

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

СИСТЕМЕ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання циклу лабораторних робіт з дисципліни
“Системне програмування” для студентів базового напрямку 6.050102
“Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № 1 від 29.08.2013 р.

Львів 2014

Системне програмування: методичні вказівки до виконання циклу лабораторних робіт з дисципліни “Системне програмування” для студентів базового напрямку 6.050102 “Комп’ютерна інженерія”/ Укл.: В.С.Мархивка, М.В.Олексів, І.В.Мороз, О.І.Акимишин – Львів: Видавництво Національного університету “Львівська політехніка”, 2014. – 74 с.

Укладачі

Мархивка В.С., ст. викладач
Олексів М. В., асистент, к.т.н.
Мороз І.В., ст. викладач, к.т.н.
Акимишин О.І., доцент, к.т.н.

Рецензент

Дунець Р.Б., д. т. н, професор

Відповідальний за випуск:

Мельник А. О., професор, завідувач кафедри

ЗМІСТ

ВСТУП	4
Лабораторна робота №1. Особливості програмування з використанням 32-розрядного Асемблера.....	5
Лабораторна робота №2. Змішане програмування на мовах С та Асемблер	18
Лабораторна робота №3. Використання математичного співпроцесора.....	32
Лабораторна робота №4. Обчислення елементарних функцій на математичному співпроцесорі.....	42
Лабораторна робота №5. Особливості програмування з використанням функцій Win32 API	52
Лабораторна робота №6. Створення DLL та їх використання при неявному зв'язуванні на мові С.....	58
Лабораторна робота №7. Створення DLL та їх використання при явному зв'язуванні на мові Асемблер.....	67

ВСТУП

Розробка якісного, ефективного та надійного програмного забезпечення для комп'ютерних систем є складним та клопітким завданням. У сучасній інженерії програмного забезпечення виділяються два основні напрямки програмування прикладне та системне. Якщо, в першому випадку більше уваги приділяють розумінню предметної області, спрощенню процесу розробки та наближення мови програмування до термінів предметної області, універсальності та переносимості програмного коду, високій надійності програмного продукту. То, для системного програмування, основними задачами є створення високопродуктивного, компактного, ефективного коду. Створення подібних програм неможливе без належного знання принципів функціонування операційної системи та комп'ютерних засобів. Безпосередня взаємодія прикладного програмного забезпечення з системними компонентами часто дозволяє виконувати поставлені завдання якісніше, надійніше та ефективніше, ніж при використанні високорівневих засобів програмування. Тому системне програмування у цій області набуває особливої ваги, адже саме системні засоби дозволяють створювати продуктивні та надійні програми, направлені на вирішення різних нестандартних завдань. Низькорівневе програмування на мові асемблера забезпечує тонке використання усіх особливостей архітектури комп'ютера, що дозволяє створювати компактний та продуктивний продукт, хоча і за рахунок складності розробки та переносимості коду. Однак, саме на мові низького рівня написані ядра операційних систем, що є основою функціонування сучасних комп'ютерних систем.

Цикл лабораторних робіт укладений відповідно до навчальної програми “Системне програмування” та містить сім робіт. Вони орієнтовані на використання низькорівневого програмування на асемблері мікропроцесорів сімейства Intel x86, так як мікропроцесори цього сімейства домінують на ринку апаратних засобів та використані в більшості сучасних універсальних комп'ютерних систем.

За тематикою лабораторні роботи умовно поділяються за двома взаємопов'язаними напрямками: створення елементарних, компактних та ефективних програм; відпрацювання взаємодії між програмами написаними на різних мовах, функціями операційної системи та динамічними бібліотеками.

Лабораторні роботи орієнтовані на наявну на кафедрі ЕОМ лабораторну базу. Методичні вказівки допоможуть студентам ефективно використовувати відведений на виконання лабораторних робіт аудиторний час.

ЛАБОРАТОРНА РОБОТА №1.

ОСОБЛИВОСТІ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ 32-РОЗРЯДНОГО АСЕМБЛЕРА

Мета: Ознайомитись з програмною моделлю 32-розрядних мікропроцесорів Intel та оволодіти навиками створення програм, використовуючи 32-розрядний Асемблер.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Основою для розробки низькорівневого системного програмного забезпечення є програмна модель комп'ютера, частиною якої є *програмна модель мікропроцесора*. До складу програмної моделі мікропроцесорів Intel сімейства x86 входять 32 регістри в тій чи іншій мірі доступні для використання програмістом. Дані регістри можна розділити на дві великі групи:

- 16 регістрів користувача;
- 16 системних регістрів.

У програмах на мові асемблера регістри використовуються дуже інтенсивно. Більшість регістрів мають певне функціональне призначення.

Регістри користувача

Як випливає з назви, *призначеними для користувача* регістри називаються тому, що програміст може використовувати їх при написанні своїх програм. До цих регістрів відносяться (рис.1.1):

- вісім 32-бітових регістрів, які можуть використовуватися програмістами для зберігання даних і адрес (їх ще називають регістрами загального призначення (РЗП)):
 - **eax/ax/ah/al**;
 - **ebx/bx/bh/bl**;
 - **edx/dx/dh/dl**;
 - **ecx/cx/ch/cl**;
 - **ebp/bp**;
 - **esi/si**;
 - **edi/di**;
 - **esp/sp**.
- шість сегментних регістрів: **cs, ds, ss, es, fs, gs**;

- реєстри управління та стану:
 - реєстр прапорів **eflags/flags**;
 - реєстр показчика команди **eip/ip**.

