідомлення
```

```
        MB_OK                ; вигляд діалогового вікна

invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
        end _st              ; директива закінчення програми
```

Результат виконання програми з лістингу 3.15.1, отриманий за допомогою API-функції MessageBox, наведено у вигляді вікна на рис. 3.15.1.

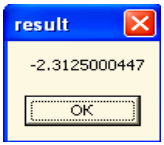


Рис. 3.15.1. Результат виконання програми з лістингу 3.15.1

За назву вікна відповідає параметр addr st1.

У випадку необхідності порівняння всього масиву з 8-ма чисел, розміром у байт можна використати порівняння за два етапи. Такий прийом здійснено у прикладі 3.15.1.

Приклад 3.15.2. Виконати паралельне порівняння 2-х масивів по 8 знакових цілих чисел розміром у байт кожне. Якщо один масив більший від іншого, то виконати операцію

$$(a - c)/b + c, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;$$

інакше – виконати операцію $a - c$.

Лістинг 3.15.2. Програма обчислення прикладу 3.15.2:

```
        title CopyRight by Rysovaniy A. N.                rysov@rambler.ru
        .686                 ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
        .MMX                         ; директива визначення команд MMX
option casemap: none        ; відмінність малих та великих літер
      include \masm32\include\windows.inc   ; файли структур, констант …
      include \masm32\macros\macros.asm
      uselib kernel32, user32, fpu;
      BSIZE equ  30                  ; задання реальної кількості байтів

.data                                ; директива визначення даних
  str1  db -1, -2, 3, 2, 5, 1, 12, -8 ; резервування в пам'яті комірок для str1
```

```
  str2  db -1, -2, 3, 2, 0, 1, 12, -8 ;
  res  DD 8 DUP (0)     ; резервування в пам'яті комірок для res
  a1 DWORD  0.2         ; константа a
  b1 DWORD  10.05   ; константа b
  c1 DWORD  2.3         ; константа c
   stdout DWORD ?           ; резервування в пам'яті 32-розрядної комірки
                          ; з ім'ям stdout для збереження дескриптора виведення
   cWritten  DWORD  ?       ; для адреси символів виведення
buf  BYTE BSIZE dup (?) ; кількість байтів для запису послідовності символів
res2 DWORD ?

.code          ; директива початку коду програми
  _st:          ; мітка початку програми з ім'ям _st
       lea esi,str1          ; занесення в esi адреси масиву str1
       lea edi,str2          ; занесення в edi адреси масиву str2
       movq    M0, qword ptr [esi]            ; занесення 8-ми байтів до Pr MM0
       pcmpeqb MM0, qword ptr [edi]          ; перевірка на рівняння
       movq  qword ptr res, MM0          ; збереження результату порівняння
       emms                                ; визвільнити  регістри MMX
     cmp dword ptr res,0                  ; порівняння старшої частини результату
       jnz m1                           ; перейти на m1, якщо результат не нульовий
     cmp dword ptr res+4,0       ; порівняння молодшої частини результату
       jnz m1                           ; перейти на m1, якщо результат не нульовий
  finit                               ; ініціалізація співпроцесора
     fld c1                      ; занесення у вершину стека st(0) змінної c1
     fld b1                      ; занесення у вершину стека st(0) змінної b1
     fld a1                      ; занесення у вершину стека st(0) змінної a1
       fsub st(0),st(2)                      ; a – c  та збереження в st(0)
       fdiv st(0),st(1)                      ; (a – c)/b  та збереження в st(0)
       fadd st(0),st(2)                      ; (a – c)/b + a  та збереження в st(0)
       fst res2               ; збереження вершини стека в пам'яті з ім'ям res2
       jmp m2               ; безумовний перехід на мітку m2
m1:
     finit                  ; ініціалізація співпроцесора
     fld c1                           ; занесення у вершину стека st(0) змінної c1
     fld a1                           ; занесення у вершину стека st(0) змінної a1
       fsub st(0),st(1)      ; a1 – c1 та збереження результату у вершині стека
       fst res2                ; збереження вершини стека в пам'яті з ім'ям res2
m2:
     invoke  GetStdHandle, STD_OUTPUT_HANDLE
     mov  stdout, eax       ; збереження одержаного дескриптора у пам'яті
invoke FpuFLtoA, \  ; API-функція перетворення 80-розрядного числа
  0, 10,\ ; адреса числа, що відображається та кількість знаків
     ADDR buf, \      ; адреса буфера для символів, які перетворяться
     SRC1_FPU or SRC2_DIMM   ; адреса 80-розрядного числа та саме число
```

```
invoke  WriteConsoleA, stdout, \
     ADDR buf, BSIZE, \ ; адреса початку повідомлення та розмір
          ADDR cWritten, 0\    ; адреса, де зберігається кількість символів
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
        end _st          ; директива закінчення програми
```

Результат виконання програми з лістингу 3.15.2 наведено на рис. 3.15.2.

Рис. 3.15.2. Результат виконання програми з лістингу 3.15.2

Правильність отриманих результатів підтверджено результатами, одержаними на калькуляторі.

3.16. Лабораторна робота 16
SSE-КОМАНДИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням команд SSE, написаних мовою асемблера для програмування МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант та написати на асемблері за допомогою SSE-команд програму обчислення одного з виразів з використанням дійсних чисел та ***виведенням варіанта завдання та його вмісту, результату і прізвища автора*** програми на екран за допомогою функції MessageBoxIndirect.

Методичні рекомендації

Програми з використанням команд SSE першого доповнення можна виконувати на masm32. Однак вважається доцільним виконувати асемблювання та лінкування в прогресивнішому середовищі, в такому, як Visual Studio 2008.

Асемблювання та компонування через безпосереднє звертання до Visual Studio 2008

Послідовність дій може складатися з декількох етапів.

Етап 1. Настроювання асемблера ml.exe.

Відкрити програму Visual Studio 2008 та прописати шлях находження ml.exe та його параметрів.

Вибрати в меню програми Tools\Externel Tools, у вікні, яке з'явиться (рис. 3.16.1), натиснути Add. У рядок Title написати ім'я, наприклад masm32.

У рядку Command натиснути на клавішу  та вказати шлях находження ml.exe: C:\masm32\bin\ml.exe.

У рядок Arguments вставити параметри асемблювання: /c /coff /Cp /nologo /Fm /Zi /Zd /Fl $(ItemFileName).asm. Обов'язково перевірити, щоб перед першим символом параметра не було пропуску.

У рядок Initial directory прописати параметр $(ItemDir).

Зняти прапорець Close on exit.

Натиснути на кнопку Apply.

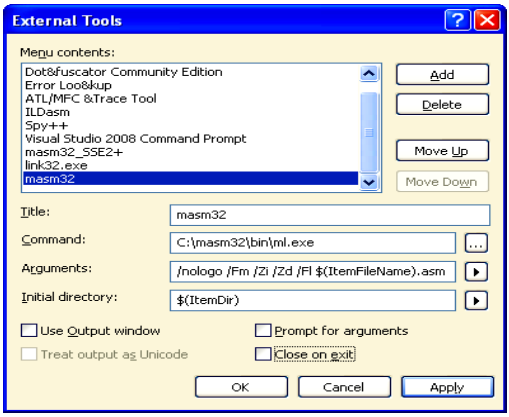


Рис. 3.16.1. Вигляд вікна прописування шляхів та параметрів асемблювання Visual Studio 2008 для masm32

Етап 2. Настроювання лінкера link.exe.

Для цього необхідно прописати шлях знаходження файлу link.exe та його параметрів.

У вікні External Tools натиснути Add. У рядку Title написати ім'я, наприклад link32.exe. У рядок Title написати ім'я, наприклад link32 (рис. 3.16.2).

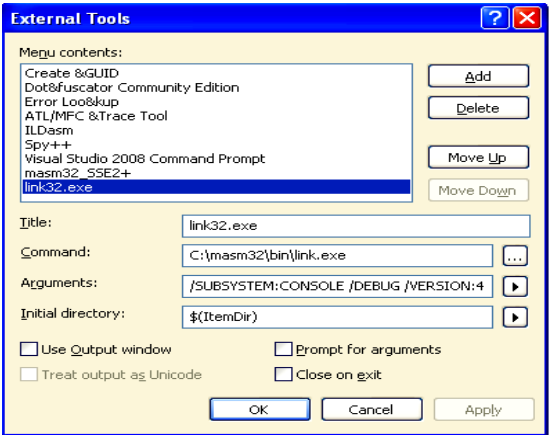


Рис. 3.16.2. Вигляд командного рядка Visual Studio 2008 Command Prompt

У рядку Command натиснути на клавішу … та вказати шлях знаходження link: C:\masm32\bin\link.exe.

У рядок Arguments вставити параметри асемблювання /SUBSYSTEM:CONSOLE /DEBUG /VERSION:4.0 $(ItemFileName).obj.

У рядок Initial directory прописати параметр $(ItemDir).

Зняти прапорець Close on exit.

Натиснути на кнопку Apply.

Етап 3. Виконання програми.

Для цього необхідно завантажити файл у Visual Studio 2008: File\Open\File…

Виконати **асемблювання**: Tools\ masm32. У результаті таких дій створиться .obj-файл, який розташується за адресою C:\masm32\bin\sse1_2.obj.

Виконати **лінкування**: Tools\ link32.exe. Натиснути Enter. Буде створено .exe-файл, який знаходитиметься за адресою C:\masm32\bin\sse1_2.exe.

Завдання

1. Виконати паралельне порівняння за допомогою SSE-команд 2-х масивів по 11 чисел. Якщо всі числа другого масиву більші від усіх чисел першого, то виконати операцію

$$(2a - e)b^2 - d/\sqrt{b}, \text{ де } a = 0{,}1;\ b = 1{,}05;\ c = 2{,}1;\ d = 3{,}2;$$

інакше – виконати операцію $d/\sqrt{b}$.

2. Виконати операцію паралельного логічного множення чисел за допомогою SSE-команд. Якщо всі числа другого масиву більші від першого, то виконати операцію

$$(de - 2e)/(c - 2^a), \text{ де } a = 3{,}1;\ b = 9{,}11;\ c = 2{,}3;\ d = 5{,}5;$$

інакше – виконати операцію $de - 2e$.

3. Виконати паралельне порівняння за допомогою SSE-команд 2-х масивів по 10 чисел. Якщо всі числа першого масиву більші від другого, то виконати операцію

$$(2^a - c)/b + 2^c, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;$$

інакше – виконати операцію $2^a - c$.

4. Виконати паралельне порівняння за допомогою SSE-команд 2-х масивів по 9 чисел. Якщо всі числа першого масиву менші від другого, то виконати операцію

$$2^a - e/c^3 - ab, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;\ e = 21{,}07;$$

інакше – виконати операцію e/c^3.

5. Виконати паралельну операцію логічного *АБО* за допомогою SSE-команд 2-х масивів. Якщо всі значення другого числа результату більше 25, то виконати операцію

$$2^a/b - cd^3, \text{ де } a = 4{,}5;\ b = 12{,}3;\ c = 2{,}5;\ d = 6{,}5;$$

інакше – виконати операцію $2^a/b$.

6. Виконати операцію паралельного логічного додавання за допомогою SSE-команд над масивами чисел. Якщо друге число результату порівняння більше 55, то виконати операцію

$a - e/b - 2^d e$, де $a = 0{,}2$; $b = 8{,}05$; $c = 2{,}2$; $d = 3{,}3$;

інакше – виконати операцію $2^d e$.

7. Виконати операцію паралельного логічного множення за допомогою SSE-команд над масивами чисел. Якщо друге число результату порівняння більше 555, то виконати операцію

$(b + 4d)/ba - 2^c$, де $a = 0{,}2$; $b = 7{,}05$; $c = 2{,}3$; $d = 3{,}4$;

інакше – виконати операцію $b + 4d$.

8. Виконати паралельне обчислення квадратного кореня з масиву чисел і додати всі отримані результати. Якщо 0 та 1 біти дорівнюють одиницям, то виконати операцію

$a^2 e - d/c - 2^b$, де $a = 1{,}2$; $b = 5{,}05$; $c = 0{,}3$; $d = 3{,}5$;

інакше – виконати операцію $a^2 e$.

9. Виконати паралельне обчислення мінімального значення для 2-х масивів. Якщо всі числа першого масиву менші від другого, то виконати операцію

$(ab^3 - d)/b - 2^d$, де $a = 3{,}6$; $b = 9{,}17$; $c = 2{,}8$; $d = 3{,}9$;

інакше – виконати операцію $ab^3 - d$.

10. Виконати операцію паралельного додавання за модулем за допомогою SSE-команд над масивами цілих чисел. Якщо друге слово менше 15, то виконати операцію

$(b - d\sqrt{c})/2^a - c$, де $a = 0{,}2$; $b = 6{,}05$; $c = 1{,}3$; $d = 3{,}5$;

інакше – виконати операцію $b - d\sqrt{c}$.

Зміст звіту

1. Постановка задачі для конкретного варіанта.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій та з детальним коментарем і описом роботи.
4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

Приклад 3.16.1. Виконати паралельне порівняння за допомогою SSE-команд 2-х масивів по 8 чисел. Якщо перший масив більший від другого,

то виконати операцію $(a - c)b - d/b$, де $a = 3{,}0$; $b = 0.2$; $c = 1{,}0$; $d = 2{,}2$; інакше – виконати операцію d/b.

Лістинг 3.16.1:

```
      title Rysovaniy A.N.                                  rysov@rambler.ru
      .686                        ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
      .XMM                                ; директива визначення команд SSE
option casemap:none                ; відмінність малих та великих літер
include \masm32\include\\user32.inc   ; файли інтерфейсу …
include \masm32\include\kernel32.inc ; системні функції застосувань…
include \masm32\include\fpu.inc
include \masm32\include\windows.inc ; файли структур, констант …
includelib \masm32\lib\user32.lib
includelib \masm32\fpulib\fpu.lib
includelib \masm32\lib\kernel32.lib
  BSIZE equ 14                          ; задання реальної кількості байтів

.data                                   ; директива визначення даних
      mas1  dd  1.4,  2.1, 3.8, 4.6, 5.4, 6.12,7.54, 8.1,  9.0,  10.65
      mas2  dd -1.34,-5.0,-3.54,1.5,-5.8,-6.53,7.5, -8.34,-9.54,-10.1
      len equ ($-mas2)/ type mas2  ; кількість чисел масиву mas2
      a   dd  3.0 ;
      b   dd  0.2 ;
      c1  dd  1.0 ;
      d   dd  2.2 ;
info db "x1 = "                    ; підпис числа для віконця результату
buf BYTE BSIZE dup(?),0 ; розмір буфера для символів, які перетворяться
      st2 db "Результат",0              ; назва віконця

.code                                   ; директива початку коду програми
_st:                                    ; мітка початку програми з ім'ям _st
      mov eax, len             ; кількість подвійних слів у масиві
      mov ebx,4                ; кількість подвійних слів у XMM регістрі
      xor edx,edx              ; підготування до ділення
      div ebx ; визначення кількості циклів для паралельного обробки
      mov ecx,eax               ; лічильник циклів для паралельного зчитування
      lea esi,mas1             ; занесення в esi адреси масиву mas1
      lea edi,mas2             ; занесення в esi адреси масиву mas2

next:  movups XMM0,[esi]     ; пересилання 4-х 32 чисел з mas1
      movups XMM1,[edi]       ; пересилання 4-х 32 чисел з mas2
      cmpltps XMM0,XMM1   ; порівняння на менше: якщо менше, то нулі
      movmskps ebx,XMM0   ; перенесення знакових бітів
      add esi,16           ; підготування адреси для нового зчитування mas1
      add edi,16          ; підготування адреси для нового зчитування mas2
```