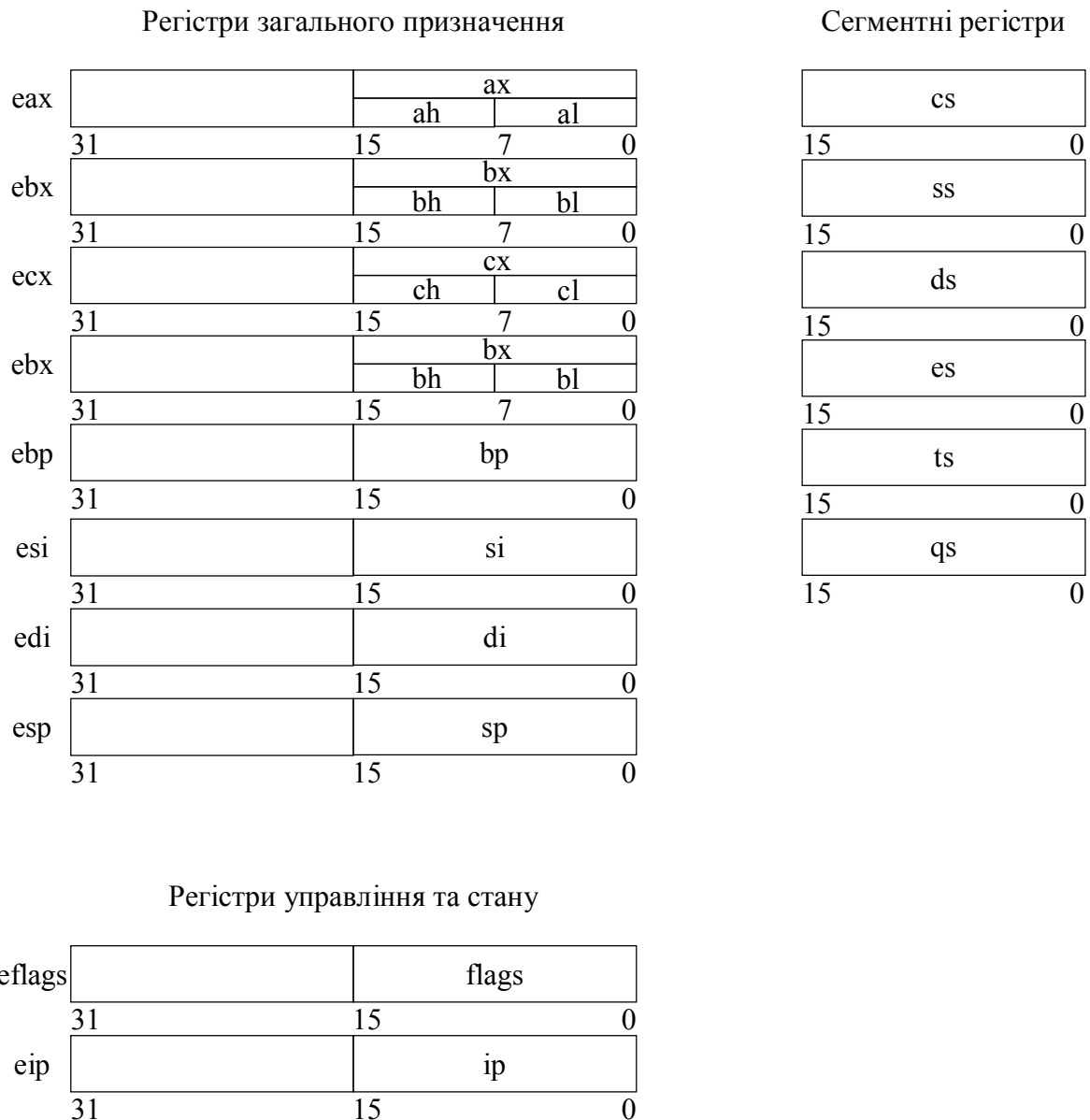


Рис. 1.1. Регістри користувача мікропроцесорів i486 і Pentium

Як видно з рис. 1.1, частина реєстрів зображені розділено. Це не різні реєстри — це частини одного великого 32-розрядного реєстру. Їх можна використовувати в програмі як окремі об'єкти. Так зроблено, для забезпечення сумісності з програмами, написаними для ранніх, 16-розрядних, моделей мікропроцесорів фірми Intel, починаючи з i8086. Мікропроцесори i486 і Pentium мають в основному 32-розрядні реєстри. Їх кількість, за винятком

сегментних реєстрів, така ж, як і у i8086, але розмірність більше, що і відображено в їх позначеннях — вони мають приставку **e** (*Extended*).

Реєстри загального призначення

Всі реєстри цієї групи дозволяють звертатися до своїх “молодших” частин (див. рис. 1.1). Слід відзначити, що використовувати для самостійної адресації можна тільки молодші 16- і 8-бітові частини цих реєстрів. Старші 16 біт цих реєстрів як самостійні об’єкти програмування недоступні. До РЗП відносяться:

- **eax/ax/ah/al** (Accumulator register) — акумулятор. Використовується для зберігання проміжних даних. У деяких командах використання цього реєстра обов’язкове;
- **ebx/bx/bh/bl** (Base register) — базовий реєстр. Зазвичай застосовується для зберігання базової адреси деякого об’єкту в пам’яті;
- **ecx/cx/ch/cl** (Count register) — *реєстр-лічильник*. Застосовується в командах, що проводять деякі дії, що повторюються. Його використання часто неявно і приховано в алгоритмі роботи відповідної команди. Наприклад, команда організації циклу `loop` окрім передачі управління команді, що знаходиться за деякою адресою, аналізує і зменшує на одиницю значення реєстра `ecx/cx`;
- **edx/dx/dh/dl** (Data register) — реєстр *даних*. Так само, як і реєстр `eax/ax/ah/al`, зберігає проміжні дані. У деяких командах його використання обов’язкове, а для деяких це відбувається неявно.

Наступні два реєстри використовуються для підтримки операцій, що проводять послідовну обробку ланцюжків елементів, кожний з яких може мати довжину 32, 16 або 8 біт:

- **esi/si** (Source Index register) — *індекс джерела*.
- **edi/di** (Destination Index register) — *індекс приймача* (одержувача).

В архітектурі мікропроцесора на програмно-апаратному рівні підтримується така структура даних, як *стек*. Для роботи із стеком в системі команд мікропроцесора є спеціальні команди, а в програмній моделі мікропроцесора для цього існують спеціальні реєстри:

- **esp/sp** (Stack Pointer register) — реєстр *покажчика стеку*. Містить покажчик вершини стеку в поточному сегменті стеку.
- **ebp/bp** (Base Pointer register) — реєстр *покажчика бази кадру стеку*. Призначений для організації довільного доступу до даних усередині стеку.

Жорстке закріплення реєстрів для деяких команд дозволяє компактніше кодувати їх машинне подання. Знання цих особливостей дозволяє при необхідності хоч би на декілька байтів заощадити пам’ять, що займається кодом програми.

Робота з масивами

Регістри загального призначення використовуються для адресації масивів. При цьому застосовується адресація за базою з масштабуванням: початкова адреса + база * масштабуючий коефіцієнт бази. Допустимі значення масштабуючого коефіцієнту бази (кількість байтів, які займає 1 елемент масиву) рівні 1, 2, 4, 8.

Таким чином, щоб записати 3-й елемент масиву оголошеного мовою C як `short arr[15]` в `edx` треба написати код:

```
mov eax, arr
mov ebx, 2
mov edx, [eax+ebx*2]
```

Сегментні регістри

У програмній моделі мікропроцесора є шість сегментних регістрів: *cs*, *ss*, *ds*, *es*, *gs* та *fs*. Їх існування обумовлено специфікою організації і використання оперативної пам'яті мікропроцесорами Intel. Вона полягає в тому, що мікропроцесор апаратно підтримує структурну організацію програми у вигляді трьох частин, що називаються сегментами. Відповідно, така організація пам'яті називається **сегментною**.

Для того, щоб вказати на сегменти, до яких програма має доступ в конкретний момент часу, і призначені *сегментні регістри*. Фактично, з невеликою поправкою, як показано далі, в цих регістрах містяться адреси пам'яті з яких починаються відповідні сегменти. Логіка обробки машинної команди побудована так, що при вибірці команди доступу до даних програми або до стеку неявно використовуються адреси з певних сегментних регістрів. Мікропроцесор підтримує наступні типи сегментів:

1. **Сегмент коду**. Містить команди програми. Для доступу до цього сегменту служить регістр **cs** (code segment register) — *сегментний регістр коду*. Він містить адресу сегменту з машинними командами, до якого має доступ мікропроцесор (тобто ці команди завантажуються в конвейєр мікропроцесора).
2. **Сегмент даних**. Містить оброблювані програмою дані. Для доступу до цього сегменту служить регістр **ds** (data segment register) — *сегментний регістр даних*, який зберігає адресу сегменту даних поточної програми.
3. **Сегмент стеку**. Цей сегмент є ділянкою пам'яті, що називається *стеком*. Роботу із стеком мікропроцесор організовує за наступним принципом: останній записаний в цю ділянку елемент вибирається першим. Для доступу до цього сегменту служить регістр **ss** (stack segment register) — *сегментний регістр стеку*, що містить адресу сегменту стеку.

4. **Додатковий сегмент даних.** Неявно алгоритми виконання більшості машинних команд припускають, що оброблювані ними дані розташовані в сегменті даних, адреса якого знаходиться в сегментному реєстрі *ds*. Якщо програмі недостатньо одного сегменту даних, то вона має можливість використовувати ще три додаткові сегменти даних. Але на відміну від основного сегменту даних, адреса якого міститься в сегментному реєстрі *ds*, при використанні додаткових сегментів даних їх адреси потрібно указувати явно за допомогою спеціальних префіксів перевизначення сегментів в команді. Адреси додаткових сегментів даних повинні міститися в реєстрах **es**, **gs**, **fs** (extension data segment registers).

Регістри стану і управління

У мікропроцесор включені декілька реєстрів (див. рис. 1.1), які постійно містять інформацію про стан як самого мікропроцесора, так і програми, команди якої в даний момент завантажені в конвеєр. До цих реєстрів відносяться:

- реєстр прапорів **eflags/flags**;
- реєстр покажчика команди **eip/ip**.

Використовуючи ці реєстри, можна одержувати інформацію про результати виконання команд і впливати на стан самого мікропроцесора. Розглянемо докладніше призначення і вміст цих реєстрів.

eflags/flags (flag register) — реєстр *прапорів*. Розрядність **eflags/flags** — 32/16 бітів. Окремі біти даного реєстра мають певне функціональне призначення і називаються прапорцями. Молодша частина цього реєстра повністю аналогічна реєстру **flags** для i8086. На рис. 1.2 показано вміст реєстра *eflags*.

Виходячи з особливостей використання, прапори реєстра *eflags/flags* можна розділити на три групи:

- *8 прапорців стану*. Ці прапорці можуть змінюватися після виконання машинних команд. Прапорці стану реєстра *eflags* відображають особливості результату виконання арифметичних або логічних операцій. Це дає можливість аналізувати стан обчислювального процесу і реагувати на нього за допомогою команд умовних переходів і викликів підпрограм.
- *1 прапорець управління*. Позначається **df** (Directory Flag). Він знаходиться в 10-му біті реєстру *eflags* і використовується для операцій рядками даних. Значення прапорця *df* визначає напрям поелементної обробки в цих операціях: від початку рядка до кінця або навпаки, від кінця рядка до його початку (*df* = 1). Для роботи з прапорцем *df* існують спеціальні команди: **cld** (зняти прапорець *df*) і **std** (встановити прапорець *df*). Застосування

цих команд дозволяє привести прапорець *df* у відповідність з алгоритмом і забезпечити автоматичне збільшення або зменшення лічильників при виконанні операцій з рядками;

- 5 системних прапорців, для керування вводом/виводом, маскованими перериваннями, відлагодженням, перемиканням між завданнями і віртуальним режимом 8086. Прикладним програмам не рекомендується модифікувати без необхідності ці прапорці, оскільки в більшості випадків це приведе до переривання роботи програми.