```
        dec ecx       ; зменшення лічильника циклів паралельного зчитування
        jnz m1                ; перевірка лічильника на ненульове значення
        jmp m2                ;
m1:  shl ebx,4                ; зсув ліворуч на 4 біти
        jmp next              ; на новий цикл
m2:  cmp edx,0                        ; перевірка остачі
        jz _end               ; якщо в остачі нуль, то на мітку _end
        mov ecx,edx   ; якщо в остачі не нуль, то встановлення лічильника
m4:
movss XMM0,dword ptr[esi] ; пересилання одного 32-розр. числа з mas1
movss XMM1,dword ptr[edi] ; пересилання одного 32-розр. числа з mas2
        comiss XMM0,XMM1              ; порівняння молодших чисел масивів
        jg m6                 ; якщо більше
        shl ebx,1             ; зсув ліворуч на 1 розряд
        or ebx,1              ; встановлення 1, оскільки XMM0[0] < XMM1[0]
        jmp m3
m6:
        shl ebx,1     ; зсув ліворуч на 1 розряд
m3:
        add esi,4             ; адреса для нового числа mas1
        add edi,4             ; адреса для нового числа mas2
        loop m4
_end:   cmp ebx,0             ; перевірка знакових бітів
        finit
        jz mb                 ; якщо ebx = 0, то перейти на мітку mb
        fld d                 ; занесення d
        fdiv b                ; d/b
        jmp m5
mb:
        fld  a                ;
        fsub c1               ; a – c
        fmul b                ; (a – c)b
        fld  d                ; занесення d
        fdiv b                ; d/b
        fsubp  st(1),st(0)    ; (a – c)b – d/b
m5:
invoke  FpuFLtoA, 0, 10,\ ; адреса числа та кількість знаків після коми (10)
 ADDR buf, SRC1_FPU or SRC2_DIMM ; 1-е число – в FPU, результат – число
invoke MessageBox,NULL,addr info,ADDR st2,MB_ICONINFORMATION
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
        end _st               ; директива закінчення програми
```

Результат виконання програми з лістингу 3.16.1 наведено на рис. 3.16.3.

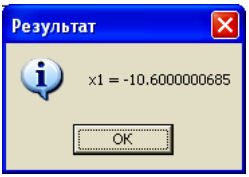


Рис. 3.16.3. Результат виконання
програми з лістингу 3.16.1

У програмі для паралельного порівняння чотирьох 32-розрядних чисел використовується цикл з міткою next. У регістр ebx записуються всі знакові біти чисел, які перевіряються. Кількість масивів, які не перевіряються паралельно, залежно від змісту регістра edx обробляються поодинці (скалярно).

3.17. Лабораторна робота 17
SSE2-КОМАНДИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням команд SSE, написаних мовою асемблера для програмування МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати свій варіант задачі та написати на асемблері за допомогою SSE2-команд програму обчислення одного з виразів з використанням дійсних чисел, ***виведенням на екран варіанта завдання, його вмісту, результату, прізвища та e-mail*** автора програми.

Методичні рекомендації

Асемблер masm32v10 не підтримує команди SSE2 та вищі. У такій ситуації можна діяти двома шляхами: через безпосередній запис до каталогу програми Visual Studio 2008 та через безпосереднє звертання до середовища Visual Studio 2008.

Асемблювання через безпосередній запис до каталогу Visual Studio 2008 та компонування через masm32

Перший й самий простий шлях – переписати файл з розширенням .asm у встановлену програму Microsoft Visual Studio 2008 за такою адресою: c:\Program Files\ Microsoft Visual Studio 9.0\VC\bin\. У цій директорії знаходиться програма ml.exe, яка виконує асемблювання з параметрами ml /c /coff ім'я_прог.asm. При виконанні цієї дії програма ml.exe звертається до **своїх** Visual Studio-бібліотек і тому не можна просто змінити в masm32 файл ml.exe. Треба виконувати асемблювання тільки в Microsoft Visual Studio 2008, так, як, наприклад, наведено на рис. 3.17.1.

У такому випадку отримується необхідний .obj-файл, який після натискання на кнопку Enter буде знаходитися в цій же директорії.

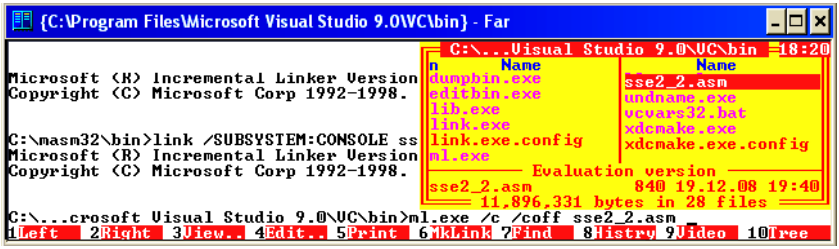


Рис. 3.17.1. Отримання файлу .obj в Visual Studio 2008 через файловий менеджер FAR

Отриманий .obj-файл необхідно переписати у програму masm32 та вже там отримати .exe-файл, для чого необхідно вказати параметри link /SUBSYSTEM:CONSOLE ім'я_прог.obj, так, як, наприклад, показано на рис. 3.17.2.

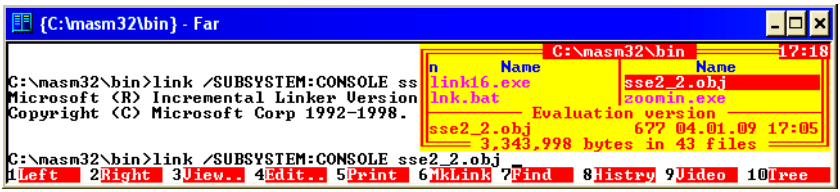


Рис. 3.17.2. Отримання файлу .exe через файловий менеджер FAR

Асемблювання та компонування через безпосереднє звертання до Visual Studio 2010

Послідовність дій може складатися з декількох етапів.

Для отримання виконуваного файлу початковий asm-файл зручніше розмістити за адресою C:\Program Files\ Microsoft Visual Studio 10.0\VC.

Етап 1. Настроювання асемблера ml.exe

Відкрити програму Visual Studio 2010 та прописати шлях знаходження ml.exe та його параметрів.

Вибрати в меню програми Tools\Externel Tools та натиснути Add. У рядок Title написати якесь ім'я, наприклад masm32_SSE2+.

У рядку Command натиснути на клавішу  та вказати шлях знаходження ml.exe: C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\ml.exe.

У рядок Arguments вставити параметри асемблювання: **-c -coff -Fl -Fo $(ItemFileName).obj $(ItemFileName).asm.** Обов'язково перевірити, щоб перед першим символом параметра не було пропуску.

У рядок Initial directory прописати параметр $(ItemDir)
Зняти прапорець Close on exit.
Натиснути на кнопку Apply та кнопку OK (рис. 3.17.3).

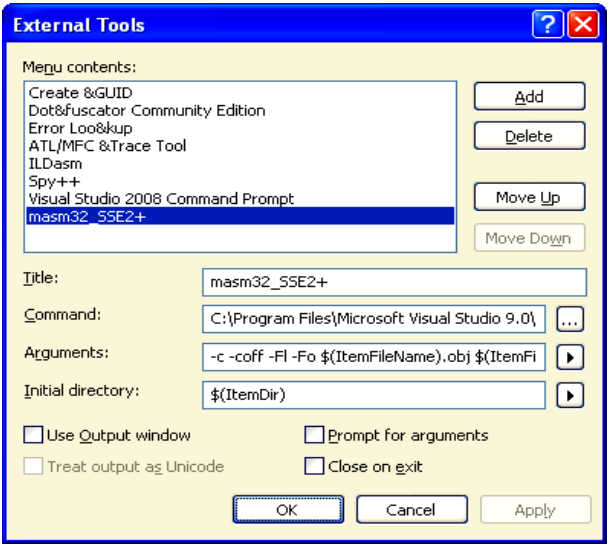


Рис. 3.17.3. Вигляд вікна прописування шляхів та параметрів асемблювання Visual Studio 2010 для команд SSE2+

Етап 2. Настроювання лінкера link.exe

Для цього необхідно прописати шлях знаходження файлу link.exe та його параметри. У зв'язку зі складністю параметрів для лінкера link.exe з програми Visual Studio 2010 простіше безпосередньо їх вводити в командний рядок (процес введення параметрів у командний рядок розглянуто в 3-му етапі).

Етап 3. Виконання програми

Для цього необхідно завантажити файл у Visual Studio 2008: File\ Open\File…

Виконати **асемблювання**: Tools\ masm32_SSE2+. У результаті таких дій *за змістом знаходження* asm-файлу створиться .obj-файл, який розташується, наприклад, за адресою C:\Program Files\Microsoft Visual Studio 10.0\VC\sse2_3.obj або C:\mas32\bin\sse2_3.obj.

Після виправлення помилок необхідно обов'язково знову зберігати файл : File\Save ... (рис. 3.17.4).

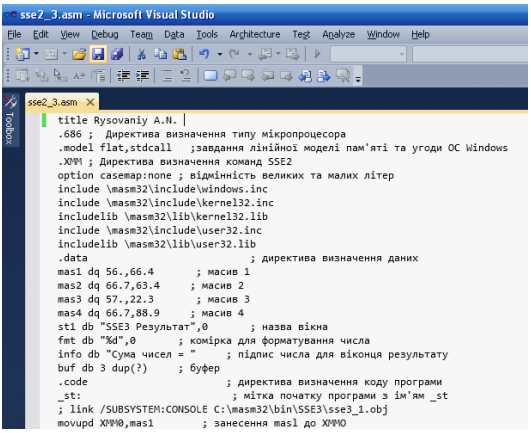


Рис. 3.17.4. Зберігання файлу в середовищі Visual Studio 2010

Для виконання **лінкування** необхідно викликати командний рядок Visual Studio 2010, який знаходиться шляхом послідовного натискання кнопок Пуск\ Все программы\ Microsoft Visual Studio 2010\, Visual Studio Tools\ Visual Studio Command Prompt (2010). Після того, як з'явиться вікно для введення командного рядка необхідно спочатку скопіювати команду для лінкування, наприклад: **link /SUBSYSTEM:CONSOLE sse2_3.obj** (з адресою знаходження .obj-файлу**),** а потім натиснути праву кнопку миші та вставити (рис. 3.17.5).

Натиснути клавішу Enter. Буде створено .exe-файл, який знаходитиметься за такою адресою: C:\Program Files**Microsoft Visual Studio 10.0\VC** sse2_3.exe.

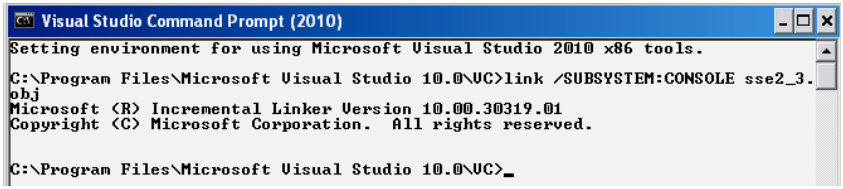


Рис. 3.17.5. Вигляд командного рядка Visual Studio 2008 Command Prompt

Завдання

1. Виконати паралельне порівняння за допомогою SSE2-команд 2-х масивів по 11 чисел. Якщо всі числа другого масиву більші від першого,

то виконати пошук максимального значення в парах упакованих чисел, а якщо навпаки – то мінімального.

2. Задано два масиви по 5 чисел. За допомогою SSE2-команд визначити суму чисел всіх масивів та виконати аналіз чисел 0 та 1 результату. Якщо вони нульові, то виконати операцію добування квадратного кореня з нього.

3. Виконати паралельне порівняння за допомогою SSE2-команд 2-х масивів по 10 чисел. Якщо перший масив більший від другого, то виконати операцію

$$(a - 3e)b - d/\sqrt{b}, \text{ де } a = 0{,}1;\ b = 1{,}05;\ c = 2{,}1;\ d = 3{,}2;$$

інакше – виконати операцію d/b.

4. Виконати паралельне порівняння за допомогою SSE2-команд 2-х масивів по 9 чисел. Якщо перший масив менший від другого, то виконати операцію

$$(4a - \sqrt{e})b - d/b, \text{ де } a = 0{,}1;\ b = 1{,}05;\ c = 2{,}1;\ d = 3{,}2;$$

інакше – виконати операцію d/b.

5. Виконати за допомогою SSE2-команд паралельне додавання за модулем 2-х масивів по 13 чисел. Якщо сума всіх отриманих чисел більше 255, то виконати операцію

$$(a - 5c)/b + \sqrt{c}, \text{ де } a = 0{,}2;\ b = 10{,}05;\ c = 2{,}3;$$

інакше – виконати операцію $a - c$.

6. Виконати операцію паралельного логічного перемноження 2-х масивів по 10 чисел. Додати всі результати. Якщо сума всіх отриманих чисел більше 100, то виконати обчислення квадратного кореня, а якщо навпаки – не виконувати.

7. Виконати паралельне множення двох упакованих 64-розрядних чисел з плаваючою точкою подвійної точності. Додати всі результати та обчислити квадратний корінь.

8. Виконати паралельне ділення двох упакованих 64-розрядних чисел з плаваючою точкою подвійної точності. Додати всі результати та обчислити квадратний корінь.

9. Виконати паралельний пошук мінімального значення в парах упакованих 64-розрядних чисел з плаваючою точкою подвійної точності. Додати всі результати та обчислити квадратний корінь.

10. Виконати паралельне добування квадратного кореня з двох упакованих 64-розрядних чисел з плаваючою точкою подвійної точності. Додати всі результати та обчислити квадратний корінь.

Зміст звіту

1. Постановка задачі для конкретного варіанта.

2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.

3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій та з детальним коментарем і описом роботи.

4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.

5. Короткий опис виконання програми.

6. Висновки за результатами роботи.

Приклад 3.17.1. Виконати паралельне порівняння за допомогою SSE2-команд 2-х масивів по 9 чисел. Якщо перший масив менший від другого, то виконати операцію

$$(a - e)b - d/b, \text{ де } a = 0{,}1;\ b = 1{,}05;\ c = 2{,}1;\ d = 3{,}2;$$

інакше – виконати операцію d/b.

Лістинг 3.17.1:

```
    title Rysovaniy A.N. & Хмеленко Д., НТУ “ХПІ”, КІТ-17а
    .386                        ; директива визначення типу мікропроцесора
.model flat,stdcall  ; задання лінійної моделі пам’яті та угоди ОС Windows
    .XMM                        ; використання регістрів SSE
    option casemap:none         ; відмінність малих та великих літер
include \masm32\include\kernel32.inc ; файли систем. функцій застосувань…
    include \masm32\include\windows.inc ; файли структур, констант …
    include \masm32\include\\user32.inc   ; файли інтерфейсу …
    include \masm32\include\fpu.inc
    includelib \masm32\lib\fpu.lib
    includelib \masm32\lib\user32.lib
    includelib \masm32\lib\kernel32.lib
    BSIZE equ 50

.data
 mas1 dd  11.4, 14.02, 9.8, 4.0, 12.14, 1.82, 67.54, 9.95, 7.16     ; масив №1
       len EQU $- mas1
mas2 dd   11.4, 14.02, 3.14, 2.718, 5.0, 45.45, 23.35, 9.95, 79.2 ; масив №2
      rez dd len dup(0)      ; результат порівняння
      rez2 dq ?              ; для результату виконання операцій
      a1 dd 0.10
      b1 dd 1.05
      c1 dd 2.10
      d1 dd 3.20
      buf BYTE BSIZE dup(?)
      st1 db "The Rezult mas1!=mas2",0          ; заголовки вікна повідомлень
      st2 db "The Rezult mas1==mas2",0
      st3 dd 0
```

```
      flag db 0
    .code
    _st:
        xor edx,edx
        mov eax,len       ; завантаження 40 байтів масиву
        shr eax,2         ; len/4 = 10  – визначення кількості чисел
        mov ebx,4
        div ebx           ; визначення кількості циклів
        mov ecx,eax
        lea esi,mas1      ; завантаження масиву №1 в esi
        lea edi,mas2      ; завантаження масиву №2 в edi
        xor ebx,ebx
    .WHILE(ecx!=0)
        movupd XMM0,[esi]     ; завантаження масиву №1 в регістр XMM0
        movupd XMM1,[edi]     ; завантаження масиву №1 в регістр XMM0
     cmppd XMM0, XMM1, 0 ; виконання паралельного порівняння масивів
        movupd  rez[ebx],XMM0   ; збереження результату порівняння
        add esi,16                ; перехід на наступну адресу
        add edi,16
        add ebx,16
        dec ecx
    .ENDW
  mov ecx,edx ; залишок чисел, що не ввійшли до паралельного порівняння
.WHILE(ecx!=0)
        movupd XMM0,[esi]      ; завантаження масиву №1 в регістр XMM0
       movupd XMM1,[edi]       ; завантаження масиву №1 в регістр XMM0
       cmpsd XMM0, XMM1,8    ; виконання скалярного порівняння масивів
        movupd  rez[ebx],XMM0       ; збереження результату порівняння
        add esi,4                   ; перехід на наступну адресу
        add edi,4
        add ebx,4
        dec ecx
.ENDW
    mov ecx,9
    xor ebx,ebx
    .WHILE(ecx!=0)
.IF (rez[ebx]==0)   ; якщо елементи масивів хоча б один раз не дорівнюють
        mov al,1   ; одне одному, то встановлюємо прапорець в 1
        mov flag,al
.ENDIF
        dec ecx
        inc ebx
.ENDW
    .IF (flag==1)    ; якщо масиви не рівні, виконуємо вираз (a – c)b – d/b
        finit
        fld a1
```

```
        fsub c1
        fmul b1
        fld d1
        fdiv b1
        fsub
        fst rez2
        lea ebx,st1
    .ELSE          ; якщо ж масиви рівні, то виконуємо d/b
        finit
        fld d1
        fdiv b1
        fst rez2
        lea ebx,st2
    .ENDIF
        mov st3, ebx                          ; збереження заголовка вікна
invoke FpuFLtoA,0,10,addr buf,SRC1_FPU or SRC2_DIMM ; перетворення
invoke MessageBox, NULL, ADDR buf,  st3, MB_OK   ; виведення на екран
      invoke ExitProcess,0
       end _st
```

Результат виконання програми з лістингу 317.1 наведено на рис. 3.17.6.

Рис. 3.17.6. Результат виконання
програми з лістингу 3.17.1

3.18. Лабораторна робота 18
SSE3-КОМАНДИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок складання, налагодження і виконання програм з використанням команд SSE3, написаних мовою асемблера для програмування МП платформи x86.

Постановка задачі

Згідно з останньою цифрою номера студента в групі вибрати варіант свого завдання та написати на асемблері програму за допомогою SSE3-команд, ***вивести на екран варіант завдання, його вміст, результат, прізвище та e-mail*** автора програми.

Завдання

1. Написати програму дослідження команди PABS.
2. Написати програму дослідження команди PSIGN.
3. Написати програму дослідження команди PALIGNR.
4. Написати програму дослідження команди PSHUFB.
5. Написати програму дослідження команди PMULHRSW.
6. Написати програму дослідження команди PHSUB.
7. Написати програму дослідження команди PHSUBSW.
8. Написати програму дослідження команди PHADDSW.
9. Написати програму дослідження команди PMADDUBSW.
10. Написати програму дослідження команди PHADDSW.

Зміст звіту

1. Постановка задачі для конкретного варіанта завдання.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми з виведенням даних на екран монітора з використанням API-функцій та з детальним коментарем і описом роботи.
4. Print screen екрана 32-розрядного налагоджувача з виконаною програмою.
5. Короткий опис виконання програми.
6. Висновки за результатами роботи.

3.19. Лабораторна робота 19
WINDOWS-ЗАСТОСУВАННЯ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок у побудові базового застосування під Win32 з дослідженням параметрів віконних процедур.

Постановка задачі

Написати програму створення вікна Windows з файлом ресурсів. Вікно повинне містити заголовок, у якому відображається прізвище студента, номер його навчальної групи та його електронна адреса. Зміна розмірів вікна повинна супроводжуватись повідомленням. Параметри ж самого вікна повинні відповідати варіанту завдання. **Функцію MessageBox в програмі використати тільки один раз.**

Методичні вказівки

У випадку якщо треба взяти ресурс піктограми з іншого exe-файлу, то можна скористатися, наприклад, програмою PE Resource Explorer (wasm.ru).

Варіанти завдань

1. Параметри вікна:

– піктограма застосування за умовчанням;

– курсор у вигляді стандартної стрілки та малого пісочного годинника;

– вікно має заголовок і рамку з обрамленням;

– вигляд вікна функції ShowWindow: вікно сховане;

– вікно розташувати у лівому верхньому куті екрана;

– розміри вікна 300 × 500 точок.

Використовувати кнопки віконного меню в такому порядку:

– перша кнопка: “Результат”. У ній – спливаюча кнопка “Выход из программы”;

– друга кнопка – “Справка”. У ній – спливаючі кнопки “Автор программы”, “Требования к программе”, “Выход из программы”;

– третя кнопка – “Выход из программы”.

Використати в функції AnimateWindow параметр AW_SLIDE.

2. Параметри вікна:

– піктограма у вигляді білого хреста на фоні червоного кола;

– курсор у вигляді перехрестя;

– біля вікна є кнопка максимізації;

– вигляд вікна функції ShowWindow: вікно показане в його нормальних розмірах;

– вікно розташувати у правому верхньому куті екрана;

– розміри вікна 200 × 550 точок.

Використовувати кнопки віконного меню в такому порядку:

– перша кнопка – “О программе”;

– друга кнопка – “Автор”. У ній – спливаюча кнопка “Об авторе”;

– третя кнопка – “Выход из программы”.

Використати в функції AnimateWindow параметр AW_CENTER.

3. Параметри вікна:

– піктограма «і»;

– курсор у вигляді стрілки та знака питання;

– біля вікна є кнопка мінімізації;

– вигляд вікна функції ShowWindow: вікно згорнуте і показане як піктограма;

– вікно розташувати у лівому нижньому куті екрана;

– розміри вікна: 500 × 600 точок.

Використовувати кнопки віконного меню в наступному порядку:

– перша кнопка – “Справка”. У ній – спливаючі кнопки “О программе”, “Об авторе”;

– друга кнопка – “Результат”;

– третя кнопка – “Выход из программы”.

Використати в функції AnimateWindow параметр AW_HOR_ POSITIVE.

4. Параметри вікна:

– піктограма «?»;

– курсор у вигляді текстового двотавру;

– біля вікна є достатньо товста рамка;

– вигляд вікна функції ShowWindow: вікно відображається в його розмірах і позиції, встановлених безпосередньо перед значеннями розмірів і позиції;

– вікно розташувати по центру екрана;

– розміри вікна 200 × 650 точок.

Використовувати кнопки віконного меню у такому порядку:

– перша кнопка – “Справка”. У ній – спливаючі кнопки “О программе”, “Требования к программе”, “Об авторе”;

– друга кнопка – “Результат”;

– третя кнопка – “Выход из программы”.

Використати в функції AnimateWindow параметр AW_HOR_ NEGATIVE.

5. Параметри вікна:

– піктограма «!»;

– курсор у вигляді перекресленого кола;
– біля вікна є горизонтальна лінійка прокручування.
– вигляд вікна функції ShowWindow: вікно відображається в його потокових розмірах і позиції;
– вікно розташувати у лівому верхньому куті екрана;
– розміри вікна 250 × 700 точок.
Використовувати кнопки віконного меню в такому порядку:
– перша кнопка – “Справка”;
– друга кнопка – “Результат”. У ній – спливаючі кнопки “Выход из программы”, “Результат”.
Використати в функції AnimateWindow параметр AW_VER_ POSITIV.

6. Параметри вікна:
– піктограма з логотипом Windows;
– курсор у вигляді чотирикінцевої стрілки;
– біля вікна є вертикальна лінійка прокручування;
– вигляд вікна функції ShowWindow: вікно згорнуте й активізує вікно верхнього рівня в списку системи;
– вікно розташувати у правому верхньому куті екрана;
– розміри вікна 300 × 750 точок.
Використовувати кнопки віконного меню в такому порядку:
– перша кнопка – “О программе”;
– друга кнопка – “Результат”. У ній – спливаючі кнопки “Результат”, “Об авторе”, “Выход”.
Використати в функції AnimateWindow параметр AW_VER_ NEGATIVE.

7. Параметри вікна:
– піктограма у вигляді білого хреста на фоні червоного кола;
– курсор у вигляді двокінцевої стрілки, яка вказує на північний схід і південний захід;
– біля вікна є рамка, яка зазвичай буває у діалогових вікон;
– вигляд вікна функції ShowWindow: вікно згорнуте;
– вікно розташувати у лівому нижньому куті екрана;
– розміри вікна 300 × 800 точок.
Використовувати кнопки віконного меню в такому порядку:
– перша кнопка – “Сведения про автора”;
– друга кнопка – “О программе”. У ній – спливаючі кнопки “Требования к программе”, “Результат”, “Выход”
– третя кнопка: назва “Выход”.
Використати в функції AnimateWindow параметр AW_CENTER.

8. Параметри вікна:
– піктограма «і»;

– курсор у вигляді двокінцевої стрілки, яка вказує на захід і схід;
– біля вікна є тонка обмежувальна рамка;
– вигляд вікна функції ShowWindow: вікно сховане;
– вікно розташувати у правому нижньому куті екрана;
– розміри вікна 250 × 850 точок.
Використовувати кнопки віконного меню в такому порядку:
– перша кнопка – “Сведения про автора”;
– друга кнопка – “О программе”. У ній – спливаючі кнопки “Требования к программе”, “Результат”, “Выход”;
– третя кнопка: назва “Выход”.
Використати в функції AnimateWindow параметр AW_HOR_ POSITIVE.

9. Параметри вікна:
– піктограма «!»;
– курсор у вигляді вертикальної стрілки;
– створюється спливаюче (popup) вікно;
– вигляд вікна функції ShowWindow: вікно показане в його потоковому стані;
– вікно розташувати по центру екрана;
– розміри вікна500 × 900 точок.
Використовувати кнопки віконного меню в такому порядку:
– перша кнопка – “Справка”. У ній – спливаючі кнопки “Требования к программе”, “ Сведения про автора ”;
– друга кнопка – “О программе”. У ній – спливаючі кнопки “Требования к программе”, “Результат”;
– третя кнопка: назва “Выход”.
Використати в функції AnimateWindow параметр AW_VER_ NEGATIVE.

10. Параметри вікна:
– піктограма у вигляді білого хреста на фоні червоного кола;
– курсор у вигляді пісочного годинника;
– створюється спливаюче (popup) вікно;
– вигляд вікна функції ShowWindow: вікно сховане;
– вікно розташувати у правому нижньому куті екрана;
– розміри вікна 500 × 950 точок.
Використовувати кнопки віконного меню в наступному порядку:
– перша кнопка – “Результат”. У ній – спливаючі кнопки “Результат”, “Выход из программы”;
– друга кнопка: назва “Справка”. У ній – спливаючі кнопки “Автор программы”, “Требования к программе”, “Выход из программы”;
– третя кнопка – “Выход из программы”.
Використати в функції AnimateWindow параметр AW_BLEND.

Приклад. Розглянемо програму створення вікна з пунктами меню та з горизонтальною смугою прокрутки. Файл ресурсів наведено в лістингу 3.19.1.

Лістинг 3.19.1. Файл ресурсів:

```
#define IDM_HELLO   1
#define IDM_EXIT    2
#define IDM_ABOUT   3
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL  "mk_icon.ico"
FirstMenu MENU {
        POPUP "Программа"{
      MENUITEM "Задание",IDM_HELLO
      MENUITEM SEPARATOR
      MENUITEM "Выйти",IDM_EXIT
                              }
        POPUP "Информация"{
     MENUITEM "About",IDM_ABOUT
                                 }
              }
```

У програмі з лістингу 3.19.2 використовуються дві піктограми: одна піктограма – стандартна з параметром IDI_WARNING, а друга – піктограма, яка відображається в панелі завдань екрана монітора. Горизонтальна смуга прокручування визначається у директиві створення вікна

```
invoke  CreateWindowEx,NULL, ADDR ClassName,\
   ADDR AppName, WS_OVERLAPPEDWINDOW or WS_HSCROLL,\
     0, 0,250,450,0,0,  hInstance,NULL
```

Лістинг 3.19.2:

```
.386                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\macros\macros.asm
 uselib user32,kernel32,gdi32
  IDM_HELLO  equ 1
  IDM_EXIT   equ 2
  IDM_ABOUT  equ 3
  IDI_ICON equ 22
.data                   ; директива визначення даних
      ClassName db "SimpleWinClass",0
      AppName  db "Окно с параметрами",0
      MenuName db "FirstMenu",0
```

```
      Hello_string db "Создать окно с такими параметрами:",10,13
str1 db " - пиктограмма ‘!’;",10,13
str2 db " - курсор в виде перечёркнутого круга;",10,13
str3 db " - у окна есть горизонтальная полоса прокрутки;",10,13
str4 db " - вид окна функции ShowWindow: окно отображается в его
текущих размерах и позиции;",10,13
str5 db " - окно розмещено в левом верхнем углу;",10,13
str6 db " - размеры окна: 250 на 450 точек.",0
About_string db "Выполнил:   ",10,13
str7 db " e-mail: ",10,13,0
      wc  WNDCLASSEX   <>
      msg MSG      <>
      hwnd HWND    ?
      hInstance HINSTANCE ?
.code                              ; директива початку сегмента команд
start:                             ; мітка початку програми з ім’ям start
      invoke  GetModuleHandle, NULL    ; отримання дескриптора програми
      mov   hInstance,eax                ; збереження дескриптора програми
      mov     wc.cbSize,SIZEOF WNDCLASSEX   ; кількість байтів структури
mov  wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
      mov     wc.lpfnWndProc, OFFSET WndProc
      mov     wc.cbClsExtra,NULL
      mov     wc.cbWndExtra,NULL
      push   hInstance             ; збереження в стека  дескриптора програми
      pop     wc.hInstance         ; повернення дескриптора в поле структури
      mov     wc.hbrBackground,COLOR_WINDOW+1  ; колір вікна
      mov     wc.lpszMenuName,OFFSET MenuName  ; ім’я ресурсу меню
      mov     wc.lpszClassName,OFFSET ClassName  ; ім’я класу
invoke  LoadIcon,NULL,IDI_WARNING    ; піктограми у вигляді оклику
      mov     wc.hIcon,eax               ; дескриптор «великої» піктограми
      mov     wc.hIconSm,eax             ; дескриптор маленького віконця
invoke  LoadCursor,NULL,IDC_NO ; курсор –  перекреслений кружок
      mov     wc.hCursor,eax
      invoke  RegisterClassEx, addr wc   ; функція реєстрації класу вікна
      invoke  CreateWindowEx,NULL, ADDR ClassName,\
         ADDR AppName, WS_OVERLAPPEDWINDOW or WS_HSCROLL,\
          0, 0,\           ; горизонтальні та вертикальні координати вікна
            250,450,NULL,NULL,\   ; ширина та висота вікна
            hInstance,NULL
      mov     hwnd,eax
invoke  ShowWindow, hwnd,SW_SHOW     ; видимість вікна
      invoke SetScrollRange, hwnd,SB_HORZ,0,10,TRUE
      .WHILE TRUE                                    ; поки істинне, то
        invoke  GetMessage, ADDR msg,NULL,0,0  ; читання повідомлення
          or eax, eax          ; формування ознак
          jz Quit              ; перейти на мітку Quit, якщо eax = 0
```

```
      invoke DispatchMessage, ADDR msg ; відправка до WndProc
       .ENDW              ; закінчення циклу оброблення повідомлень
       Quit:
       mov     eax,msg.wParam
invoke ExitProcess, eax ; повернення управління та вивільнення ресурсів

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_DESTROY     ; якщо є повідомлення про знищення вікна
        invoke PostQuitMessage,NULL  ; передача повідомлення WM_Quit
.ELSEIF uMsg==WM_COMMAND       ; якщо є повідомлення від меню
         mov eax,wParam
        .IF ax==IDM_HELLO          ; якщо є повідомлення IDM_HELLO
invoke  MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
        .ELSEIF ax==IDM_ABOUT      ; якщо є повідомлення IDM_ABOUT
invoke  MessageBox,NULL,ADDR About_string, OFFSET AppName, MB_OK
        .ELSE
           invoke DestroyWindow,hWnd          ; знищення вікна
        .ENDIF
.ELSE
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; стандартна обробка
        ret    ; повернення з процедури
.ENDIF
      xor    eax,eax
      ret             ; повернення з процедури
      WndProc endp     ; закінчення процедури WndProc
      end start       ; закінчення програми
```

Результати роботи програми наведені на рис. 3.19.1.

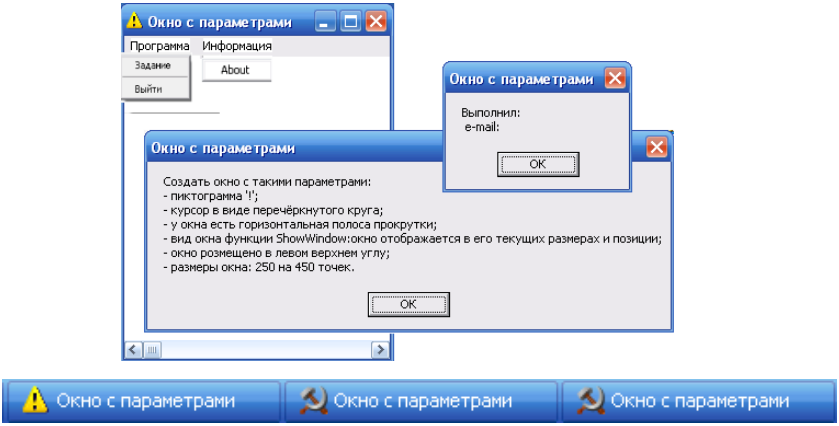


Рис. 3.19.1. Результати роботи програми

Зміна прозорості вікна розглядається у лістингу 3.19.3.

Лістинг 3.19.3. Програма зі зміною прозорості вікна:

```
      .686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc   ; файли структур, констант …
include \masm32\macros\macros.asm
uselib kernel32, user32;
      WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
.data?
      hInstance HINSTANCE ?
      CommandLine LPSTR ?
      Value dd ?
.data
      ClassName db "MASM Builder",0
      Caption db "Исследование функции AnimateWindow",0
.code
 start:
      invoke GetModuleHandle, NULL ; отримання дескриптора програми
      mov    hInstance,eax         ; збереження дескриптора програми
      invoke WinMain,hInstance,NULL,CommandLine,SW_SHOWDEFAULT
        invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
                                                    CmdShow:DWORD
      LOCAL wc :WNDCLASSEX
      LOCAL msg :MSG
      LOCAL hWnd :HWND
mov wc.cbSize,SIZEOF WNDCLASSEX       ; кількість байтів структури
mov wc.style,CS_HREDRAW or CS_VREDRAW ; стиль
(CS_BYTEALIGNCLIENT)
mov wc.lpfnWndProc,offset WndProc     ; адреса процедури WndProc
mov wc.cbClsExtra,NULL          ; кількість байтів для структури
mov wc.cbWndExtra,NULL          ; кількість байтів для структури
      push hInst                ; збереження в стеці дескриптора програми
      pop wc.hInstance          ; повернення дескриптора в поле структури
      mov wc.hbrBackground, COLOR_BTNFACE+1  ; сірий колір вікна
      mov wc.lpszClassName,OFFSET ClassName  ; ім'я класу
      invoke LoadIcon,NULL,IDI_APPLICATION ; без піктограми
      mov wc.hIcon,eax          ; дескриптор піктограми
      mov wc.hIconSm,eax        ; дескриптор маленького віконця
      invoke LoadCursor,NULL,IDC_ARROW  ; ресурс курсора
      mov wc.hCursor,eax
invoke RegisterClassEx,addr wc          ;  реєстрація  класу вікна
```

```
invoke CreateWindowEx,0,ADDR ClassName,ADDR Caption,\
   WS_OVERLAPPEDWINDOW,350,80,400,200,0,0,hInst,0
      mov hWnd,eax
      invoke ShowWindow,hWnd,SW_SHOWNORMAL
      invoke UpdateWindow,hWnd
.WHILE TRUE
        invoke GetMessage,ADDR msg,0,0,0 ; читання повідомлення
.BREAK .IF (!eax)
        invoke TranslateMessage,ADDR msg
        invoke DispatchMessage,ADDR msg  ; відправка на обслуговування
.ENDW
      mov eax,msg.wParam
      ret           ; повернення з процедури
      WinMain endp     ; закінчення процедури з ім'ям WinMain

      WndProc proc
hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
      .IF uMsg == WM_DESTROY
      invoke PostQuitMessage,NULL
      .ELSEIF uMsg == WM_CREATE ;
invoke AnimateWindow,hWnd,1000,AW_BLEND+AW_ACTIVATE
invoke GetWindowLongA,hWnd,GWL_EXSTYLE    ; витягання інф. із вікна
      or eax,WS_EX_LAYERED
invoke SetWindowLongA,hWnd,GWL_EXSTYLE,eax ; атрибути вікна
mov Value,150 ;  прозорість вікна: 0 – повністю прозоре і 255 – непрозоре
invoke SetLayeredWindowAttributes,\ ; встановлює  прозорість
       hWnd,Value,Value,LMA_COLORKEY + LMA_ALPHA
.ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
.ENDIF                  ; закінчення логічної структури
      xor eax,eax
      ret               ; повернення з процедури
      WndProc endp      ; закінчення процедури WndProc
      end start         ; закінчення програми з ім'ям start
```

3.20. Лабораторна робота 20
BMP-ФАЙЛИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок у застосуванні BMP-файлів з дослідженням параметрів віконних процедур.

Постановка задачі

Написати програму створення основного вікна та додаткового з викликом BMP-файлу, в якості якого використати свою фотографію. Крім того, створити кнопку (кнопки) з BMP-файлу.

Методичні рекомендації

Для створення кнопки використовують функцію BmpButton. Ця функція створює кнопку з двох BMP-файлів. У початковому стані відображається один BMP рисунок, а при натисканні на цю кнопку – інший. Ця функція за координатами x та y завантажує BMP рисунки та ідентифікує їх своїми ID номерами. При натисненні на кнопку завантажується друге зображення і посилається повідомлення WM_COMMAND з параметром ID цього контрола. За цим ID контролу в wParam проводяться дії з обробки цього натиснення. Функція BmpButton розташована в бібліотеці **masm32lib**, а описана в masm32\help\masmlib.

Функція **BmpButton** має синтаксис:

```
hParent:DWORD, \ ; хендл вікна, в якому обробляється WM_COMMAND
  topX:DWORD, \ ; x-координата кнопки
    topY:DWORD, \ ; y-координата кнопки
     rnum1:DWORD, \ ; ресурс ID для UP точкового  рисунка bmp1
     rnum2:DWORD, \ ; ресурс ID для DOWN точкового рисунка  bmp2
 ID:DWORD ; ідентифікаційний номер, який приписується контролю
```

Наприклад:

```
Uselib masm32;
…
  win32asm dd 0 ; початкове встановлення тригеру натискання на кнопку
…
.IF uMsg==WM_INITDIALOG ;
invoke SendMessage,hWin,WM_SETICON,1,FUNC(LoadIcon, 0, IDI_ASTERISK)
 invoke BmpButton, hWin, 32, 40, 203, 204, 2;
```

```
.ELSEIF uMsg==WM_COMMAND ;
…
.ELSEIF wParam==2 ;
 inc win32asm
   .if win32asm == 1
 invoke MessageBox,0,addr szInf0,addr szTitle0,MB_ICONINFORMATION
  mov  win32asm,0
   .endif
```

Фрагмент файлу ресурсів може бути таким:

```
203 BITMAP MOVEABLE PURE LOADONCALL DISCARDABLE "up_11.BMP"
204 BITMAP MOVEABLE PURE LOADONCALL DISCARDABLE "up_22.BMP"
```

Графічний файл із зовнішнім виглядом отримано за допомогою програми DeKnop, зовнішній вигляд інтерфейсу якого наведено на рис. 3.20.1.

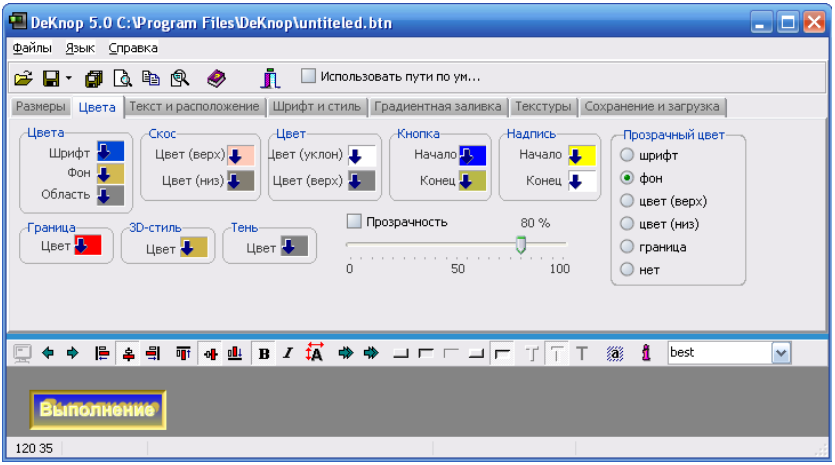


Рис. 3.20.1. Вигляд інтерфейсу програми DeKnop для створення кнопки з BMP-файлом

3.21. Лабораторна робота 21
ЕЛЕМЕНТИ ІНТЕРФЕЙСУ ВІКНА

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок у застосуванні BMP-файлів з дослідженням параметрів віконних процедур.

Постановка задачі

Написати програму створення основного вікна та додаткового з викликом BMP-файлу, в якості якого використати свою фотографію. Крім того, створити кнопки всіх типів згідно з методичними рекомендаціями та вимогами викладача.

Методичні рекомендації

Створення інтерфейсу діалогової програми за допомогою візуального редактора ресурсів ResEd

Приклад 21.1. Створення елементарної діалогової програми з двома кнопками: при натисканні на одну кнопку виводиться повідомлення про програму, а при натисканні на іншу – вихід з програми.

Діалогове вікно простіше створювати візуально. Для цього спочатку необхідно вибрати редактор ресурсів. Їх досить багато: від самих простих (ResEd, RadAsmIDE, MasmEd1 та ін.) до професійних (наприклад, Visual Studio).

Скористаємося візуальним редактором ресурсів ResEd (Resource editor, http://radasm.cherrytree.at). Запускаємо його та отримуємо видимий інтерфейс, який наведено на рис. 3.21.1.

Спочатку створюємо новий файл ресурсів (File – New Project та вводимо ім'я). У правій верхній панелі з'являється значок папки та ім'я файлу.

Натискаємо на неї правою кнопкою миші та обираємо «Add Dialog». У робочій області програми ресурсів з'являється нове поле вікна. Тепер можемо візуально спроектувати інтерфейс вікна і змінити його настройки.

При обранні конкретних ресурсів треба уявляти те, що потрібно відобразити. Розміри вікна можна змінювати візуально, якщо підвести курсор до міток вікна. Для зміни надпису вікна обираємо в правій нижній таблиці рядок таблиці Caption, а праворуч – змінюємо й саму назву.

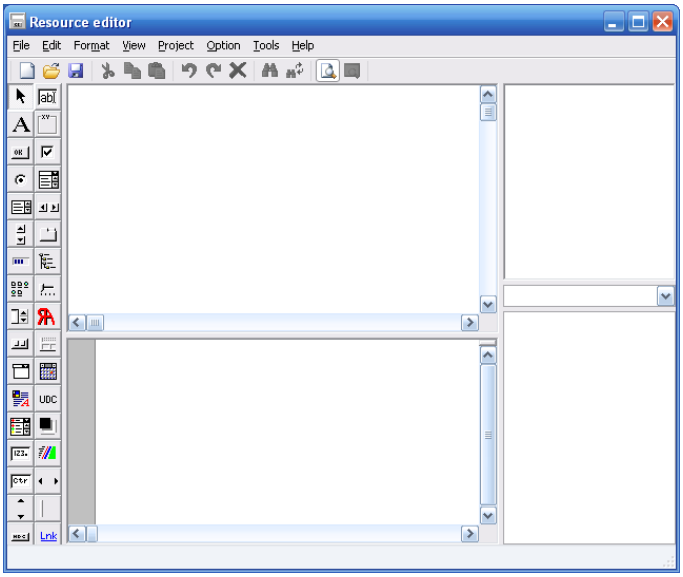


Рис. 3.21.1. Інтерфейс редактора ресурсів ResEd

Для створення кнопки необхідно обрати кнопку  з надписом OK. Поки кнопка обрана (на ній знаходиться фокус вікна) можна змінити (для зручності): назву кнопки – в рядку Caption, а назву текстового повідомлення – в рядку з позначенням (Name).

Наприклад, вибрати курсором миші рядок з позначенням (Name) та в правому його полі змінити назву повідомлення на потрібну, наприклад, TEST_BTN. Числовий ідентифікатор повідомлення знаходиться в полі (ID).

Друга кнопка створюється аналогічно.

Таким чином, діалогове вікно з двома кнопками наведено на рис. 3.21.2.

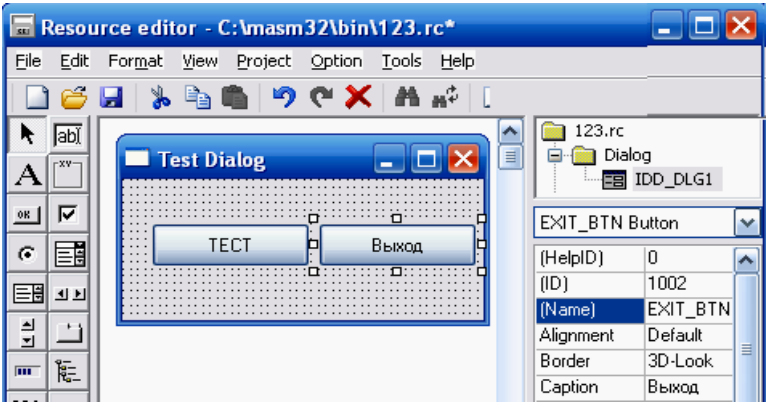


Рис. 3.21.2. Створення діалогового вікна

Після створення дизайну інтерфейсу необхідно добавити файл констант. Для цього, по-перше, натискаємо правою кнопкою миші на значок папки, вибираємо «Include file», по-друге, розширюємо основне поле відображення діалогового вікна (щоб були видні додаткові кнопки «Add» та «Delete»), по-третє, натискаємо кнопку «Add» та вводимо шлях знаходження файлу констант RESOURCE.H, наприклад, C:\masm32\include\ RESOURCE.H, який необхідний для компіляції програми. Відображення кнопки додавання елементів файлу ресурсів наведено на рис. 3.21.3.

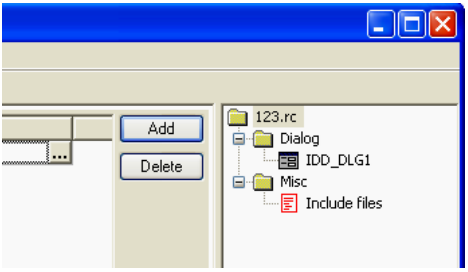


Рис. 3.21.3. Відображення кнопки додавання елементів файлу ресурсів

У створений файл доцільно вставити сучасний XP-стиль. Для цього необхідно додати до файлу ресурсів XP Manifest. Додається він аналогічно попереднім пунктам (Add XP Manifest).

У результаті, файл ресурсів повинен бути, наприклад, таким, як наведено в лістингу 3.21.1.

Лістинг 3.21.1. Файл ресурсів:

```
#define MANIFEST 24
#define TEST_DIALOG 1000
#define TEST_BTN 1001
#define EXIT_BTN 1002
#define IDR_XPMANIFEST1 1
#include "C:/masm32/include/RESOURCE.H"
TEST_DIALOG DIALOGEX 300,300,134,51
CAPTION "Test Dialog"
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_CAPTION|WS_SYSMENU
BEGIN
CONTROL "Тест",TEST_BTN,"Button", \
  WS_CHILD|WS_VISIBLE|WS_TABSTOP,6,18,54,13
CONTROL "Выход",EXIT_BTN,"Button", \
  WS_CHILD|WS_VISIBLE|WS_TABSTOP, 72,18,54,13
END
IDR_XPMANIFEST1 MANIFEST "xpmanifest.xml"
```

У лістингу 3.21.2 наведено основну програму з використанням кнопок.

Лістинг 3.21.2. Файл програми:

```
386                         ; директива визначення типу мікропроцесора
.model  flat, stdcall  ; задання лінійної моделі пам'яті та угоди ОС Windows
  option casemap:none              ; відмінність малих та великих  літер
    include \masm32\include\windows.inc ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib user32,kernel32, comctl32

WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD ; прототип проц.
    TEST_DIALOG = 1000 ; ідентифікатор діалогового вікна
    TEST_BTN = 1001      ; ідентифікатор кнопки «Тест»
    EXIT_BTN = 1002      ; ідентифікатор кнопки «Выход»
.data?                          ; директива невизначених даних
     hInstance dd ?
     hWnd dd ?
     icce InitCOMMONCONTROLSEX <> ; інф.  для управ. вікном
.code                                   ; директива початку сегмента команд
start:                                   ; мітка початку програми з ім'ям start
mov icce.dwSize, SIZEOF InitCommonControlsEX ; розмір структури
; mov icce.dwICC, ICC_DATE_CLASSES or \ ; клас органів управл. даними
```

```
; ICC_COOL_CLASSES ; додаткова інструментальна панель типу Rebar

invoke InitCommonControlsEx, offset icce ; реєстрація класів
                                          ; стандартного органу управління
      invoke GetModuleHandle, NULL ; отримання дескриптора програми
         mov hInstance, eax          ; збереження дескриптора програми
invoke DialogBoxParam, hInstance,\; дескриптор екземпляра програми
  TEST_DIALOG,\  ; ідентифікація шаблона блока діалогу
    0,\       ; дескриптор вікна власника
      offset WndProc, 0 ; покажчик на процедуру діал. вікна та LPARAM
invoke ExitProcess,eax ; виклик процедури повернення управління ОС…

WndProc proc hWin :DWORD, uMsg :DWORD, wParam :DWORD, lParam :DWORD
      ;  switch uMsg
      ;    case WM_INITDIALOG
; invoke SendMessage, hWin, WM_SETICON, 1, FUNC(LoadIcon, 0,
IDI_ASTERISK)
      ; case WM_COMMAND
      ;      switch wParam
      ;        case TEST_BTN
      ; invoke MessageBox, hWin, chr$("Диалоговое окно"), chr$("Test"), 0
      ;        case EXIT_BTN
      ;          jmp exit_program
      ;      endsw
      ; case WM_CLOSE
      ;      exit_program:
      ;      invoke EndDialog, hWin, 0
      ; endsw
      ;  xor eax,eax
      ; ret
      ; WndProc ENDP
      ; end start

      .if uMsg==WM_INITDIALOG ;
 invoke SendMessage,\  ; відправлення повідомлення вікну
   hWnd,\          ; хендл вікна
    WM_SETICON, \ ; повідомлення про виведення іконки
     1,                    ; wParam – яке завгодно
  FUNC(LoadIcon, 0, IDI_ASTERISK) ; ф-я для відображення іконки
      .elseif uMsg==WM_COMMAND ; якщо є повідомлення від меню
        .if wParam==TEST_BTN ;
      invoke MessageBox, hWin, chr$("Диалоговое окно"), chr$("Test"),\
       MB_ICONINFORMATION+180000h
        .elseif wParam==EXIT_BTN ;
         jmp exit_program
        .else
```

```
        .endif
    .elseif uMsg==WM_CLOSE ; якщо є повідомлення про закриття
exit_program:
    invoke EndDialog, hWin, 0
    .else                       ; інакше
    mov eax, FALSE
        ret                                 ; повернення з процедури
    .endif                  ; закінчення логічної структури .IF – .ELSEIF
    xor    eax,eax                  ; підготування до завершення
        ret                     ; повернення з процедури
    WndProc endp                    ; закінчення процедури
    end start                       ; закінчення програми з ім'ям start
```

У діалоговій програмі створюються ідентифікатори не тільки кнопок, а, насамперед, самого вікна. Цей ідентифікатор (TEST_DIALOG = 1000) створюється першим.

Вигляд створених вікон програми наведено на рис. 3.21.4.

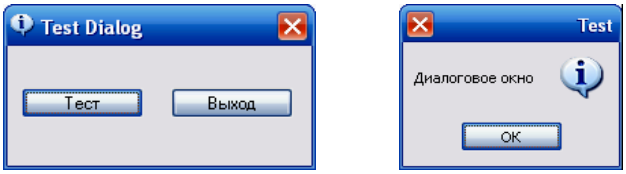


Рис. 3.21.4. Вигляд створених вікон

У програмі можна використовувати процедуру WndProc proc як у звичайному вигляді, так й у вигляді макросів switch та case при підключенні бібліотеки **macros**.asm.

Структура InitCommonControLSeX використовується для того, щоб завантажувати класи стандартних органів управління з бібліотеки, що динамічно підключається (DLL). Структура використовується з відповідною функцією InitCommonControlsEx.

```
Синтаксис: typedef struct tagINITCOMMONCONTROLSEX {
    DWORD dwSize;
    DWORD dwICC;
} INITCOMMONCONTROLSEX, *LPINITCOMMONCONTROLSEX;
```

Члени структури:

dwSize – розмір структури, в байтах;

dwICC – набір бітових прапорців, які указують які класи стандартного органу управління будуть завантажені з бібліотеки, що

динамічно підключається (DLL). Це значення може бути комбінацією наведених нижче прапорців:

ICC_ANIMATE_CLASS – завантажує клас органів управління анімацією;

ICC_BAR_CLASSES – завантажує класи органів управління: інструментальна панель (toolbar), рядок стану (status bar), панель завдань (trackbar), підказка (ToolTip);

ICC_COOL_CLASSES – завантажує клас органів управління – додаткова інструментальна панель типу Rebar;

ICC_DATE_CLASSES – завантажує клас органів управління даними і пристрою вибору часу;

ICC_HOTKEY_CLASS – завантажує клас органів управління "швидкими" клавішами;

ICC_INTERNET_CLASSES – завантажує клас вікна з IP адресою;

ICC_LINK_CLASS – завантажує клас органів управління гіперпосиланнями;

ICC_LISTVIEW_CLASSES – завантажує клас органів управління колекціями значків з надписами (list-view) і заголовками (header);

ICC_NATIVEFNTCTL_CLASS – завантажує клас органів управління власним шрифтом (native font);

ICC_PAGESCROLLER_CLASS – завантажує клас органів управління пейджером (pager);

ICC_PROGRESS_CLASS – завантажує клас органів управління – вікно ходу операції (progress bar);

ICC_STANDARD_CLASSES – завантажує один із вбудованих класів органу управління в User32. Призначені для користувача органи управління включають кнопку, поле редагування (edit), статичний елемент (static), вікно із списком (listbox), комбіноване вікно (combobox) і смуги прокручування (scrollbar);

ICC_TAB_CLASSES – завантажує клас органів управління вкладками (tab) і підказками (ToolTip);

ICC_TREEVIEW_CLASSES – завантажує клас органів управління деревоподібним списком (tree-view) і підказками (ToolTip);

ICC_UPDOWN_CLASS – завантажує клас органів управління збільшення/зменшення на 1 значення в полі введення (up-down control);

ICC_USEREX_CLASSES – завантажує клас покращуваного комбінованого вікна (ComboBoxEx);

ICC_WIN95_CLASSES – завантажує класи всіх перелічених вище органів управління (animate control, header, hot key, list-view, progress bar, status bar, tab, ToolTip, toolbar, trackbar, tree-view и up-down).

Приклад 3.21.2. Створення файлу ресурсів з кнопкою меню та додатковою кнопкою.

Для виконання таких вимог необхідно дотримуватись таких пунктів:

1. Створення нового проекту файлу ресурсів. Обираємо File – New Project та вводимо ім'я, наприклад Test2.rc.

2. Створення діалогового вікна. Для цього підводимо курсор миші в праве верхнє вікно до назви проекту (Test2.rc), натискаємо на праву кнопку миші та вибираємо «Add Dialog». Змінюємо назву в полі Caption та пишемо: "Диалоговое окно с меню".

3. Створення кнопки. Для створення однієї кнопки необхідно обрати кнопку  з надписом ОК та перетягнути її на поле діалогового вікна. Поки кнопка обрана (на ній знаходиться фокус вікна) можна змінити її назву в нижньому правому полі рядка Caption, а також тип повідомлення в полі (Name) (рис. 3.21.5).

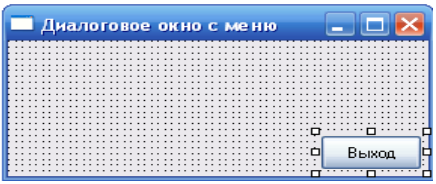


Рис. 3.21.5. Створення кнопки діалогового вікна

4. Створення меню. Для цього підводимо курсор миші до правого верхнього вікна з назвою проекту (Test2.rc), натискаємо на праву кнопку миші та вибираємо «Add Menu».

Для створення основної кнопки меню необхідно вибрати параметри (рис. 3.21.6) кнопки меню.

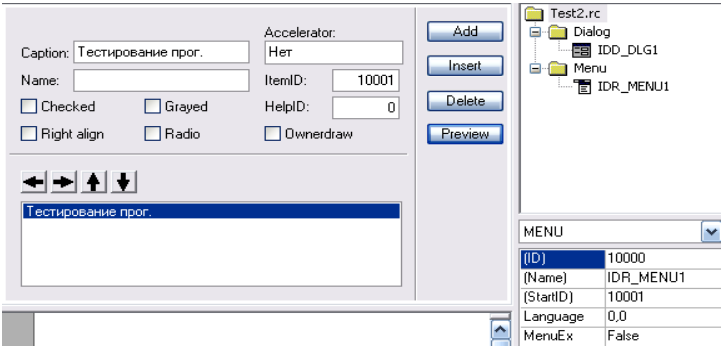


Рис. 3.21.6. Параметри основної кнопки меню діалогового вікна

Для того щоб подивитися на зроблені зміни, необхідно натиснути на кнопку «Preview».

Додаткові основні кнопки меню створюються через натискання кнопки Add.

5. Створення спливаючих кнопок. Для цього необхідно поставити курсор миші нижче назви основної кнопки (рис. 3.21.7) та обрати параметри спливаючої кнопки меню. Обов'язково натиснути кнопку  (стрілка), яка свідчить про те, що створюються підпункт основної кнопки меню.

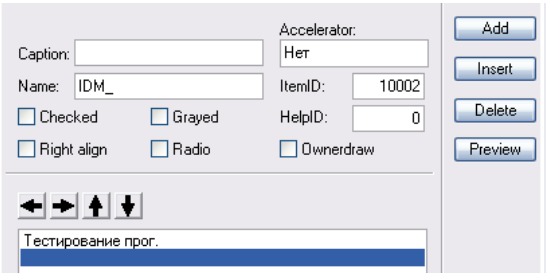


Рис. 3.21.7. Створення спливаючої кнопки меню діалогового вікна

Аналогічним чином створюються й інші кнопки.

При натисканні на кнопку «Preview» відобразиться зовнішній вигляд діалогового вікна (рис. 3.21.8).

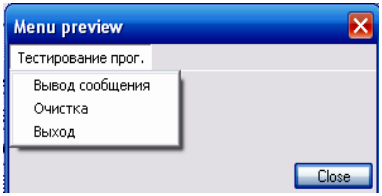


Рис. 3.21.8. Зовнішній вигляд діалогового вікна

6. Добавлення файлу констант. Для цього натискаємо правою кнопкою миші на значок папки, вибираємо «Include file», «Add» і вводимо шлях, наприклад, C:\masm32\include\RESOURCE.H.

7. Вставити сучасний XP-стиль. Для цього необхідно додати до файлу ресурсів XP Manifest. Додається він аналогічно попереднім пунктам (Add XP Manifest).

Розглянемо програму, яка містить **контрол** (поле для введення-виведення) та кнопку. У верхній частині вікна створена іконка. Меню вікна містить два пункти. При натисканні на пункт "Відображення тексту" виводиться рядок тексту з комірки з ім'ям TestString. Якщо набрати текст у контролі та вибрати пункт "Введення тексту", то цей текст відобразиться у функції MessageBox.

Лістинг 3.21.3. Файл програми:

```
    .386                ; директива визначення типу мікропроцесора
.model flat,stdcall  ; задання лінійної моделі пам'яті та угоди ОС Windows
    option casemap:none      ; відмінність малих та великих літер
    WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
  include \masm32\include\windows.inc ; файли структур, констант …
  include \masm32\macros\macros.asm
  uselib user32,kernel32
.data                              ; директива визначення даних
    ClassName db "SimpleWinClass",0
    AppName  db "Дочірнє вікно",0
    MenuName db "FirstMenu",0
    ButtonClassName db "button",0
    ButtonText db "Кнопка введення",0
    EditClassName db "edit",0
    TestString db "Введення повідомлення у вікно",0
.data?                             ; директива невизначених даних
    hInstance HINSTANCE ?
    CommandLine LPSTR ?
    hwndButton HWND ?
    hwndEdit HWND ?
    buffer db 512 dup(?)
.const
    ButtonID equ 1
    EditID    equ 2
    IDM_HELLO equ 1
    IDM_GETTEXT equ 3
    IDM_EXIT equ 4
    IDI_ICON  equ 22 ;
.code                              ; директива початку сегмента команд
 start:                                ; мітка початку програми
      invoke GetModuleHandle, NULL ; отримання дескриптора програми
        mov   hInstance,eax          ; збереження дескриптора програми
        invoke GetCommandLine
```

```
        mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
invoke ExitProcess,eax ; повернення управління та вивільнення ресурсів

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,\
                                                    CmdShow:DWORD
    LOCAL wc:WNDCLASSEX             ; резервування стека під структуру
    LOCAL msg:MSG                   ; резервування стека під структуру MSG
    LOCAL hwnd:HWND             ;  резервування стека під хендл програми
    mov   wc.cbSize,SIZEOF WNDCLASSEX      ; кількість байтів структури
mov  wc.style, CS_HREDRAW or CS_VREDRAW  ; стиль та поведінка вікна
    mov  wc.lpfnWndProc, OFFSET WndProc ; адреса процедури WndProc
    mov   wc.cbClsExtra,NULL                ; кількість байтів для структури
    mov   wc.cbWndExtra,NULL  ; кількість байтів для додаткових структур
    push  hInst         ; збереження в стека  дескриптора програми
    pop   wc.hInstance          ; повернення дескриптора в поле структури
    mov   wc.hbrBackground,COLOR_BTNFACE+1        ; колір вікна
    mov   wc.lpszMenuName,OFFSET MenuName         ; ім'я ресурсу меню
    mov   wc.lpszClassName,OFFSET ClassName       ; ім'я класу
invoke  LoadIcon,hInstance, IDI_ICON ; завантаження ресурсу піктограми
    mov   wc.hIcon,eax                  ; дескриптор «великої» піктограми
    mov   wc.hIconSm,eax                ; дескриптор маленького віконця
invoke LoadCursor,NULL,IDC_ARROW  ; ресурс курсора
        mov   wc.hCursor,eax
invoke RegisterClassEx, addr wc     ; функція реєстрації класу вікна
invoke CreateWindowEx, \            ; функція створення вікна за зразком
 WS_EX_CLIENTEDGE,\ ; стиль – вікно приймає та передає повідомлення
 ADDR ClassName, \       ; адреса імені класу
  ADDR AppName,\                    ; адреса імені вікна
  WS_OVERLAPPEDWINDOW, \            ; базовий стиль
CW_USEDEFAULT, CW_USEDEFAULT,\ ; гориз. та верт. координати вікна
   300,200,\                        ; висота та ширина вікна
    NULL,NULL,hInst,NULL
        mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL   ; видимість вікна
    invoke UpdateWindow, hwnd
        .WHILE TRUE
    invoke GetMessage, ADDR msg,NULL,0,0
          .BREAK .IF (!eax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage, ADDR msg ; відправка на обслуговування
        .ENDW
        mov     eax,msg.wParam
        ret                                     ; повернення з процедури
    WinMain endp                    ; закінчення процедури з ім'ям WinMain
```

```
WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM
.IF uMsg==WM_DESTROY ; якщо є повідомлення про знищення вікна
invoke PostQuitMessage,NULL ; передача повідомлення про знищення
.ELSEIF uMsg==WM_CREATE ; якщо є повідомлення про створення вікна
invoke CreateWindowEx,WS_EX_CLIENTEDGE, ADDR EditClassName,\
 NULL, WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT or\
  ES_AUTOHSCROLL,\
   30,20,230,25,hWnd,EditID,hInstance,NULL
          mov  hwndEdit,eax
          invoke SetFocus, hwndEdit
invoke CreateWindowEx,NULL, ADDR ButtonClassName,\
 ADDR ButtonText, WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
  75,70,150,25,hWnd,ButtonID,hInstance,NULL
                  mov  hwndButton,eax
.ELSEIF uMsg==WM_COMMAND ; обробка повідомлень від меню
                  mov eax,wParam
          .IF lParam==0
                  .IF ax==IDM_HELLO
invoke SetWindowText,hwndEdit,ADDR TestString          ; встановлює текст
                   .ELSEIF  ax==IDM_GETTEXT
invoke GetWindowText,hwndEdit,ADDR buffer,512 ; тексту в контрол
invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
                   .ELSE
invoke DestroyWindow,hWnd ; знищення вікна та діалогів
                   .ENDIF
          .ELSE
                  .IF ax==ButtonID
                          shr eax,16
                    .IF ax==BN_CLICKED ; код при відпусканні кл. миші
      invoke SendMessage,\          ; функція відсилання повідомлення вікну
       hWnd,\                       ; хендл вікна
        WM_COMMAND,\                ; тип повідомлення
         IDM_GETTEXT,\              ; wParam
         0                          ; lParam
                    .ENDIF
                  .ENDIF
          .ENDIF
          .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam    ;  обробка та
                                ; відправка повідомлення до функції WndProc
                  ret
          .ENDIF
          xor    eax,eax                    ; підготування до закінчення
          ret                               ; повернення з процедури
      WndProc endp                           ; закінчення процедури WndProc
```

```
end start                                   ; закінчення програми з ім'ям start
```

У функції CreateWindowEx задано стилі, які для кнопок починаються з префікса "BS_", а контролі – з "ES_". Дочірній елемент управління не має хендла, тому передається ідентифікатор ID.

Меню вікна й контрол відсилають повідомлення WM_COMMAND. Для їх відрізнення використовуються параметри цього повідомлення. Так, ID меню має wParam := 0 та lParam := 0. А ID контрола має wParam := код повідомлення та lParam := хендл батьківського вікна. **Тому завжди перевіряють lParam.** Якщо lParam := 0, то повідомлення від меню.

Копіювання рядка тексту в контрол виконує функція GetWindowText.

Обробка натискання на кнопку виконує фрагмент коду:

```
            .IF ax==ButtonID
               shr eax,16
               .IF ax==BN_CLICKED
invoke SendMessage,\ ; функція відсилання повідомлення вікну
 hWnd,\                         ; хендл вікна
  WM_COMMAND,\                  ; тип повідомлення
   IDM_GETTEXT,\                ; wParam
   0                            ; lParam
               .ENDIF
            .ENDIF
```

У цьому фрагменті спочатку перевіряється молодше слово wParam, щоб переконатися, що ID контрола належить кнопці. Якщо це так, то перевіряється старше слово wParam, щоб переконатися, що був посланий код повідомлення BN_CLICKED, тобто кнопка була натиснута.

Функція SendMessage посилає повідомлення вікну з параметрами wParam та lParam.

Лістинг 3.21.4. Файл ресурсів:

```
#define IDM_HELLO 1
#define IDM_GETTEXT 3
#define IDM_EXIT 4
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL  "mk_icon.ico"
FirstMenu MENU
{
 POPUP "&Діагностування вікна"
      {
       MENUITEM "Відображення тексту",IDM_HELLO
       MENUITEM "Введення тексту", IDM_GETTEXT
       MENUITEM SEPARATOR
```

```
        MENUITEM "E&xit",IDM_EXIT
      }
}
```

Результатом програми є дочірнє вікно, яке наведено на рис. 3.21.9.

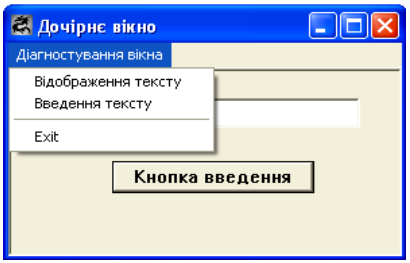


Рис. 3.21.9. Дочірнє вікно програми з лістингу 3.21.4

Лістинг 3.21.5. Командний bat-файл:

```
ml /c /coff "Dt1.asm"
rc "Dt1.rc"
link /SUBSYSTEM:windows "Dt1.obj"  "Dt1.res"
pause
start Dt1.exe
```

Використання кнопки checkbox

Checkbox з погляду програмування є кнопкою. Як стиль у функції CreateWindows треба вказати BS_AUTOCHECKBOX (у цьому випадку відмітка в checkbox'е автоматично ставитиметься / забиратися) або BS_CHECKBOX (у цьому випадку відмітка сама ставитися не буде – і це буде завдання програміста).

Приклад створення checkbox у функції WinMain після виклику CreateWindow для головного вікна може бути таким:

```
invoke CreateWindowEx,NULL, addr button_class, addr button_title ,\
        WS_CHILD OR WS_VISIBLE OR BS_AUTOCHECKBOX,\
                10, 60, 160, 50, hwnd, checkbox_iden, hInstance, NULL
```

Приклад обробки процесу клацання по checkbox може бути таким:

```
        .ELSEIF ax==IDM_CHECKED
                .IF (menuchecked==1)
invoke CheckMenuItem,hMainMenu, IDM_CHECKED, MF_UNCHECKED
                .ELSE
invoke CheckMenuItem,hMainMenu, IDM_CHECKED, MF_CHECKED
                .ENDIF
                xor menuchecked,1
```

```
        .ELSEIF ax==10000                   ; обробляємо кнопку
        invoke MessageBox,0,addr About_string,offset MenuName,MB_OK
        .ELSEIF ax==10001                   ; оброблюваний чекбокс
        invoke GetDlgItem,hwnd,10001
                mov hwndCheck,eax
        invoke SendMessage, hwndCheck, BM_GETCHECK, 0, 0
                mov BX,AX
                .IF (BX==BST_CHECKED)           ; якщо натиснутий, то
                                        ; міняємо заголовок вікна на Checked
        invoke SetWindowText,hwnd, addr txt1  ; встановлює текст
                .ENDIF
                .IF (BX==BST_UNCHECKED) ; якщо знятий, то міняємо
                                                ; заголовок вікна на Unchecked
        invoke SetWindowText,hwnd, addr txt2  ; встановлює текст
                .ENDIF
    ...
```

Для з'ясування стану checkbox йому посилається повідомлення BM_GETCHECK за допомогою виклику функції SendMessage, яка і повертає як значення одну з констант BST_CHECKED або BST_UNCHECKED (насправді є ще одне значення: BST_INDETERMINATE – для невизначеного значення стану check-box'а). Функція SendMessage вимагає як перший параметр HWND вікна, якому посилаємо повідомлення (у нашому випадку це вікно – і це сам checkbox). HWND для нашого checkbox'а визначається шляхом виклику функції GetDlgItem, перший параметр якої – HWND батьківського вікна для елементу управління, а другий – ідентифікатор елемента управління (той самий, який задали при створенні елемента управління у виклику функції CreateWindow).

Після з'ясування стану checkbox'а міняється заголовок головного вікна шляхом виклику функції SetWindowText.

Приклад використання checkbox наведено у лістингу 3.21.6.

Лістинг 3.21.6:

```
    .686    ; директива визначення типу мікропроцесора
.model  flat, stdcall  ; задання лінійної моделі пам'яті та угоди ОС Windows
    option casemap:none      ; відмінність малих та великих  літер
  include \masm32\include\windows.inc ; файли структур, констант …
  include \masm32\macros\macros.asm
  uselib user32,kernel32,gdi32
    IDM_HELLO           equ 1
    IDM_EXIT   equ 2
    IDM_CHECKED equ 3
.data
```

```
        ClassName db "Кнопки и их функционирование",0
        MenuName db "FirstMenu",0 ; Buttons and their application
        Hello_string db "Автор:     НТУ ХПИ, каф ВТП",0
        About_string db "Испытание кнопок",0
        button_class db "button",0
        button_title1 db "Обычная кнопка",0
        button_title2 db "Кнопка чекбокс",0
        button_title3 db "Радиокнопка",0
        button_title3_1  db  "Опция 1",0
        button_title3_2  db  "Опция 2",0
        button_title3_3  db  "Опция 3",0
        button_iden HMENU 10000
        checkbox_iden HMENU 10001
        radiobutton1 HMENU 10002
        radiobutton2 HMENU 10003
        radiobutton3 HMENU 10004
        hwndCheck HWND ?
        menuchecked db 0
        txt1 db "Checked",0
        txt2 db "Unchecked",0
        wc  WNDCLASSEX  <>
        msg MSG       <>
        hwnd HWND     ?
        hInstance HINSTANCE ?
        hMainMenu HMENU ?
.code
start:
        invoke  GetModuleHandle, NULL
        mov     hInstance,eax
        mov     wc.cbSize,SIZEOF WNDCLASSEX
        mov     wc.style, CS_HREDRAW or CS_VREDRAW
        mov     wc.lpfnWndProc, OFFSET WndProc
        mov     wc.cbClsExtra,NULL
        mov     wc.cbWndExtra,NULL
        push            hInstance
        pop     wc.hInstance
        mov     wc.hbrBackground,COLOR_WINDOW+1
        mov     wc.lpszMenuName,OFFSET MenuName
        mov     wc.lpszClassName,OFFSET ClassName
        invoke  LoadIcon,NULL,IDI_APPLICATION
        mov     wc.hIcon,eax
        mov     wc.hIconSm,eax
        invoke  LoadCursor,NULL,IDC_ARROW
        mov     wc.hCursor,eax
        invoke  RegisterClassEx, addr wc
        invoke  CreateWindowEx,NULL, ADDR ClassName,\
```

```
        ADDR ClassName, WS_OVERLAPPEDWINDOW,\
          0, 0,\            ; горизонтальні та вертикальні координати вікна
            450,300,NULL,NULL,\   ; ширина та висота вікна
            hInstance,NULL
      mov     hwnd,eax
      invoke  ShowWindow, hwnd,SW_SHOWNORMAL
        ; Створюємо кнопку
invoke CreateWindowEx,NULL, addr button_class, addr button_title1 ,\
      WS_CHILD OR WS_VISIBLE OR BS_PUSHBUTTON,\
      10, 20, 150, 30, hwnd, button_iden, hInstance, NULL
        ; Створюємо чекбокс
invoke CreateWindowEx,NULL, addr button_class, addr button_title2 ,\
      WS_CHILD OR WS_VISIBLE OR BS_AUTOCHECKBOX,\
      10, 60, 150, 50, hwnd, checkbox_iden, hInstance, NULL
        ; Створюємо радіокнопки
invoke CreateWindowEx,NULL, addr button_class, addr button_title3,\
      WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
      10,120, 150,30 , hwnd, radiobutton1, hInstance, NULL
invoke CreateWindowEx,NULL, addr button_class, addr button_title3,\
      WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
      10,145, 150,30 , hwnd, radiobutton2, hInstance, NULL
invoke CreateWindowEx,NULL, addr button_class, addr button_title3,\
      WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
      10,170, 150,30 , hwnd, radiobutton3, hInstance, NULL
          .WHILE TRUE
             invoke  GetMessage, ADDR msg,NULL,0,0
             or      eax, eax
             jz Quit
             invoke DispatchMessage, ADDR msg
          .ENDW
          Quit:
          mov     eax,msg.wParam
          invoke  ExitProcess, eax
      WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM
.IF uMsg==WM_DESTROY
         invoke PostQuitMessage,NULL
.ELSEIF uMsg==WM_CREATE
invoke GetMenu,hWnd ; витягує дескриптор меню, пов'язаного із вікном
           mov hMainMenu,eax
.ELSEIF uMsg==WM_COMMAND
         mov eax,wParam
          .IF ax==IDM_HELLO
           invoke MessageBox,0,addr Hello_string,offset MenuName,MB_OK
          .ELSEIF ax==IDM_CHECKED
                .IF (menuchecked==1)
```

```
invoke CheckMenuItem,hMainMenu, IDM_CHECKED, MF_UNCHECKED
                .ELSE
invoke CheckMenuItem,hMainMenu, IDM_CHECKED, MF_CHECKED
              .ENDIF
            xor menuchecked,1
           .ELSEIF ax==10000                    ; обробляємо кнопку
       invoke MessageBox,0,addr About_string,offset MenuName,MB_OK
           .ELSEIF ax==10001                    ; оброблюваний чекбокс
            invoke GetDlgItem,hwnd,10001
            mov hwndCheck,eax
            invoke SendMessage, hwndCheck, BM_GETCHECK, 0, 0
            mov BX,AX
.IF (BX==BST_CHECKED) ; натиснутий? – міняємо заголовок на Checked
            invoke SetWindowText,hwnd, addr txt1  ; встановлює текст
.ENDIF
 .IF (BX==BST_UNCHECKED) ; знятий? – міняємо заголовок на Unchecked
            invoke SetWindowText,hwnd, addr txt2  ; встановлює текст
 .ENDIF
        .ELSEIF ax==10002 ; міняємо заголовки вікна при виборі радіокнопки
invoke SetWindowText,hwnd, addr button_title3_1   ; встановлює текст
          .ELSEIF ax==10003
invoke SetWindowText,hwnd, addr button_title3_2  ; встановлює текст
          .ELSEIF ax==10004
invoke SetWindowText,hwnd, addr button_title3_3  ; встановлює текст
          .ELSE
            invoke DestroyWindow,hWnd
          .ENDIF
       .ELSE
         invoke DefWindowProc,hWnd,uMsg,wParam,lParam
            ret
       .ENDIF
       xor    eax,eax
       ret
       WndProc endp
       end start
```

Результат роботи програми наведено на рис. 3.21.10.

Використання радіокнопок

Радіокнопки створюються подібно до звичайних кнопок – для цього треба викликати функцію CreateWindows, указавши в потрібному місці стиль BS_AUTORADIOBUTTON (або BS_RADIOBUTTON – але у цьому випадку необхідно самим писати код, який ставитиме крапку усередині

вибраної радіокнопки). Клас радіокнопки – це клас "button" як і у звичайної кнопки.

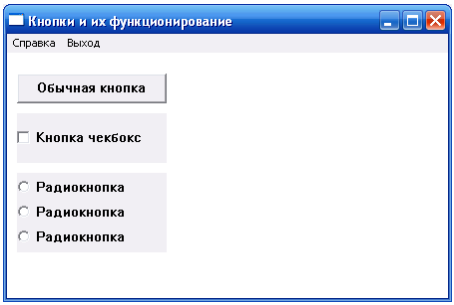


Рис. 3.21.10. Результат роботи програми

Фрагмент коду створення трьох радіокнопок може бути таким:

```
   ...
invoke CreateWindowEx,NULL, addr button_class, addr auto1,\
          WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
                10,120, 150,30 , hwnd, radiobutton1, hInstance, NULL
invoke CreateWindowEx,NULL, addr button_class, addr auto2,\
          WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
                10,145, 150,30 , hwnd, radiobutton2, hInstance, NULL
invoke CreateWindowEx,NULL, addr button_class, addr auto3,\
          WS_CHILD OR WS_VISIBLE OR BS_AUTORADIOBUTTON,\
                10,170, 150,30 , hwnd, radiobutton3, hInstance, NULL
      ...
```

Цей фрагмент коду вставляється у процедуру WinMain після виклику CreateWindow для створення головного вікна.

Обробка повідомлень від радіокнопок виконується у циклі обробника системного повідомлення WM_COMMAND у віконній процедурі. Наприклад, наступний код мінятиме заголовок вікна залежно від того, що за радіокнопку користувач вибрав:

```
      …
      auto1       db       "Опция 1",0
      auto2       db       "Опция 2",0
      auto3       db       "Опция 3",0
      …
.ELSEIF ax==10002 ; міняємо заголовки вікна при виборі радіокнопки
          invoke SetWindowText,hwnd, addr auto1  ; встановлює текст
```

```
.ELSEIF ax==10003
        invoke SetWindowText,hwnd, addr auto2  ; встановлює текст
.ELSEIF ax==10004
        invoke SetWindowText,hwnd, addr auto3  ; встановлює текст
.ELSE
    invoke DestroyWindow,hWnd
     ...
```

Приклад використання радіокнопок наведено у лістингу 3.23.6.

Використання рядка стануStatus Bar

Рядок стану (status bar) – це панель у нижній частині вікна, призначена для виведення допоміжної інформації: параметрів документа, з яким працює користувач, підказок до пунктів меню і так далі (http://docstore.mik.ua/press/skpress/pc_mag/p962.htm, http://www.avinout.com/ praktikum_t4r11part1.html).

Рядок стану створюється як дочірнє вікно на базі зумовленого класу STATUSCLASSNAME за допомогою функції CreateWindowEx. Зазвичай як координати і розміри вікна указуються нульові значення, а як заголовок вікна – NULL. Якщо вказаний стиль вікна SBARS_GRIP, рядок стану розташовується по нижньому краю вікна, а якщо вказаний стиль CCS_TOP – по верхньому краю.

Можна також використовувати спеціальну функцію CreateStatus-Window .

Для виведення простого рядка стану з однією панеллю необхідно зробити в програмі додавання такі, як наприклад:

```
include \masm32\include\comctl32.inc
includelib \masm32\lib\comctl32.lib
…
IDC_STATUS equ 6
Msg2 db "   Строка состояний     НТУ “ХПИ"
…
invoke InitCommonControls ; реєструє і ініціалізує загальні контрольні
                                                          ; віконні класи
…
.ELSEIF uMsg==WM_CREATE ;
invoke CreateStatusWindow,WS_CHILD+WS_VISIBLE,addr Msg2,hWnd,
    IDC_STATUS
```

Рядок стану може працювати в стандартному або спрощеному режимі. У стандартному режимі додаток може розділити вікно StatusBar

на декілька областей для роздільного виведення в них текстової або графічної інформації.

Для розділення смуги перегляду на області необхідно відразу після її створення відправити повідомлення SB_SETPARTS. Як і для панелі інструментів, батьківське вікно повинно посилати рядку стану повідомлення WM_SIZE при зміні розмірів батьківського вікна. Крім того, в цьому випадку необхідно визначити нові розміри областей і знов послати повідомлення SB_SETPARTS.

Для відображення текстової або графічної інформації в кожній області рядка стану використовується повідомлення SB_SETTEXT. Параметр wParam цього повідомлення є логічною комбінацією номера області (починаючи з нуля) та однієї з констант SBT_*, що задають зовнішній вигляд області, а параметр lParam – покажчик на текстовий рядок, що відображається, або довільне 32-розрядне значення для графічного зображення (через це поле можна, наприклад, передати ідентифікатор ресурсу bitmap). Якщо в параметрі wParam використана константа SBT_OWNERDRAW, батьківське вікно отримуватиме повідомлення WM_DRAWITEM з параметром wParam, що містить ідентифікатор органу управління (у нашому випадку – рядку стану). Параметр lParam повідомлення WM_DRAWITEM містить покажчик на структуру типу DRAWITEMSTRUCT, з якої додаток отримує координати області для рисування і виконує рисування за допомогою засобів графічного інтерфейсу GDI. Через одне з полів даної структури передається також вміст параметра lParam повідомлення SB_SETTEXT. Повідомлення SB_SIMPLE використовується для перемикання режимів роботи рядка стану. У спрощеному режимі рядок стану не може бути розділений на області і в ній не можна відобразити графічне зображення.

Рядок стану не посилає повідомлення батьківському вікну за допомогою повідомлень WM_NOTIFY або WM_COMMAND.

3.22. Лабораторна робота 22
ОБРОБКА ПОВІДОМЛЕНЬ ВІД ТАЙМЕРА

Мета заняття

– набути практичних навичок обробки повідомлень від **таймера**, написаних мовою асемблера з використанням API-функцій під Win32.

Постановка задачі

Написати програму з нестандартним вікном та обробкою повідомлень від таймера з пунктами меню й з виконанням завдання згідно з останньою цифрою номера студента в групі:

Методичні вказівки

Рекомендується звернення до сайтів:

http://i-assembler.ru/34/ch3-1.html;

http://sources.ru/msdn/library/using_timers.shtml;

http://www.vsokovikov.narod.ru/New_MSDN_API/Timers/use_timer_trap_mouse.htm;

http://sait.tti.sfedu.ru/electro_book/SRV/laba_1.html.

Варіанти завдань

1. Відстежування часу.
2. Таймер для зворотного відліку часу.
3. Періодичне виконання декількох функцій.
4. Періодичне виведення на екран оновленої інформації.
5. Автозбереження.
6. Задання темпу зміни яких-небудь об'єктів на екрані.
8. Використання функцій таймера для перехоплення введення даних від миші.

9–10. Мультиплікація (виведення декількох графічних об’єктів).

Приклад. Приклад створення вікна з використанням трьох таймерів: за першим таймером видається звуковий сигнал, за другим таймером – відбувається мерехтіння заголовка вікна, за третім – зміна назви вікна (наведений в лістингу 3.22.1). Також програма видає назву комп’ютера.

Лістинг 3.22.1:

```
.686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам’яті та угоди ОС Windows
```

```
        option casemap:none         ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\include\user32.inc ; файли інтерфейсу …
include \masm32\include\kernel32.inc ; системні функції застосувань…
        includelib \masm32\lib\user32.lib
        includelib \masm32\lib\kernel32.lib
        IDM_HELLO  equ 1
        IDM_EXIT     equ 2
.data                               ; директива визначення даних
        ClassName db "SimpleWinClass",0
        MenuName db "FirstMenu",0
        Hello_string db "Деркач Андрей КИТ-19А",0
        About_string db "Испытание таймера",0
        wc  WNDCLASSEX  <>
        msg MSG      <>
        hwnd HWND    ?
        hInstance HINSTANCE ?
      temp db 1
      compName db 30 dup(?),0
        lenname dd $-compName
.code                               ; директива початку сегмента команд
  start:                            ; мітка початку програми з ім'ям start
invoke GetModuleHandle, NULL  ; отримання дескриптора програми
        mov hInstance, eax   ; збереження дескриптора програми
        mov     wc.cbSize,SIZEOF WNDCLASSEX
        mov     wc.style, CS_HREDRAW or CS_VREDRAW
        mov     wc.lpfnWndProc, OFFSET WndProc
        mov     wc.cbClsExtra,NULL
        mov     wc.cbWndExtra,NULL
        push    hInstance
        pop     wc.hInstance
        mov     wc.hbrBackground,COLOR_WINDOW+1
        mov     wc.lpszMenuName,OFFSET MenuName
        mov     wc.lpszClassName,OFFSET ClassName
        invoke  LoadIcon,NULL,IDI_APPLICATION
        mov     wc.hIcon,eax
        mov     wc.hIconSm,eax
        invoke  LoadCursor,NULL,IDC_ARROW
        mov     wc.hCursor,eax
        invoke  RegisterClassEx, addr wc
        invoke  CreateWindowEx,NULL, ADDR ClassName,\
              ADDR ClassName, WS_OVERLAPPEDWINDOW,\
              CW_USEDEFAULT, CW_USEDEFAULT,\
              CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
              hInstance,NULL
        mov     hwnd,eax
```

```
        invoke  ShowWindow, hwnd,SW_SHOWNORMAL
        ; Timer
        invoke SetTimer,hwnd,1,1500,0
        invoke SetTimer,hwnd,2,3667,0
        invoke SetTimer,hwnd,3,1000,0

        .WHILE TRUE                             ; поки істинне, то
invoke  GetMessage, ADDR msg,NULL,0,0  ; читання повідомлення
            or eax, eax                 ; формування ознак
            jz Quit                     ; перейти на мітку Quit, якщо eax = 0
invoke DispatchMessage, ADDR msg ; відправка до WndProc proc
        .ENDW               ; закінчення циклу оброблення повідомлень
Quit:

          invoke KillTimer,hwnd,1
          invoke KillTimer,hwnd,2
          invoke KillTimer,hwnd,3
          mov   eax,msg.wParam
          invoke  ExitProcess, eax