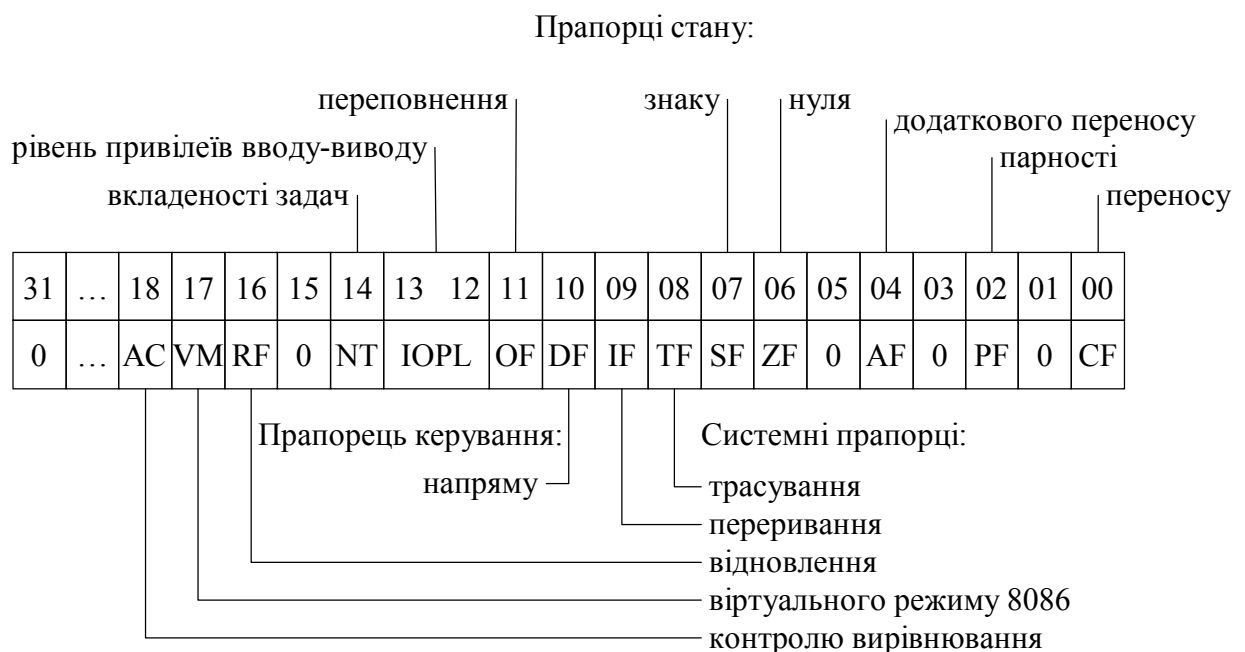


Рис. 1.2. Вміст регістра eflags

eip/ip (Instruction Pointer register) — регістр-показчик команд. Регістр *eip/ip* має розрядність 32/16 бітів і містить зміщення до наступної команди, що має виконуватись, відносно адреси записаної в сегментний регістр *cs*. Цей регістр безпосередньо недоступний програмісту, але завантаження і зміна його значення проводяться різними командами управління, до яких відносяться команди умовних і безумовних переходів, виклику і повернення з процедур. Виникнення переривань також приводить до модифікації регістра *eip/ip*.

Системні регістри мікропроцесора

Сама назва цих регістрів говорить про те, що вони виконують специфічні функції в системі. Використання системних регістрів жорстко регламентовано. Саме вони забезпечують роботу захищеного режиму. Їх також можна розглядати як частина архітектури мікропроцесора, яка навмисно залишена

видимою для того, щоб кваліфікований системний програміст міг виконати самі низькорівневі операції. Системні регістри можна розділити на три групи:

- чотири регістри управління;
- чотири регістри системних адрес;
- вісім регістрів відлагодження.

Регістри управління

До групи регістрів управління входять 4 регістри: **cr0**, **cr1**, **cr2**, **cr3**.

Ці регістри призначені для загального управління системою. Регістри управління доступні тільки програмам з рівнем привілеїв 0.

Хоча мікропроцесор має чотири регістри управління, доступними є тільки три з них — виключаючи **cr1**, функції якого поки не визначені (він зарезервований для майбутнього використання).

Регістр **cr0** містить *системні прапорці*, для керування режимами роботи мікропроцесора, що відображають його стан глобально, незалежно від конкретних завдань, що виконуються.

Призначення системних прапорців:

- **pe** (Protect Enable), біт 0 — *дозвіл захищеного режиму роботи*. Стан цього прапорця показує, в якому з двох режимів — реальному ($pe = 0$) або захищеному ($pe = 1$) — працює мікропроцесор в даний момент часу.
- **mp** (Math Present), біт 1 — *наявність співпроцесора*. Завжди 1.
- **ts** (Task Switched), біт 3 — *перемикання завдань*. Процесор автоматично встановлює цей біт при перемиканні на виконання іншого завдання.
- **am** (Aligment Mask), біт 18 — *маска вирівнювання*. Цей біт дозволяє ($am = 1$) або забороняє ($am = 0$) контроль вирівнювання.
- **cd** (Cache Disable), біт 30, — *заборона кеш-пам'яті*. За допомогою цього біта можна заборонити ($cd = 1$) або дозволити ($cd = 0$) використання внутрішньої кеш-пам'яті (кеш-пам'яті першого рівня).
- **pg** (PaGing), біт 31, — *дозвіл* ($pg = 1$) або *заборона* ($pg = 0$) *сторінкового перетворення*.

Регістр **cr2** використовується при сторінковій організації оперативної пам'яті для реєстрації ситуації, коли поточна команда звернулася за адресою, що міститься в сторінці пам'яті, відсутній в даний момент часу в пам'яті.

У такій ситуації в мікропроцесорі виникає виняткова ситуація з номером 14, і лінійна 32-бітова адреса команди, що викликала цей виняток, записується в регістр **cr2**. Маючи цю інформацію, обробник винятку 14 визначає потрібну сторінку, здійснює її підкачку в пам'ять і відновлює нормальну роботу програми.

Регістр **cr3** також використовується при сторінковій організації пам'яті. Це так званий *регістр каталогу сторінок першого рівня*. Він містить 20-бітову фізичну базову адресу каталогу сторінок поточного завдання. Цей каталог містить 1024 32-бітових дескриптора, кожний з яких містить адресу таблиці сторінок другого рівня. У свою чергу кожна з таблиць сторінок другого рівня містить 1024 32-бітових дескриптора, що адресують сторінкові кадри в пам'яті. Розмір сторінкового кадру — 4 Кбайт.

Регістри системних адрес

Ці регістри ще називають *регістрами управління пам'яттю*. Вони призначені для захисту програм і даних в мультизадачному режимі роботи мікропроцесора. При роботі в захищеному режимі мікропроцесора адресний простір ділиться на:

- *глобальний* — загальний для всіх завдань;
- *локальний* — окремий для кожного завдання.

Цим розділенням і пояснюється присутність в архітектурі мікропроцесора наступних системних регістрів:

- *регістра таблиці глобальних дескрипторів **gdt*** (Global Descriptor Table Register) що має розмір 48 біт і що містить 32-бітову (біти 16—47) базову адресу глобальної дескрипторної таблиці GDT і 16-бітове (біти 0—15) значення межі, що є розміром в байтах таблиці GDT;
- *регістра таблиці локальних дескрипторів **ldtr*** (Local Descriptor Table Register) що має розмір 16 біт і що містить так званий селектор дескриптора локальної дескрипторної таблиці LDT. Цей селектор є покажчиком в таблиці GDT, який і описує сегмент, що містить локальну дескрипторну таблицю LDT;
- *регістра таблиці дескрипторів переривань **idt*** (Interrupt Descriptor Table Register) що має розмір 48 біт і що містить 32-бітову (біти 16—47) базову адресу дескрипторної таблиці переривань IDT і 16-бітове (біти 0—15) значення межі, що є розміром в байтах таблиці IDT;
- *16-бітового регістра завдання **tr*** (Task Register), який подібно до регістру **ldtr**, містить селектор, тобто покажчик на дескриптор в таблиці GDT. Цей дескриптор описує *поточний сегмент стану завдання* (TSS — Task Segment Status). Цей сегмент створюється для кожного завдання в системі, має жорстко регламентовану структуру і містить контекст (поточний стан) завдання. Основне призначення сегментів TSS — зберігати поточний стан завдання у момент перемикання на інше завдання.

Регістри відлагодження

Це дуже цікава група регістрів, призначених для апаратного відлагодження. Засоби апаратного відлагодження вперше з'явилися в мікропроцесорі i486. Мікропроцесор містить вісім регістрів відлагодження, але реально з них використовуються тільки 6.

Регістри **dr0**, **dr1**, **dr2**, **dr3** мають розрядність 32 біти і призначені для завдання лінійних адрес чотирьох точок переривання. Регістр **dr6** називається регістром стану відлагодження. Біти цього регістра встановлюються відповідно до причин, які викликали виникнення останнього винятку з номером 1. Перерахуємо ці біти і їх призначення:

- **b0** — якщо цей біт встановлений в 1, то останній виняток (переривання) виникло в результаті досягнення контрольної крапки, визначеної в регістрі *dr0*;
- **b1** — аналогічно *b0*, але для контрольної крапки в регістрі *dr1*;
- **b2** — аналогічно *b0*, але для контрольної крапки в регістрі *dr2*;
- **b3** — аналогічно *b0*, але для контрольної крапки в регістрі *dr3*;
- **bd** (біт 13) — служить для захисту регістрів відлагодження;
- **bs** (біт 14) — встановлюється в 1, якщо виключення 1 було викликано станом прапорця *tf* = 1 в регістрі *eflags*;
- **bt** (біт 15) встановлюється в 1, якщо виключення 1 було викликано перемиканням на завдання зі встановленим бітом пастки в TSS *t* = 1.

Решта всіх бітів в цьому регістрі заповнюється нулями. Обробник винятку 1 за вмістом *dr6* повинен визначити причину, після якої відбувся виняток, і виконати необхідні дії.

Регістр *dr7* називається регістром управління відлагодженням. У ньому для кожного з чотирьох регістрів контрольних точок відлагодження є поля, за допомогою яких можна уточнити наступні умови, при яких слід згенерувати переривання:

- *місце реєстрації контрольної крапки* — тільки в поточному завданні або в будь-якому завданні. Ці біти займають молодші вісім біт регістра *dr7* (по два біта на кожен контрольну крапку (фактично точку переривання), що задається регістрами *dr0*, *dr1*, *dr2*, *dr3* відповідно). Перший біт з кожної пари — це так званий локальний дозвіл; його установка говорить про те, що точка переривання діє якщо вона знаходиться в межах адресного простору поточного завдання. Другий біт в кожній парі визначає глобальний дозвіл, який говорить про те, що дана контрольна крапка діє в межах адресних просторів всіх завдань, що знаходяться в системі;

- *тип доступу*, за яким ініціюється переривання: тільки при вибірці команди, при записі або при записі/читанні даних. Біти, що визначають подібну природу виникнення переривання, локалізуються в старшій частині даного регістра.

КОНТРОЛЬНІ ПИТАННЯ

1. Які програмно доступні регістри архітектури IA-32 ви знаєте?
2. У чому полягає різниця між програмними моделями архітектур x86 і IA-32?
3. Як написати програму використовуючи MASM 32?
4. Як відлагодити програму використовуючи MASM 32?
5. Принципи роботи з масивами даних в Асемблері.