      WndProc   proc   hWnd:HWND,   uMsg:UINT,   wParam:WPARAM,
lParam:LPARAM
  .IF uMsg==WM_DESTROY    ; якщо є повідомлення про знищення вікна
          invoke PostQuitMessage,NULL       ; передача повідомлення
  .ELSEIF uMsg==WM_COMMAND          ; якщо є повідомлення від меню
          mov eax,wParam
           .IF ax==IDM_HELLO                ; якщо є повідомлення IDM_HELLO
            invoke GetComputerName,addr compName, addr lenname
            invoke MessageBox,0,addr compName,offset MenuName,MB_OK
          .ELSE
             invoke DestroyWindow,hWnd   ; знищення вікна
          .ENDIF
.ELSEIF uMsg==WM_TIMER  ; якщо є повідомлення від таймера
            mov eax,wParam
            .IF wParam==1
                        invoke Beep,01234,300
            .ELSEIF wParam==2
                        invoke FlashWindow,hwnd,1
            .ELSE
                        .IF temp==0
            invoke SetWindowText,hwnd,addr Hello_string ; встановлює текст
                        .ELSE
            invoke SetWindowText,hwnd,addr About_string ; встановлює текст
                        .ENDIF
                        xor temp,1
            .ENDIF
```

```
        .ELSE
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; стандартна обробка
                                              ; повідомлень, які явно не оброблюються
    ret
    .ENDIF
    xor   eax,eax                    ; підготування до завершення
    ret                                 ; повернення управління ОС
    WndProc endp                  ; закінчення процедури
    end start                       ; закінчення програми з ім'ям start
```

Лістинг 22.2. Файл ресурсів:

```
    #define IDM_HELLO   1
    #define IDM_EXIT    2
FirstMenu MENU
    {   MENUITEM "Название компьютера", IDM_HELLO
        MENUITEM "Выход",IDM_EXIT
    }
```

API-функція **FlashWindow** призначена для створення вікна з миготливим заголовком.

Функція **GetComputerName** призначена для отримання імені комп'ютера, на якому запускається програма. Функція GetComputerTame повертаємо ім'я комп'ютера через свій перший параметр. Другий же параметр працює і як вхідний, і як вихідний параметр – в останьому випадку він містить фактичну кількість символів в імені комп'ютера. Дана функція повертає ноль, якщо її виклик відбувся невдало.

Функція **GetWindowText** Функція встановлює текст усередині елемента управління або текст заголовка вікна. Для вікон установлюється текст заголовка, для кнопок установлюється текст надпису на кнопці, для полів введення встановлюється текст, який введений в поле. Ця функція протилежна функції **SetWindowText**, яка встановлює віконний заголовок.

З погляду Windows вікна – це не тільки вікна додатка, але і контролі, розташовані на цих вікнах. Наприклад, checkbox'и, кнопки або текстові поля (edit'и). У кожного такого вікна є HWND (тому, власне кажучи, вони і є вікнами).

3.23. Лабораторна робота 23
ОБРОБКА ПОВІДОМЛЕНЬ ВІД КЛАВІАТУРИ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок обробки повідомлень від **клавіатури**, написаних мовою асемблера з використанням API-функцій під Win32.

Постановка задачі

Написати програму обробки повідомлень від клавіатури з відображенням іконки вікна, пунктів меню й з виконанням завдання згідно з останньою цифрою номера студента в навчальній групі. Відобразити у вікні своє прізвище, назву своєї навчальної групи, свого e-mail та своєї фотографії як іконки. Використати функцію MessageBoxIndirect:

1. Установити час затримки для автоповтору клавіатури за допомогою функції SystemParametersInfo.

2. Установити частоту затримки для автоповтору клавіатури за допомогою функції SystemParametersInfo.

3. Установити частоту мигання курсора за допомогою функції SetCaretBlinkTime.

4. Надати можливість встановлення різних мов підтримки клавіатури за допомогою функції LoadKeyboardLayout.

5. Використати блокування подій введення від миші та клавіатури за допомогою функції BlockInput.

6. Використати функцію SetKeyboardState копіювання 256-байтового масиву станів клавіш клавіатури в таблицю станів введення з клавіатури.

7. Використати функцію RegisterHotKey визначення "гарячої" клавіші.

8. Використати функцію keybd_event синтезування натиснутої клавіші для отримання знімку екрана.

9. Використати функцію GetLastInputInfo витягування останньої події введення даних.

10. Використати функцію GetKeyNameText витягування рядка, яка позначає назву клавіші.

Програма вмикання на клавіатурі світлодіодів наведена в лістингу 23.1

Лістинг 3.23.1. Вмикання на клавіатурі світлодіодів:

```
…
_st:
invoke keybd_event, VK_RETURN, 45, 0 , 0 ; натиснення на клавішу Enter
invoke keybd_event, VK_RETURN, 45, KEYEVENTF_KEYUP, 0
                                                ; відтискання клавіші Enter

invoke keybd_event, VK_NUMLOCK, 45, 0 , 0
                                              ; натиснення на клавішу NUMLOCK
invoke keybd_event, VK_NUMLOCK, 45, KEYEVENTF_KEYUP, 0
                                                ; відтискання клавіші NUMLOCK

invoke keybd_event, VK_SCROLL, 45, 0, 0
                                            ; натиснення на клавішу SCROLL LOCK
invoke keybd_event, VK_SCROLL, 45, KEYEVENTF_KEYUP, 0
                                            ; відтискання клавіші SCROLL LOCK

invoke keybd_event, VK_CAPITAL, 45, 0, 0
                                              ; натиснення на клавішу CAPSLOCK
invoke keybd_event, VK_CAPITAL, 45, KEYEVENTF_KEYUP, 0
                                              ; відтискання клавіші CAPSLOCK
 end _st
```

Скен-код обрано як 45, але він може бути яким завгодно.

Розглянемо програму виведення символу натиснутої клавіші у вікно (лістинг 3.23.2). При обробці повідомлень від клавіатури **натиснутий на клавіатурі символ посилається вікну на обробку в параметрі WPARAM** та зберігається в однойменній комірці.

Лістинг 3.23.2. Виведення символу натиснутої клавіші у вікно:

```
 .686                ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
 option casemap:none                ; відмінність малих та великих літер
 include \masm32\include\windows.inc ; файли структур, констант …
  include \masm32\include\windows.inc
  include \masm32\macros\macros.asm
  uselib user32,kernel32,gdi32
  WinMain proto :DWORD,:DWORD

.data                          ; директива визначення даних
  ClassName db "SimpleWinClass",0
AppName  db "Обробка повідомлень від КЛАВІАТУРИ",0     ; назва вікна
char WPARAM 20h ; символ пропуску від клавіатури

.data?
```

```
   hInstance HINSTANCE ?

.code                    ; директива початку сегмента команд
start:                   ; мітка початку програми з ім'ям start
  invoke  GetModuleHandle, NULL        ; отримання дескриптора програми
       mov   hInstance,eax             ; збереження дескриптора програми
  invoke WinMain, hInstance,  SW_SHOWDEFAULT
invoke ExitProcess,eax ; повернення управління та вивільнення ресурсів

   WinMain proc hInst:HINSTANCE, CmdShow:DWORD
       LOCAL wc:WNDCLASSEX ; резервування стека під структуру
       LOCAL   msg:MSG          ; резервування стека під структуру MSG
       LOCAL   hwnd:HWND        ; резервування стека під хендл програми
   push  hInst                  ; збереження в стека  дескриптора програми
   pop     wc.hInstance         ; повернення дескриптора в поле структури
mov  wc.cbSize,SIZEOF WNDCLASSEX          ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW    ; стиль та поведінка вікна
mov  wc.lpfnWndProc, OFFSET WndProc       ; адреса процедури WndProc
mov  wc.hbrBackground,COLOR_WINDOW+1                    ; колір вікна
mov     wc.lpszMenuName,NULL                            ; ім'я ресурсу меню
mov     wc.lpszClassName,OFFSET ClassName               ; ім'я класу
invoke  LoadIcon,NULL,IDI_APPLICATION                   ; ресурс піктограми
mov     wc.hIcon,eax                    ; дескриптор «великої» піктограми
mov     wc.hIconSm,eax                  ; дескриптор маленького віконця
invoke  LoadCursor,NULL,IDC_ARROW                       ; ресурс курсора
mov     wc.hCursor,eax
mov     wc.cbClsExtra,NULL          ; кількість байтів для структури
mov  wc.cbWndExtra,NULL          ; кількість байтів для додаткових структур

invoke  RegisterClassEx, ADDR wc       ; функція реєстрації класу вікна

invoke CreateWindowEx, 0, ADDR ClassName,\; стиль та адр. імені класу
  ADDR AppName,\                        ; адреса імені вікна
  WS_OVERLAPPEDWINDOW, \                ; базовий стиль
  CW_USEDEFAULT,CW_USEDEFAULT,\ ; координати вікна
    600,400, \          ; ширина та висота вікна
      0,0, hInst,NULL  ; дескриптори батьківського вікна та меню і програми
       mov   hwnd,eax
invoke  ShowWindow, hwnd,SW_SHOWNORMAL      ; видимість вікна
invoke UpdateWindow, hwnd   ; оновлення клієнтської області вікна
                                          ; повідомленням WM_PAINT

       .WHILE TRUE                        ; поки істинне, то
invoke  GetMessage, ADDR msg, NULL,0,0      ; читання повідомлення
       .BREAK .IF (!eax)                  ; закінчення циклу при виконанні умови
 invoke TranslateMessage, ADDR msg        ; трансляція до черги
```

```
 invoke DispatchMessage, ADDR msg  ; відсилання повідомлення
.ENDW                              ; закінчення циклу оброблення повідомлень
      mov     eax,msg.wParam
       ret                                    ; повернення з процедури
       WinMain endp              ; закінчення процедури з ім'ям WinMain

WndProc proc hWnd:HWND, uMsg:UINT,\
                                              wParam:WPARAM, lParam:LPARAM
LOCAL hdc:HDC                  ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека
.IF uMsg==WM_DESTROY           ; якщо є повідомлення про знищення вікна
     invoke PostQuitMessage, NULL ; передача повідомлення про знищення
.ELSEIF  uMsg==WM_CHAR         ; якщо є повідомлення від клавіатури,
         push wParam           ; то тимчасово зберегти введений символ
         pop  char             ; та перезаписати в комірку для виведення у вікно
invoke InvalidateRect, hWnd, NULL, TRUE ; виклик функції та WM_PAINT

.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
       invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури
       mov   hdc,eax                   ; збереження контексту
invoke TextOut,hdc,20,20,\ ; виведення тексту та координати тексту
            addr char,1    ; адреса зберігання тексту та кількість байтів тексту
       invoke EndPaint,hWnd, ADDR ps     ; закінчення рисування
        .ELSE                    ; інакше
invoke DefWindowProc,hWnd,uMsg,wParam,lParam  ;  обробка та
                                  ; відправка повідомлення до функції WndProc
         ret                      ; повернення з процедури
.ENDIF                    ; закінчення логічної структури .IF – .ELSEIF
       xor     eax,eax                    ; підготування до закінчення
       ret                                ; повернення з процедури
     WndProc endp                ; закінчення процедури WndProc
       end start                 ; закінчення програми з ім'ям start
```

Як приклад на рис. 3.23.1 наведено результат виконання програми з виведенням символу K.

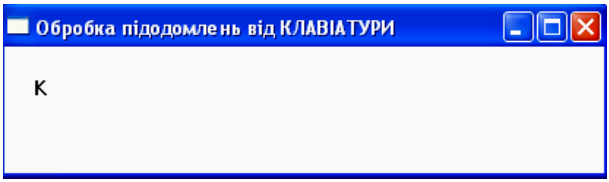


Рис. 3.23.1. Результат виконання програми з лістингу 3.23.2

3.24. Лабораторна робота 24
ОБРОБКА ПОВІДОМЛЕНЬ ВІД МИШІ

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок обробки повідомлень від **миші**, написаних мовою асемблера з використанням API-функцій під Win32.

Постановка задачі

Написати програму обробки повідомлень від кнопок миші зі зміною прозорості, з відображенням іконки вікна, зміни значка відображення курсора, з пунктами меню й з виконанням завдання згідно з останньою цифрою номера студента в групі.

Завдання 1:

1. Після подвійного клацання лівої кнопки в клієнтській області вікна воно починає переміщуватися по вертикалі. Натискання на будь-яку клавішу припиняє рух.

2. Натискання на праву кнопку на клієнтській області вікна приводить до зміни його розмірів (зменшення у 2 рази). Натискання на будь-яку клавішу повертає початкові розміри.

3. Натискання на праву кнопку на клієнтській області вікна приводить до зміни реакції на натиснення цифр. Кожне наступне натиснення цифри змінює розмір вікна. Натискання на будь-яку іншу клавішу повертає початкові розміри.

4. Натискання на праву кнопку у верхній половині вікна приводить до його згортання. Розгортання вікна – натисненням будь-якої функціональної клавіші.

5. Переміщення миші у верхній половині вікна приводить до появи періодичного мерехтіння заголовка вікна. Натискання на певну букву переміщає вікно у правий верхній кут екрану.

6. Створити два вікна, одне з яких – батьківське для іншого. Переміщення миші в неклієнтській області дочірнього вікна приводить до його згортання. Натискання на клавішу «пропуск» при установці фокусу введення на батьківське вікно приводить також до його згортання.

7. Клацання лівою кнопкою в неклієнтській області вікна переміщає його вправо на задану кількість пікселів. Натискання клавіші Alt разом з цифровою клавішею повертає вікно на місце.

8. Клацання правою кнопкою в неклієнтській області вікна приводить до того, що подальше натискання будь-якої буквеної клавіші зрушує вікно на задану кількість пікселів вправо.

9. Подвійне клацання лівої кнопки в робочій області вікна приводить до того, що при натисканні на цифрові клавіші вікно зрушується вниз на кількість пікселів відповідно до натиснутої цифри.

10. Подвійне клацання правою кнопкою в клієнтській області вікна приводить до того, що при натисканні на клавіші стрілок згодом реалізується переміщення вікна по екрану.

11. Клацання правою кнопкою в нижній половині вікна приводить до зміни тексту у заголовку вікна. Подальше натискання на букви приводить до зміни тексту заголовка вікна на натиснуту букву.

12. Переміщення миші в нижній половині вікна максимізує вікно. Натискання на ESC – повернення вікна до колишнього вигляду.

13. Подвійне клацання лівою кнопкою в неклієнтській області вікна максимізує вікно. Натискання на певну букву – повернення до колишніх розмірів.

14. Подвійне клацання правою кнопкою у неклієнтській області вікна приводить до його переміщення у циклі в горизонтальному напрямі вліво та вправо до тих пір, поки не буде натиснута певна клавіша.

15. Переміщення миші в лівій половині вікна змінює колір фону вікна (SetClassLongPtr). Натискання на клавішу F1 відміняє зміну.

16. Подвійне клацання лівою кнопкою у неклієнтській області вікна приводить до згортання його в піктограму. Натискання на функціональну клавішу приводить до розгортання вікна.

17. Створити три вікна, одне з них показати. Подвійне клацання правою кнопкою в неклієнтській області вікна приводить до виведення на екран решти вікон, якщо вони ще не показувалися. Виведення створених, але не показаних вікон дозволене тільки у випадку, якщо не натиснута клавіша Enter.

18. Переміщення миші в неклієнтській області вікна приводить до переміщення вікна у лівий верхній кут екрана. Натискання на клавіші PgUp і PgDn приводить до переміщення вікна з лівого верхнього кута у лівий нижній кут і назад.

19. Подвійне клацання лівої кнопки в клієнтській області вікна приводить до зміни заголовка вікна. Вибір поточного заголовка вікна залежить від кількості натискань на клавішу Insert. Кожне наступне натискання робить активним наступний заголовок з деякого масиву заголовків (циклічно).

20. Вивести на екран два вікна. Подвійне клацання правої кнопки в клієнтській області будь-якого з вікон міняє їх місцями. Зворотний обмін –натисканням на клавішу Delete.

21. Вивести на екран два вікна. Клацання правою кнопкою в правій половині другого вікна змінює заголовок першого. Зворотний обмін – натисканням на клавішу Insert.

22. Вивести на екран три вікна. Клацання лівою кнопкою в правій половині будь-якого вікна приводить до згортання в піктограму останніх. Зворотне розгортання, якщо вони вже згорнуті, здійснити при натисканні на будь-яку цифрову клавішу.

23. Подвійне клацання лівою кнопкою в неклієнтській області вікна приводить до виведення на екран дочірнього вікна. Скрутити в піктограму обидва вікна можна при натисканні на клавіші F1 – F5.

24. Створити два вікна, одне з них показати. Подвійне клацання правою кнопкою в неклієнтській області показаного вікна приводить спочатку до показу другого вікна, потім до його згортання, потім до його розгортання, потім до його закриття.

25. Створити два вікна. Клацання лівою кнопкою в неклієнтській області будь-якого вікна приводить до переміщення їх у лівий верхній кут екрана так, щоб менше було зверху. Натиснення на певну букву закриває обидва вікна.

Завдання 2:

1. Змінити призначення лівої та правої кнопок миші за допомогою функції SwapMouseButton.

2. Відновити значення порога миші та її швидкість переміщення за допомогою функції SystemParametersInfo.

3. Декілька разів змінити на екрані вигляд курсора функцією SetCursor та зробити його невидимим за допомогою функції Show-Cursor.

4. Зафіксувати координати переміщення миші в заданих межах за допомогою функції SystemParametersInfo.

4. Зафіксувати час переміщення миші в заданих межах за допомогою функції SystemParametersInfo.

5. Змінити час реакції системи на подвійне натискання кнопки за допомогою функції SetDoubleClickTime.

6. Заблокувати функціонування миші за допомогою функції BlockInput.

7. Отримати інформацію про кількість кнопок миші за допомогою GetSystemMetrics.

8. При управлінні мишею використати функцію TrackMouseEvent.

9. Використати архів координат про переміщення миші за допомогою функції GetMouseMovePointsEx.

10. Синтезувати переміщення миші та натискання на її кнопки за допомогою функції mouse_event.

Лістинг 3.24.1. Програма створення вікна та виведення тексту в місце екрана при натисканні лівої клавіші миші:

```
      .686                  ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
option casemap:none                  ; відмінність малих та великих літер
include \masm32\include\windows.inc ; файли структур, констант …
include \masm32\macros\macros.asm
 uselib user32,kernel32,gdi32, shell32
  WinMain proto :DWORD,:DWORD
.data                                          ; директива визначення даних
  ClassName db "SimpleWinClass",0
AppName  db "Обробка повідомлень від лівої клавіші миші  ",0 ;
      MouseClick db 0        ; прапорець натискання лівої клавіші
.data?
      hInstance HINSTANCE ?
      myXY POINT <>  ; структура визначення координат
.code                                ; директива початку сегмента команд
start:                               ; мітка початку програми з ім'ям start
invoke  GetModuleHandle, NULL          ; отримання дескриптора програми
      mov   hInstance,eax              ; збереження дескриптора програми
invoke WinMain, hInstance, SW_SHOWDEFAULT
invoke ExitProcess,eax; повернення управління ОС та вивільнення ресурсів


WinMain proc hInst:HINSTANCE,\ ; дескриптор екземпляра застосування
                        CmdShow:DWORD ; вигляд вікна
      LOCAL wc:WNDCLASSEX ; резервування стека  під структуру
      LOCAL   msg:MSG          ; резервування стека під структуру MSG
      LOCAL   hwnd:HWND       ;  резервування стека під хендл програми
      push  hInst                    ; збереження в стеку дескриптора програми
      pop     wc.hInstance           ; повернення дескриптора в поле структури
mov  wc.cbSize,SIZEOF WNDCLASSEX          ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW   ; стиль та поведінка вікна
mov  wc.lpfnWndProc, OFFSET WndProc       ; адреса процедури WndProc
mov  wc.hbrBackground,COLOR_WINDOW+1                    ; колір вікна
mov    wc.lpszMenuName,NULL                          ; ім'я ресурсу меню
mov    wc.lpszClassName,OFFSET ClassName                    ; ім'я класу
invoke  LoadIcon,NULL,IDI_APPLICATION                 ; ресурс піктограми
mov     wc.hIcon,eax                    ; дескриптор «великої» піктограми
mov    wc.hIconSm,eax                 ; дескриптор маленького віконця
```