ЛІТЕРАТУРА

1. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC XT и AT. - М. "Финансы и статистика", 1992, стор. 13-31.
2. Березко Л.О., Троценко В.В. Особливості програмування в турбо-асемблері. - Київ, НМК ВО, 1992.
3. Дао Л. Программирование микропроцессора 8088. Пер. с англ. - М.: "Мир", 1988.
4. Абель П. Язык ассемблера для IBM PC и программирования. Пер. з англ. - М., "Высшая школа", 1992.

ЗАВДАННЯ

1. Створити, використовуючи мову асемблера мікропроцесорів сімейства x86 Intel, *.exe програму, яка реалізовує обчислення, заданого варіантом виразу. $A = \{a[i]\}$ – наперед заданий масив з N чисел цілих чисел. c, d – цілі константи. K, L – цілі додатні числа.
2. Переконатися у правильності роботи програми використовуючи VKDebug.
3. Скласти звіт про виконану роботу з приведенням тексту програми.
4. Дати відповідь на контрольні запитання.

Таблиця 1.1.

Варіанти завдань

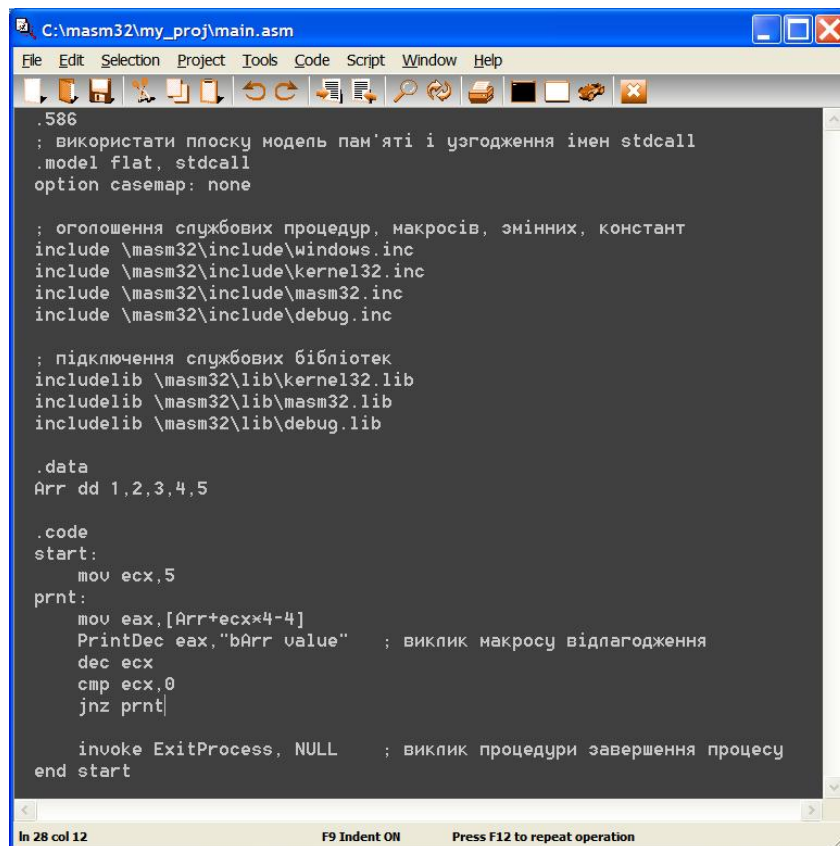
№	Завдання
1.	Знайти число елементів A за умови $c \leq a[i] \leq d$
2.	Знайти число додатних непарних елементів A

3.	Знайти число додатних елементів у A
4.	Знайти в A число від'ємних елементів з непарними індексами
5.	Знайти число парних від'ємних елементів A за умови $c-d \leq a[i] \leq c+d$
6.	Знайти число додатних елементів A за умови $c \leq a[i] \leq d$
7.	Знайти суму всіх додатних елементів A за умови $a[i] \geq d/c $
8.	Знайти суму всіх додатних елементів A
9.	Знайти число елементів з непарними індексами A за умови $c < a[i] \leq d$
10.	Знайти суму перших K елементів A , за умови $a[i] \geq c+d$
11.	Знайти суму перших K від'ємних елементів A за умови $a[i] \geq c+d$
12.	Знайти суму перших K додатних елементів A за умови $a[i] \geq c+d$
13.	Знайти число від'ємних і нульових елементів в A за умови $c \leq a[i] < d$
14.	Знайти число нульових елементів A
15.	Знайти суму всіх від'ємних елементів A
16.	Знайти суму останніх L елементів A .
17.	Знайти суму перших K від'ємних елементів A за умови $c \leq a[i] \leq d$
18.	Знайти добуток останніх L елементів A за умови $c \leq a[i] \leq d$
19.	Знайти добуток останніх L додатних елементів A за умови $c < a[i] < d$
20.	Знайти суму всіх непарних елементів A
21.	Знайти суму перших K елементів A за умови $c \leq a[i] \leq d$
22.	Знайти суму перших K елементів A за умови $2*c \leq a[i] \leq 3*d$
23.	Знайти суму елементів A за умови: $c \leq a[i] \leq d$
24.	Знайти суму останніх L додатних елементів A
25.	Знайти суму останніх L від'ємних елементів A
26.	Знайти суму всіх парних елементів A
27.	Знайти скільки ненульових елементів A
28.	Знайти суму елементів в масиві A за умови $c \leq a[i] \leq 4*d$
30.	Знайти суму елементів масиву A за умови $a[i] \geq c/d$

ПОРЯДОК ВИКОНАННЯ

1. Завантажити з <http://www.masm32.com/masmdl.htm> або взяти у керівника лабораторних робіт і встановити макроасемблер MASM32.
2. Запустити середовище розробки MASM32. Для цього потрібно запустити на виконання файл qeditor.exe з каталогу, куди встановлено MASM32.
3. Створити новий файл. Для цього в закладці File вибрати New.
4. Написати програму вносячи виклики відлагоджувальних макросів VKDebug (рис. 1.3). Перелік відлагоджувальних макросів та методів їх використання знаходиться у вкладці MASM32: Help->VKDebug Help.

5. Зберегти програму.
6. Збудувати програму. Для цього запустити: Project -> Build All.
Увага!!! Перед кожним будуванням програми її необхідно зберегти вручну.



```
.586
; використати плоску модель пам'яті і узгодження імен stdcall
.model flat, stdcall
option casemap: none

; оголошення службових процедур, макросів, змінних, констант
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
include \masm32\include\debug.inc

; підключення службових бібліотек
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\debug.lib

.data
Arr dd 1,2,3,4,5

.code
start:
    mov ecx,5
prnt:
    mov eax,[Arr+ecx*4]
    PrintDec eax,"bArr value"    ; виклик макросу відлагодження
    dec ecx
    cmp ecx,0
    jnz prnt

    invoke ExitProcess, NULL    ; виклик процедури завершення процесу
end start
```

Рис.1.3. середовище розробки MASM32 з написаною програмою.

7. Запустити програму і переконатися в правильності її виконання. Для цього необхідно запустити: File -> Run Program. Вибрати файл з розширенням .exe для запуску. Після запуску програми відкриється вікно відлагоджувача, де буде відображено відлагоджувальну інформацію. В цьому вікні обов'язково має бути відображеним результат виконання програми (рис. 1.4).

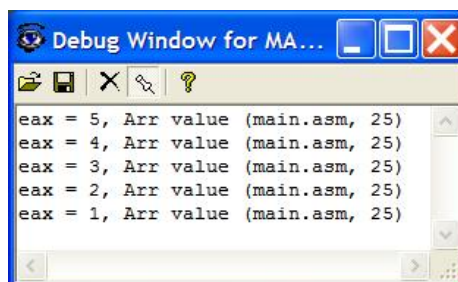


Рис.1.4. Вікно відлагоджувача.

ПРИКЛАД ПРОГРАМИ

```
; Програма виводить в зворотному порядку за допомогою
; відлагоджувача масив 4 байтних значень
.586
; використати плоску модель пам'яті і узгодження імен stdcall
.model flat, stdcall
option casemap: none

; оголошення службових процедур, макросів, змінних, констант
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
include \masm32\include\debug.inc

; підключення службових бібліотек
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\debug.lib

; оголошення масиву даних
.data
Arr dd 1,2,3,4,5

.code
start:
    mov ecx,5
prnt:
    mov eax, [Arr+ecx*4-4]
    PrintDec eax, "Arr value" ; виклик макросу відлагодження
    dec ecx
    cmp ecx, 0
    jnz prnt
    invoke ExitProcess, NULL ; виклик процедури завершення процесу
end start
```

ЛАБОРАТОРНА РОБОТА №2.

ЗМІШАНЕ ПРОГРАМУВАННЯ НА МОВАХ С ТА АСЕМБЛЕР

Мета: оволодіти навиками створення програм, частини яких написані різними мовами програмування. Засвоїти правила взаємодії між прогнаними модулями різних.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Труднощі опису зв'язку програм написаних мовою С і асемблерних програм полягає в тому, що різні версії мови С мають різні угоди з виклику функцій та передавання їм параметрів. Для більш точної інформації варто користатися посібником з наявної версії мови С. До особливостей угод щодо виклику функцій та передавання їм параметрів можна віднести:

- більшість версій мови С забезпечують передачу параметрів через стек у зворотній (у порівнянні з іншими мовами) послідовності. Зазвичай доступ, наприклад, до двох параметрів, переданих через стек, здійснюється в такий спосіб:

```
PUSH EBP
MOV EBP,ESP
MOV EAX,[EBP+8]
MOV EDX,[EBP+12]
; тіло підпрограми ...
POP EBP
RET
```

- у мові С розрізняють великі і малі літери, тому ім'я асемблерного модуля повинне бути представлено в тому ж символічному реєстрі, який використовують для посилання С-програми;
- у деяких версіях мови С потрібно, щоб асемблерні програми, що змінюють реєстри EDI і ESI, записували їхній вміст у стек при вході і відновлювали ці значення зі стеку при виході;
- асемблерні програми повинні повертати значення, якщо це необхідно, у реєстрі EAX (подвійне слово) чи в реєстровій парі EDX:EAX (8 слів);
- для деяких версій мови С, якщо асемблерна програма встановлює прапор DF, то вона повинна скинути його командою CLD перед поверненням.

Щоб скомпонувати разом модулі С++ і Макро асемблера, повинні бути дотримані наступні три умови:

1. У модулях Макро Асемблера повинні використовуватися угоди про імена, прийняті в C++.
2. C++ і Макро Асемблер повинні спільно використовувати відповідні функції й імена змінних у формі, прийнятної для C++.
3. Для комбінування модулів у виконувану програму потрібно використовувати утіліту-компоновщик (TLINK, LINK тощо).

Підкреслення і мова С

Для забезпечення взаємодії програми написаної асемблером з програмами написаними мовами С чи C++, всі зовнішні мітки повинні починатися із символу підкреслення (_). Компілятор С і C++ вставляє символи підкреслення перед всіма іменами зовнішніх функцій і змінних при трансляції в об'єктний код автоматично, тому в асемблерних програмах перший символ підкреслення необхідно вставити самостійно. Тому, ідентифікатори глобальних змінних та назви функцій які використовуються в асемблерній програмі для взаємодії з С модулями, чи навпаки, повинні починатися із символу підкреслення.