```
invoke  LoadCursor,NULL,IDC_ARROW                    ; ресурс курсора
mov  wc.hCursor,eax
mov  wc.cbClsExtra,NULL                    ; кількість байтів для структури
mov  wc.cbWndExtra,NULL          ; кількість байтів для додаткових структур
invoke  RegisterClassEx, ADDR wc         ; функція реєстрації класу вікна
invoke CreateWindowEx, 0, ADDR ClassName,\ ; стиль та адреса імені класу
  ADDR AppName,\                          ; адреса імені вікна
  WS_OVERLAPPEDWINDOW, \         ; базовий стиль
  CW_USEDEFAULT,CW_USEDEFAULT,\ ; координати вікна
    600,400, \          ; ширина та висота вікна
     0,0, hInst,0 ; дескриптори батьківського вікна, меню та програми
       mov   hwnd,eax
invoke  ShowWindow, hwnd,SW_SHOWNORMAL       ; видимість вікна
 invoke UpdateWindow, hwnd            ; оновлення клієнтської області вікна
                                                  ; повідомленням WM_PAINT
      .WHILE TRUE                                ; поки істинне, то
invoke  GetMessage, ADDR msg,NULL,0,0     ; читання повідомлення
        .BREAK .IF (!eax)          ; закінчення циклу при виконанні умови
 invoke DispatchMessage, ADDR msg  ; відсилання повідомлення
      .ENDW                          ; закінчення циклу оброблення повідомлень
      ;  mov     eax,msg.wParam
      ret                                    ; повернення з процедури
      WinMain endp             ; закінчення процедури з ім'ям WinMain

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM
LOCAL hdc:HDC              ; резервування стека під хендл вікна
LOCAL ps:PAINTSTRUCT ; резервування стека під структуру PAINTSTRUCT
.IF uMsg==WM_DESTROY        ; якщо є повідомлення про знищення вікна
     invoke PostQuitMessage,NULL ; передача повідомлення про знищення
.ELSEIF uMsg==WM_LBUTTONDOWN     ; повідомлення від лівої клавіші
      mov eax,lParam          ; 32-розрядні координати курсора миші
      and eax,0ffffh          ; виділення молодшої частини – координати X
      mov myXY.x,eax          ; збереження координати X
      mov eax,lParam           ; 32-розрядні координати курсора миші
      shr eax,16                ; зсув праворуч на 4 байти для виділення  У
      mov myXY.y,eax          ; збереження координати У
      mov MouseClick,TRUE ; підтверджено отримання координат
   invoke InvalidateRect,hWnd,NULL,TRUE ; виклик функції та WM_PAINT
                          ; для перерисовування клієнтської області вікна
.ELSEIF uMsg==WM_PAINT ; якщо є повідомлення про перерисовування
      invoke BeginPaint,hWnd, ADDR ps ; виклик підготовчої процедури та
                                                  ; заповнення структури
      mov     hdc,eax          ; збереження контексту
     .IF MouseClick            ; перевірка встановлення прапорця
      invoke lstrlen,ADDR AppName ; визначення довжини рядка
```

```
invoke TextOut,hdc,\                    ; виведення тексту
  myXY.x,myXY.y, \ ; координати початку тексту
 ADDR AppName, eax ; адреса зберігання тексту та кількість байтів тексту
.ENDIF
       invoke EndPaint,hWnd, ADDR ps
.ELSE                                   ; інакше
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam  ;  обробка та
                                        ; відправка повідомлення до функції WndProc
          ret                                           ; повернення з процедури
.ENDIF                                  ; закінчення логічної структури .IF – .ELSEIF
       xor     eax,eax                  ; підготування до закінчення
       ret                                           ; повернення з процедури
     WndProc endp                               ; закінчення процедури WndProc
       end start                               ; закінчення програми з ім'ям start
```

Результат виконання програми з лістингу 3.24.1 наведено на рис. 3.24.1.

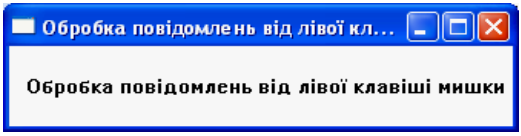


Рис.3.24.1. Результат виконання програми з лістингу 3.24.1

Розглянемо можливість програмної зміни швидкості переміщення миші. Файл ресурсів такої програми може бути таким, як наведено в лістингу 3.24.2.

Лістинг 3.24.2. Файл ресурсів програмної зміни швидкості переміщення миші:

```
#define IDM_SPEED1   1
#define IDM_SPEED2   2
#define IDM_SPEED3   3
#define IDM_EXIT    4
#define IDM_ABOUT   5
#define IDI_ICON 22

IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL  "mouse.ico"

FirstMenu MENU {
    POPUP "Операции"{
       MENUITEM "Скорость 1",IDM_SPEED1
      MENUITEM SEPARATOR
```

```
        MENUITEM "Скорость 2",IDM_SPEED2
        MENUITEM SEPARATOR
        MENUITEM "Скорость 3",IDM_SPEED3
        MENUITEM SEPARATOR
          MENUITEM "Выход",IDM_EXIT
       }
      POPUP "Автор"{
        MENUITEM "About",IDM_ABOUT
       }
}
```

Результатом виконання файлу ресурсів буде вікно з меню, як наведено на рис. 3.24.1.

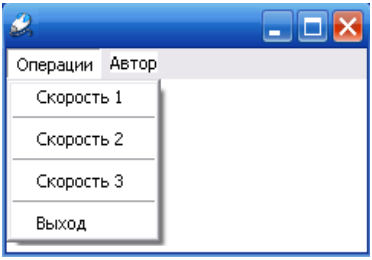


Рис. 3.24.1. Вигляд вікна виконання файлу ресурсів

Фрагмент програми зміни швидкості переміщення миші може бути таким, як наведено в лістингу 3.24.3.

Лістинг 3.24.3. Фрагмент програми зміни швидкості переміщення миші:

```
...
.const
IDM_SPEED1 equ 1
IDM_SPEED2 equ 2
IDM_SPEED3 equ 3
IDM_EXIT      equ 4
IDM_ABOUT  equ 5
IDI_ICON      equ 22
…
WndProc proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
    LOCAL hdc:HDC       ; резервування стека під хендл вікна
```

```
.if uMsg==WM_DESTROY
        invoke PostQuitMessage,NULL

.ELSEIF uMsg==WM_CREATE   ; обробка  повідомлення WM_CREATE

.elseif uMsg==WM_COMMAND
        mov eax,wParam

        .if ax==IDM_SPEED1
invoke GetDC, hWnd ; отримати дескриптор контексту пристрою
        mov hdc, eax   ; збереження
invoke SystemParametersInfo,SPI_SETMOUSESPEED,NULL,1,\
 SPIF_SENDCHANGE
        invoke ReleaseDC, hWnd, hdc

        .elseif ax==IDM_SPEED2
invoke SystemParametersInfo,SPI_SETMOUSESPEED,NULL,10,\
 SPIF_SENDCHANGE

        .elseif ax==IDM_SPEED3
invoke SystemParametersInfo,SPI_SETMOUSESPEED,NULL,20,\
 SPIF_SENDCHANGE

        .elseif ax==IDM_ABOUT
invoke MessageBox,NULL,ADDR About_string,OFFSET AppName,MB_OK

        .else
        invoke DestroyWindow,hWnd
        .endif
.else
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        Ret
.endif
        …
```

3.25. Лабораторна робота 25
ПРОЦЕС

Мета заняття

– набути практичних навичок створення **процесів**, написаних мовою асемблера з використанням API-функцій під Win32.

Постановка задачі

Написати головну програму, яка за допомогою меню та кнопок керує іншими діями, заданими в варіантах завдання. Надати можливість виведення назв файлів програм та примусового їх завершення. Варіанти завдань такі (узгоджуються з викладачем):

1. Читання файлу з інтернету та його виконання не більше 3 разів та програми калькулятора (calc.exe) не більше 2 разів. Програма з інтернету виконується 15 секунд, а програма calc.exe виконується 30 секунд.

2. Завершити процес explorer.exe, а потім запустити його. Виконати дії з маніпуляцією цієї програми (наприклад, виконати такі дії декілька разів).

3. Передати повідомлення з однієї програми в дві інші.

4. Запуск однієї програми, створеної особисто, та програми калькулятора (calc.exe). Особисто створена програми виконується 15 секунд, а програма calc.exe виконується 20 секунд.

5. Запуск двох програм, створених особисто та якої-небудь програми з системної папки, наприклад для Windows XP: C:\Windows\system32 та їх завершення через різні інтервали часу.

6. Запуск однієї програми, створеної особисто, не більше 3 разів та програми калькулятора (calc.exe) не більше 4 разів. Особисто створена програми виконується 10 секунд, а програма calc.exe виконується 20 секунд.

7. Запуск однієї програми, створеної особисто, не більше 4 разів та програми налагоджувача (OllyDbg.exe) не більше 3 разів. Обмежити час їх виконання.

8*. Примусове завершення системних процесів.

9*. Завершення довільного процесу, а також всіх процесів, які породжені ним і його дочірніми процесами.

Приклад 3.25.1. Створення програми запуску зовнішньої програми.

Лістінг 25.1. Програма запуску зовнішньої програми:

```
.686                    ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
    option casemap:none         ; відмінність малих та великих літер
```

```
    include \masm32\include\windows.inc   ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib kernel32, user32   ; макрос з підключеними бібліотеками
.data
    buff   db 100 dup(?)
    ; programname db  "c:\windows\system32\calc.exe",0
    programname db  "1-1.exe",0
    titl   db   'Level 0',0
    mask1  db   'My ProcID = %i (decimal)',0
    startInfo dd ?
processInfo PROCESS_INFORMATION <> ; інформація про процес і його первинну нитку
.code
    start:
invoke CreateProcess, ADDR programname, 0,0,0,FALSE,\
   NORMAL_PRIORITY_CLASS, 0,0, ADDR startInfo, ADDR processInfo
invoke GetCurrentProcessId ; витягує псевдодескриптор для процесу
invoke  wsprintf, addr buff, addr mask1, eax
invoke  MessageBox,NULL,addr  buff,  addr  titl,
MB_ICONINFORMATION+180000h
    invoke  ExitProcess,0
    end start                ; закінчення програми з ім'ям start
```

Результат виконання програми наведено на рис. 3.25.1.

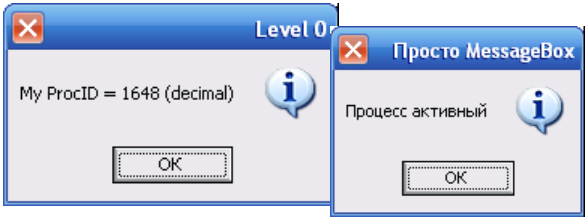


Рис. 3.25.1. Результат виконання програми з лістингу 3.25.1

Програма з іменем 1-1.asm наведена в лістингу 3.25.2.

Лістинг 3.25.2. Програма з іменем 1-1.asm, яку запускають як зовнішню:

```
    .686                ; директива визначення типу мікропроцесора
.model flat,stdcall ; задання лінійної моделі пам'яті та угоди ОС Windows
    option casemap:none      ; відмінність малих та великих літер
    include \masm32\include\windows.inc ; файли структур, констант …
    include \masm32\macros\macros.asm
    uselib kernel32,user32
.data
```

```
    st1 db "Процесс активный",0        ; буфер виведення повідомлення
    titl db "Просто MessageBox",0
.code                                  ; директива визначення даних
start:
invoke MessageBox,0 \                  ; API-функція виведення вікна консолі
  addr st1, \                          ; адреса рядка, яка містить  текст повідомлення
    addr titl, \                       ; адреса рядка, яка містить  заголовок повідомлення
    MB_ICONINFORMATION+180000h         ; вигляд вікна
invoke ExitProcess, 0 ; повернення управління ОС та вивільнення ресурсів
    end start                          ; директива закінчення програми з іменем start
```

Як приклад можна створити декілька однакових програм з різними іменами. У кожній програмі в секції даних записані імена файлів, які вона викликає й виконує з цього списку. Таким чином, одна програма викликає наступну.

Наприклад, програма з іменем Proc1 містить рядок

```
programname db  "1-1.exe",0
```

Програма з іменем Proc2 містить рядок

```
programname db  " Proc1.exe",0
```

Програма з іменем Proc3 містить рядок

```
programname db  " Proc2.exe",0
```

У такій послідовності при запуску програми Proc3.exe буде виконана програма Proc2.exe, яка викликає програму Proc1.exe. А програма Proc1.exe вже викликає програму 1-1.exe.

Якщо необхідно запускати програму декілька раз, то треба вставити лічильник.

Приклад 3.25.2. Читання файлу з інтернету та його виконання.

Лістинг 3.25.3. Читання файлу з інтернету та його виконання:

```
    .386
    .model flat, stdcall
    option casemap :none
     include \masm32\include\windows.inc
    include \masm32\macros\macros.asm
    uselib kernel32, user32, urlmon, shell32
.code
    start:
invoke URLDownloadToFile, 0, chr$("http://kaimi.ru/hello_world.exe"), \
  chr$("hello_world.exe"), 0, 0
invoke ShellExecute, 0, chr$("open"), chr$("hello_world.exe"), 0, 0, \
  SW_SHOWNORMAL
    invoke ExitProcess, 0
    end start
```

3.26. Лабораторна робота 26
РЕЄСТР

Мета заняття

– поглибити і закріпити знання з архітектури МП платформи x86 і навички його програмування;

– набути практичних навичок управління **реєстром** з використанням API-функцій під Win32.

Постановка задачі

Завдання 1

Написати програму створення ключа залежно від указаної гілки. Рядковий тип даних повинен зберігати відомості про прізвище розробника та його e-mail. Через відсутність деяких ключів розроблене застосування виконуватися не повинно. При завершенні роботи застосування повинна проводитися дереєстрація, тобто програмне видалення ключів і гілок, створених програмою.

1. Контекстне меню панелі завдань, меню папок і файлів (гілка HKCU\ Software\Microsoft\Windows\CurrentVersion\Policies\Explorer HKCR*\ shellex\ ContextMenuHandlers, HKCR\Directory\shell, HKCR\Folder\ shell, ключ: NoTrayContextMenu).

2. Діалогове вікно відкриття і збереження файлу (гілка HKCU\ SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\comdlg32, розділ "PlacesBar", параметри: NoPlacesBar, NoBackButton, NoFileMru).

3. Диспетчер завдань Windows XP і "синій екран смерті" Windows XP (гілка HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\ System та HKLM\SYSTEM\CurrentControlSet\Servises\i8042prt\Parameters відповідно. Ключі: DisableTaskMgr та CrashOnCtrlScroll).

4. Управління годинником, яке включає: синхронізацію системного годинника, вибір time-серверів, прикрасу годинників (гілки: HKLM\SYSTEM\ControlSet001\Services\W32Time\TimeProviders\NtpClient, HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ DateTime\Servers, HKCU\Control Panel\International. Ключі: Special-PollInterval, sTimeFormat).

5. Дискові операції з перевірки диска з автоматичного виправлення помилок, зміни часу очікування (гілки: HKEY_USERS\DEFAULT\ SOFTWARE\Microsoft\Windows\CurrentVersion\ Applets\Check Drive та HKLM\SYSTEM\CurrentControlSet\Control\Session Manager. Ключі: Auto-Chk та AutoChkTimeOut).

6. Повідомлення при завантаженні, автозавантаженні (гілки HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon та HKLM\ SoftWare\Microsoft\Windows\CurrentVersion. Ключі: LegalNoticeCaption, LegalNoticeText, RunOnce, RunOnceEx, RunServices, RunServicesOnce, RunServices, DisableLocalMachineRun, DisableLocalMachineRunOnce, DisableCurrentUserRun).

7. Заборона на доступ до вмісту вибраних дисків (гілка HKEY_ CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\ Policies\ Explorer. Ключ NoViewOnDrive). Організувати діалог за вибором найменувань маскованих дисків.

8. Панель перемикача завдань (гілка HKEY_CURRENT_ USER\Control Panel\Desktop. Ключі: CoolSwitch, CoolSwitchRows та CoolSwitchColumns).

9. Реєстраційні дані (гілка HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\Windows NT\CurrentVersion).

10. Паролі та безпека (гілка HKLM\SOFTWARE\Microsoft\ Windows\CurrentVersion\Policies\Network, параметри: NoDialIn, Disable-PwdCaching, HideSharePwds, NoFileSharing, NoFileSharingControl, NoPrintSharing, NoPrintSharingControl).

Приклад 3.26.1. Написати програму додавання у гілку HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer ключа NoTrayContextMenu.

Файл ресурсів програми наведено у лістингу 3.26.1.

Лістинг 26.1. Файл ресурсів:

```
#define IDM_CREATEKEY  1
#define IDM_DELETEKEY  2
#define IDM_ABOUT 3
#define IDM_EXIT 4
#define IDB_MAIN 5
#define IDI_ICON 22
IDI_ICON ICON DISCARDABLE MOVEABLE LOADONCALL  "butterfly.ico"
FirstMenu MENU
{
        POPUP "Меню"{
      MENUITEM "Создать ключ",IDM_CREATEKEY
      MENUITEM SEPARATOR
      MENUITEM "Удалить ключ",IDM_DELETEKEY
            }
        POPUP "Справка"{
     MENUITEM "About",IDM_ABOUT
              }
      MENUITEM "EXIT",IDM_EXIT
}
```

Вигляд файлу ресурсів наведено на рис. 3.26.1.

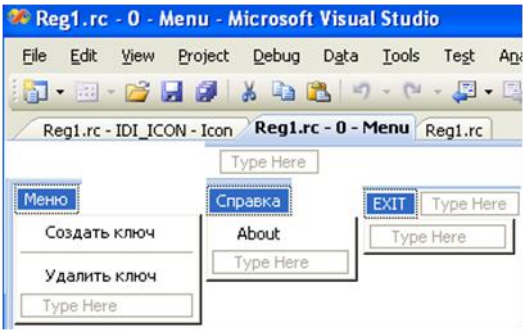


Рис. 3.26.1. Вигляд файлу ресурсів програми для прикладу 3.26.1

В останній кнопці файлу ресурсів відсутня спливаюча кнопка. За обробкою цієї кнопки з надписом EXIT здійснюється закриття вікна й вихід з програми. Програма виконання прикладу 3.26.1 наведена в лістингу 3.26.2.

Лістинг 3.26.2:

```
  ; (гілка HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
   ; ключ:NoTrayContextMenu)
          .686              ; директива визначення типу мікропроцесора
.model flat,stdcall  ; задання лінійної моделі пам’яті та угоди ОС Windows
option casemap:none          ; відмінність малих та великих літер
include \masm32\include\windows.inc   ; файли структур, констант …
include \masm32\macros\macros.asm
uselib kernel32, user32? gdi3, advapi32, ;
          ; числові значення кнопок меню (1-4) та іконки (22)
IDM_CREATEKEY equ 1
IDM_DELETEKEY equ 2
IDM_ABOUT equ 3
IDM_EXIT  equ 4
IDI_ICON  equ 22

   WinMain proto hInst:HINSTANCE,CmdShow:DWORD
.data                                        ; директива визначення даних
  ClassName db "Firstclass",0
  AppName db "Программа установки и снятия ключей в реестре",0
  MenuName db "FirstMenu",0
  Msg db "Рысованый А.Н. ",0ah,0dh,"rysov@rambler.ru", 0
  params MSGBOXPARAMS <>
```

```
  Titl db "Autor",0
  szREGSZ db "REG_SZ",0 ; рядок з назвою ключа має 0 на кінці
  szTestKey db "Software\Microsoft\Windows\CurrentVersion\Policies\
Explorer",0
  setValue db "0",0                                   ; розмір ключа в байтах
  ValSize1 db 4                                       ; розмір ключа в байтах
  szValueName1 db "NoTrayContextMenu",0
  ;getValue1 db 7 DUP(?),0
.data?
      hKey dd ?
      lpdwDisp dd ?
      hInstance HINSTANCE ?
.code                       ; директива початку сегмента даних
_st:                        ; мітка початку програми
invoke GetModuleHandle, NULL ; отримання дескриптора програми
          mov hInstance,eax      ; збереження дескриптора програми
invoke WinMain,hInstance,SW_SHOWDEFAULT
invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,CmdShow:DWORD
      LOCAL wc:WNDCLASSEX    ; резервування стека під структуру
      LOCAL msg:MSG          ; резервування стека під структуру MSG
      LOCAL hwnd:HWND      ;  резервування стека під хендл програми
  mov wc.cbSize,SIZEOF WNDCLASSEX   ; кількість байтів структури
mov wc.style, CS_HREDRAW or CS_VREDRAW ; стиль та поведінка вікна
  mov   wc.lpfnWndProc, OFFSET WndProc  ; адреса процедури WndProc
  mov   wc.cbClsExtra,0                 ; кількість байтів для структури
  mov   wc.cbWndExtra,0                 ; кількість байтів для структури
  push  hInstance               ; збереження в стеці дескриптора програми
  pop   wc.hInstance            ; повернення дескриптора в поле структури
  mov   wc.hbrBackground,COLOR_WINDOW+1      ; колір вікна
  mov   wc.lpszMenuName,OFFSET MenuName      ; ім'я ресурсу меню
  mov   wc.lpszClassName,OFFSET ClassName        ; ім'я класу
invoke LoadIcon,hInstance,IDI_ICON ; відображається особиста іконка
  mov   wc.hIcon,eax                    ; дескриптор піктограми
  mov   wc.hIconSm,eax
  invoke LoadCursor,0,IDC_ARROW  ; курсор – стандартна стрілка
  mov wc.hCursor,eax
invoke RegisterClassEx,addr wc ; регєстрація класу вікна
invoke CreateWindowEx, \       ; функція створення вікна за зразком
 NULL,ADDR ClassName, \        ; стиль та адреса імені класу
ADDR AppName,WS_OVERLAPPEDWINDOW,\ ; адр. імені вікна,баз. стиль
 500,500,400,120,0,0,hInst,0 ;
      mov hwnd,eax
invoke ShowWindow,hwnd,SW_NORMAL
invoke UpdateWindow, hwnd
```

```
    .WHILE TRUE                          ; поки істинне, то
 invoke GetMessage,AddR msg, NULL,0,0    ; читання повідомлення
   or eax,eax
   jz Quit
invoke DispatchMessage,AddR msg ; відсилання повідомлення проц. вікна
    .ENDW                                ; закінчення циклу оброблення повідомлень
Quit:
 mov eax,msg.wParam
 ret                          ; повернення з процедури
WinMain endp                  ; закінчення процедури з ім'ям WinMain

WndProc proc hWnd:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
       LOCAL hdc:HDC                  ; резервування стека під хендл вікна
       LOCAL hkey:HKEY
.IF uMsg==WM_DESTROY      ; обробка повідомлення про знищення вікна
       invoke PostQuitMessage,0     ; передача повідомлення про знищення
.ELSEIF uMsg==WM_COMMAND             ; обробка повідомлень від меню
       mov eax,wParam               ; збереження ідентифікатора кнопки меню
       .IF ax==IDM_CREATEKEY ; якщо вибрана кнопка "Создать ключ"
invoke RegCreateKeyEx,HKEY_CURRENT_USER, ADDR szTestKey,0,\
 ADDR szREGSZ, REG_OPTION_VOLATILE,\     ; опції ключа
  KEY_ALL_ACCESS,\                       ; права доступу до ключа
   0,ADDR hKey,\               ; адреса хендлу створюваного ключа
   ADDR lpdwDisp    ; місце для зберігання інформації про створений ключ
              .IF eax == ERROR_SUCCESS          ; ключ створено успішно ?
;invoke RegQueryValueEx,hKey,ADDR szValueName1,0,0,ADDR getValue1,\
;  ADDR ValSize1     ; функція відновлення типу та даних відкритого ключа
invoke RegSetValueEx,\      ; встановлення значення для вказаного ключа
  hKey,ADDR szValueName1,\       ; ім'я ключа для установки
   0,REG_SZ,\                    ; 0, рядок з назвою ключа має 0 на кінці
    ADDR setValue,\              ; розмір ключа в байтах
     ValSize1                    ; розмір ключа в байтах
invoke RegCloseKey,hKey    ; ідентифікація відкритого ключа для закриття
              .ENDIF
     .ELSEIF ax==IDM_DELETEKEY ; якщо вибрана кнопка "Удалить ключ"
invoke RegDeleteKey,HKEY_CURRENT_USER,ADDR szTestKey
              .IF eax == ERROR_SUCCESS        ;  ключ створено успішно ?
invoke RegCloseKey,hKey             ; закриття ключа в системному реєстрі
              .ENDIF
      .ELSEIF ax==IDM_ABOUT ; якщо вибрана кнопка "Справка"
           mov params.cbSize,SIZEOF MSGBOXPARAMS  ; розмір структури
           mov params.hwndOwner, 0                ; дескриптор вікна власника
invoke GetModuleHandle, 0                 ; отримання дескриптора програми
       mov params.hInstance, eax          ; збереження дескриптора програми
       mov params.lpszText, offset Msg                ; адреса повідомлення
       mov params.lpszCaption,offset Titl             ; адреса заголовка вікна
```

```
        mov params.dwStyle, MB_USERICON    ; стиль вікна
        mov params.lpszIcon, IDI_ICON      ; ресурс значка
        mov params.dwContextHelpId, 0      ; контекст довідки
        mov params.lpfnMsgBoxCallback, 0 ;
        mov params.dwLanguageId, LANG_NEUTRAL  ; мова повідомлення
invoke MessageBoxIndirect, ADDR params ;
            .ELSEIF ax==IDM_EXIT ; якщо вибрана кнопка "Выход"
invoke DestroyWindow,hWnd  ; знищення вікна
            .ELSE
            .ENDIF
.ELSE
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; знищення вікна
                                                    ; (обов'язкове)
        ret                                         ; повернення з процедури
.ENDIF
        xor    eax,eax                      ; підготування до закінчення
        ret                                   ; повернення з процедури
        WndProc endp                        ; закінчення процедури WndProc
        end _st                           ; закінчення програми з ім'ям _st
```

У програмі для створення ключа використовується функція **RegCreateKeyEx.** Ця функція після створення повертає значення ERROR_SUCCESS (ключ створено успішно?). Після перевірки цього значення викликається функція **RegSetValueEx –** встановлення значення для вказаного ключа та конкретизуються ці значення. Функція **RegCloseKey** – ідентифікація відкритого ключа – завершує процес створення ключа.

У гілці обробки повідомлення про закриття ключа **.ELSEIF ax==IDM_DELETEKEY** теж оброблюється значення ERROR_SUCCESS.

У програмі для виведення довідки про автора використовується функція MessageBoxIndirect теж зі своєю іконкою.

Зовнішній вигляд програми наведено на рис. 3.26.2.

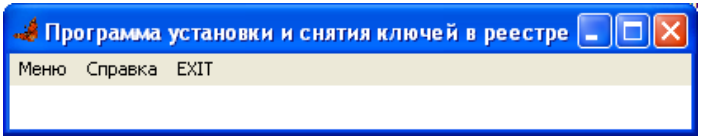


Рис. 3.26.2. Зовнішній вигляд програми для прикладу 3.26.1

Вигляд програми до створення ключа наведено на рис. 3.26.3.

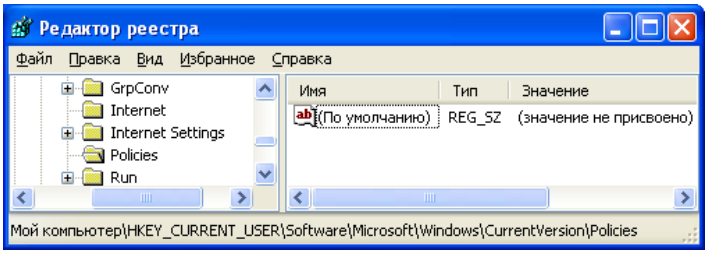


Рис. 3.26.3. Вигляд реєстру до внесених змін

Вигляд програми після створення ключа наведено на рис. 3.26.4.

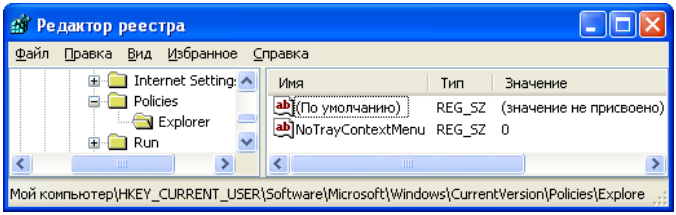


Рис. 3.26.4. Вигляд реєстру до внесених змін

Спрощене вікно довідки про автора програми, яке формується функцією MessageBoxIndirect, наведено на рис. 3.26.5.

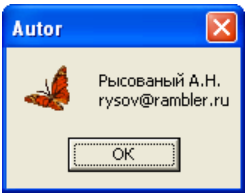


Рис. 3.26.5. Спрощене вікно довідки про автора програми

3.27. Лабораторна робота 27
ДОСЛІДЖЕННЯ КОДУ ПРОГРАМ

Мета заняття

– набути практичних навичок у дослідженні коду програм з використанням API-функцій під Win32 для МП платформи x86.

Постановка задачі

Написати діалогову програму під Windows на мові **асемблера** з відповідними повідомленнями та знайти в створених exe-файлах місця знаходження ключів реєстрації програм:

1. Вимоги реєстрації із збереженням ключа безпосередньо в програмі.
2. Вимоги реєстрації зі збереженням ключа в масиві збереження даних.

Вимоги до програм

– вивести дані про автора;

– передбачити повідомлення при неправильних діях або некоректних даних.

Зміст звіту

1. Постановка задачі.
2. Блок-схема алгоритму виконання прикладу з детальним коментарем та описом роботи.
3. Лістинг програми та коментарі до всіх команд.
4. Print screen екрана 32-розрядного налагоджувача з виконанням програми та результатами виконання.
5. Висновки за результатами роботи.

Приклад 3.27.1. Написати програму на мові C++ з реєстрацією збереження ключа в програмі. Текст програми може бути таким, як наведено в лістингу 3.27.3.

Лістинг 3.27.1:

```
#include <iostream>
void main()
{
        int password;
C:
```

```
        system("cls");
        std::cout << " Enter pssword " <<std::endl;
        std::cin >> password;
        if (password==12345)
        {
                std::cout <<"You entered a correct password"<< std::endl;
        }
        else
        {
                std::cout <<"You entered a wrong password"<< std::endl;
                system("pause");
                goto C;
        }
        system("pause");
}
```

Для програми з лістингу 3.27.1 отримати exe-файл, при запуску якої виведеться повідомлення про регістрацію з вимогою ввести ключ. Якщо ключ співпадає із тим, що зберігається в програмі, то буде виведено повідомлення про реєстрацію програми. (рис. 3.27.1).

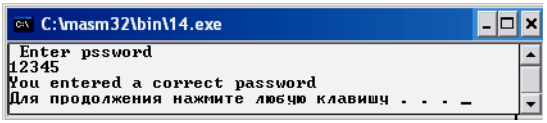


Рис. 3.27.1. Запит про реєстрацію програми з лістингу 3.27.1

Якщо ж ключ введено неправильний, то буде виведено відповідне повідомлення та меню про продовження роботи або вихід (рис. 3.27.2).

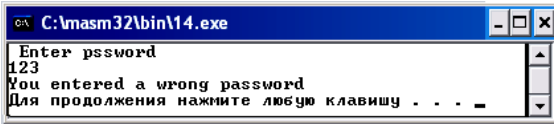


Рис. 3.27.2. Запит при введенні неправильного ключа

Щоб програма приймала будь-який ключ треба змінити її код у налагоджувачі. Відкриваємо exe-файл у налагоджувачі OllyDbg (рис. 3.27.3).

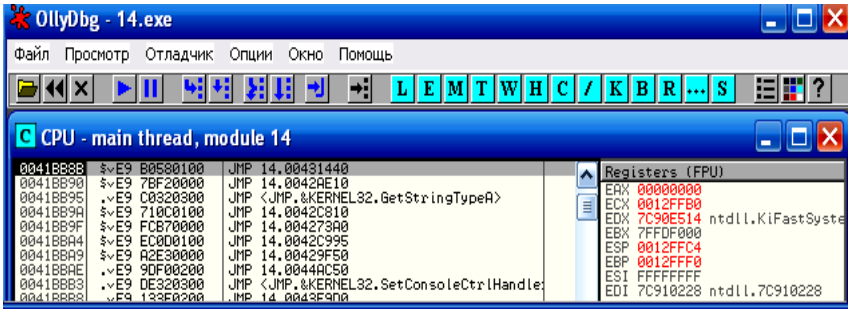


Рис. 3.27.3. Відкритий exe-файл у налагоджувачі OllyDbg

При натисканні на функціональну клавішу F8 налагоджувач перейде на початок програми, вигляд якої наведено на рис. 3.27.4.

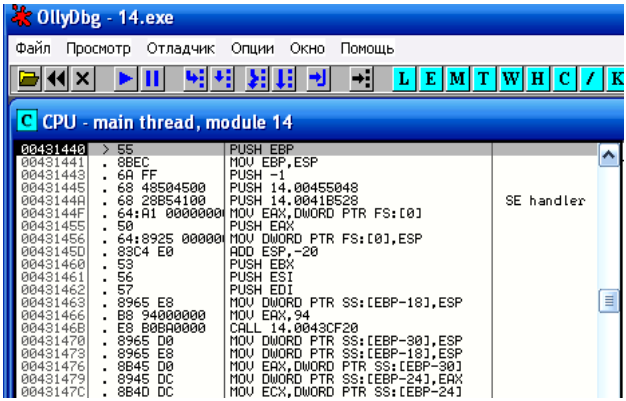


Рис. 3.27.4. Початок програми

Натискаючи F8, починаємо покроково трасувати програму, поки не виведеться повідомлення про введення ключа. Ту команду, яка вивела відповідне повідомлення, необхідно відмітити, тобто поставити контрольну точку. Для цього, підводимо курсор на цю команду та натискуємо ліву клавішу миші, а також функціональну клавішу F2. Адреса команди підсвітиться червоним кольором (рис. 3.27.5).

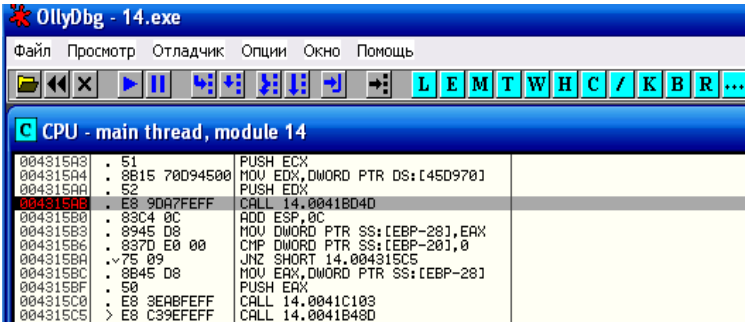


Рис. 3.27.5. Виділення команди виведення повідомлення

Після виділення цієї команди (як правило – команду call) знову перезапускаємо програму. Для цього натискаємо кнопку  , а потім кнопку  , або F9. Виконання програми зупиниться на цій виділеній команді. Входимо в цю команду натисканням на кнопку F7. Натискуємо ще раз кнопку F7 (або F8) та в полі коментарів зразу видно команди, які відповідають за відповідні повідомлення (рис. 3.27.6).

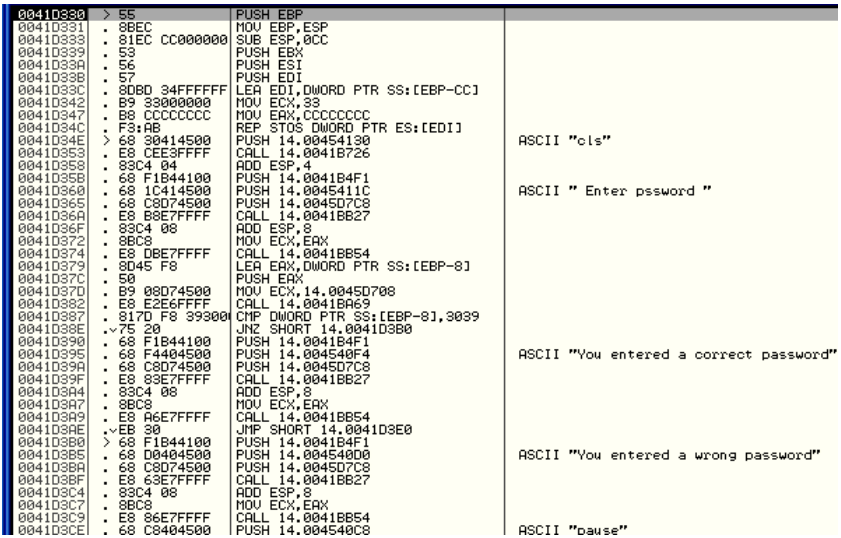


Рис. 3.27.6. Команди, які відповідають за відповідні повідомлення

Через декілька кроків буде порівняння введеного коду із заданим. Це – команда переходу JNZ. Робимо подвійний клік на ній. З'являється вікно (рис. 3.27.7). Необхідно запам'ятати код цієї команди – 75 20.

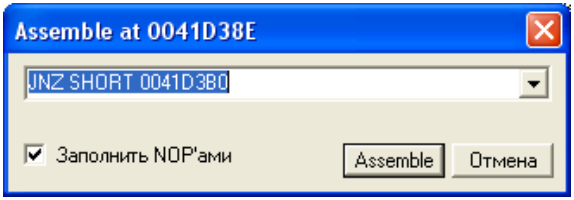


Рис. 3.27.7. Вікно з командою, яку виділили

Замінюємо цю команду на команду безумовного переходу JMP та перевіряємо адресу – чи правильно буде перенесено виконання програми (рис. 3.27.8). Якщо необхідно, то змінюємо й адресу.

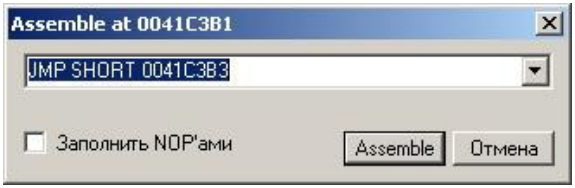


Рис. 27.8. Нові команда та адреса переходу

Тепер треба „зашити" цей код у програму. Для цього скористаємося HEX-редактором XVI32, в який завантажуємо exe-файл (рис. 3.27.9).

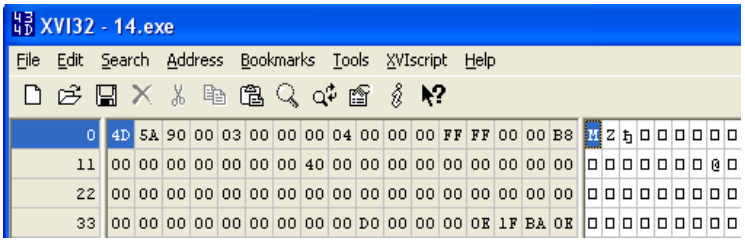


Рис. 3.27.9. Програма в Hex-редакторі

Викликаємо пункт меню Search і шукаємо **75 20.** Замінюємо знайдений рядок на інший. Для цього натискаємо на кнопочку  та змінюємо рядок (рис. 3.27.10).

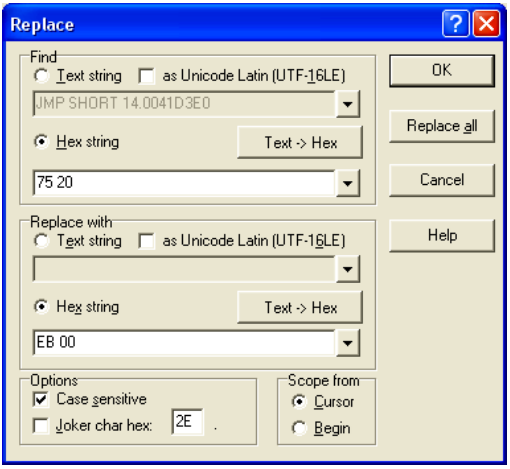


Рис. 3.27.10. Внесення коду для заміни в Hex-редакторі

Зберігаємо зміни, закриваємо редактор. Ця програма прийматиме будь-який ключ.

СПИСОК ЛІТЕРАТУРИ

1. Кравець В. О. Системне програмування. Асемблер під Win32 API : навч. посіб. / В. О. Кравець, О. М. Рисований. – Х. : НТУ "ХПІ", 2008. – 512 с.

2. Крупник А. Асемблер. Самоучитель / А. Крупник. – СПб. : Питер, 2005. – 235 с.

3. Магда Ю. С. Ассемблер для процессоров Intel Pentium / Ю. С. Магда. – СПб. : Питер, 2006. – 410 с.

4. Методичні вказівки до виконання та оформлення курсового проекту з курсу «Системне програмування» для студентів денної форми навчання за спеціальностями: 7.091501 «Комп'ютерні системи та мережі», 7.091502 «Системне програмування», 7.091503 «Спеціалізовані комп'ютерні системи» /уклад. О. М. Рисований. – Х. : НТУ «ХПІ», 2010. – 92 с.

5. Рисований О. М. Системне програмування [Текст] : підручник для студентів напряму "Комп'ютерна інженерія" вищих навчальних закладів / О.М. Рисований. – Х. : НТУ "ХПІ", 2010. – 912 с.

6. Рисований О. М. Цифрові пристрої і мікропроцесори. Архітектура і програмне забезпечення : навч. посіб. / О. М. Рисований, М. В. Грушенко. – Х. : ХУПС, 2005. – 384 с.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З КУРСУ **«СИСТЕМНЕ ПРОГРАМУВАННЯ»**

для студентів спеціальностей:
7.05010201 «Комп'ютерні системи та мережі»,
7. 05010202 «Системне програмування»,
7. 05010203 «Спеціалізовані комп'ютерні системи»

Укладач РИСОВАНИЙ Олександр Миколайович

Роботу до видання рекомендував *В. Д. Дмитрієнко*
Відповідальний за випуск *Ф.А.Домнін*

Редактор *Л.А. Пустовойтова*

План 2012 р., поз. 177

Підписано до друку25.12.2012. Формат 60х84 1/16. Папір офісний.
Riso-друк. Гарнітура Times. Ум. друк. арк. 12,6. Наклад 50 пр.
Зам. № 72 Ціна договірна.

Видавець і виготовлювач
Видавничий центр НТУ «ХПІ»
61002, Харків, вул. Фрунзе, 21

Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.