Наприклад, наступна програма мовою С (link2asm.cpp):

```
extrn int ToggleFlag();
int Flag;
main()
{
    ToggleFlag();
}
```

правильно компонується з наступною програмою на асемблері (casmlink.asm):

```
.586
.MODEL FLAT
.DATA
    EXTRN _Flag:dword
.CODE
    PUBLIC _ToggleFlag
_ToggleFlag PROC
    cmp [_Flag], 0      ; прапорець скинутий?
    jz SetFlag          ; так, установити його
    mov [_Flag], 0      ; ні, скинути його
    jmp EndToggleFlag   ; виконано
SetFlag:
    mov [_Flag], 1      ; встановити прапорець
EndToggleFlag:
    ret
_ToggleFlag ENDP
END
```

При використанні в директивах EXTERN і PUBLIC специфікатора мови С правильно компонується з наступною програмою на Асемблері (cspec.asm) (приклад для 16-ти бітної програми):

```
.MODEL Small
.DATA
    EXTRN C Flag:word
.CODE
    PUBLIC C ToggleFlag
ToggleFlag PROC
    cmp [Flag], 0           ; прапорець скинутий?
    jz SetFlag             ; так, установити його
    mov [Flag], 0          ; ні, скинути його
    jmp short EndToggleFlag ; виконано
SetFlag:
    mov [Flag], 1          ; установити прапорець
EndToggleFlag:
    ret
ToggleFlag ENDP
END
```

Примітка: Мітки, на які відсутні посилання в програмі на С (такі, як SetFlag) не вимагають попередніх символів підкреслення. Турбо Асемблер (використовується тільки для 16-ти бітних програм) автоматично при записі імен Flag і ToggleFlag в об'єктний файл помістить перед ними символ підкреслення.

Розпізнавання рядкових і прописних символів в ідентифікаторах

В іменах ідентифікаторів Макро асемблер звичайно не розрізняє малі та великі літери (верхній і нижній регістр). Оскільки в С вони розрізняються, бажано задати таке розходження й у Макро Асемблері (принаймні для тих ідентифікаторів, що спільно використовуються Асемблером і С). Це можна зробити за допомогою макрокоманди OPTION CASEMAP:NONE.

Типи міток

Хоча в програмах Асемблера можна вільно звертатися до будь-якої змінної чи даних довільного розміру (8, 16, 32 біти і т.д.), у загальному випадку бажано звертатися до змінних відповідно до їхніх розмірів. Наприклад, якщо записати слово в байтову змінну, то зазвичай це приводить до проблем:

```
...
SmallCount DB 0
...
mov WORD PTR [SmallCount],0ffffh
```

...

Тому важливо, щоб в операторі Асемблера EXTRN, у якому описуються змінні С, задавався правильний розмір цих змінних, тому що при генерації розміру доступу до змінного С, Асемблер ґрунтується саме на цих описах.

Якщо в програмі мовою С++ міститься оператор:

```
char c
```

то код Асемблера

...

```
EXTRN    c:WORD
```

...

```
inc c
```

...

може привести до дуже неприємних помилок, оскільки після того, як у кодї мовою С змінна *c* збільшиться чергові 256 разів, її значення буде скинуто, а чарез те, що вона описана в Асемблері, як змінна розміром у слово, то байт за адресою OFFSET *c*+1 буде збільшуватися некоректно, що приведе до непередбачених результатів.

Узгодження типів (С та Assembler)

Між типами даних С та Макро асемблера існує співвідношення, що наведене в табл. 2.1.

Таблиця 2.1.

Узгодження між типами С і Assembler

Тип даних С++	Тип даних Асемблера
unsigned char	byte
char	byte
enum	dword
unsigned short	dword
short	dword
unsigned int	dword
int	dword
unsigned long	dword
long	dword
float	dword
double	qword
long double	tbyte
near*	dword

Передача параметрів

У мові C функціям параметри передаються через стек. Перед викликом функції компілятор C спочатку заносить у стек передані цієї функції параметри, починаючи із самого правого параметра і закінчуючи лівим. У C виклик функції:

```
...  
Test(i, j, 1);  
...
```

компілюється в інструкції:

```
mov    eax,1  
push   eax  
push   dword ptr _j  
push   dword ptr _i  
call   _Test  
add    esp,12
```

де видно, що правий параметр (значення 1), заноситься в стек першим, потім туди заноситься параметр *j* та, нарешті, *i*.

При поверненні з функції занесені в стек параметри усе ще знаходяться там, але вони більше не використовуються. Тому безпосередньо після кожного виклику функції C прокручує вказівник стеку назад у відповідності зі значенням, що він мав перед занесенням у стек параметрів (параметри, таким чином, відкидаються). У попередньому прикладі три параметри (по два байти кожен) займають у стеку разом 12 байт, тому компілятор C додає значення 12 до вказівника стеку, щоб відкинути параметри після звертання до функції `Test`. Важливий момент полягає в тому, що відповідно до використовуваних за замовчуванням угод C/C++ за видалення параметрів зі стеку відповідає викликаюча програма.

Функції Асемблера можуть звертатися до параметрів, переданих через стек, з використанням регістру EBP. Наприклад, функція `Test` може мати таку реалізацію на мові Асемблера (`prmstack.asm`):

```
.586  
.MODEL FLAT  
.CODE  
PUBLIC _Test  
_Test PROC  
    push ebp  
    mov  ebp,esp  
    mov  eax,[ebp+8] ; одержати параметр 1  
    add  eax,[ebp+12] ; додати параметр 2 до параметра 1  
    sub  eax,[ebp+16] ; відняти від суми параметр 3
```

```

    pop    ebp
    ret
_Test ENDP

```

Функція `Test` одержує доступ до переданих з програми мовою C параметрів через стек, з використанням регістру `EBP`. (`EBP`, це регістр для базової адресації за замовчуванням).

На рис. 2.1 показано, як виглядає стек перед виконанням першої інструкції у функції `Test`.

```

i = 25;
j = 4;
Test(1, j, 1);

```

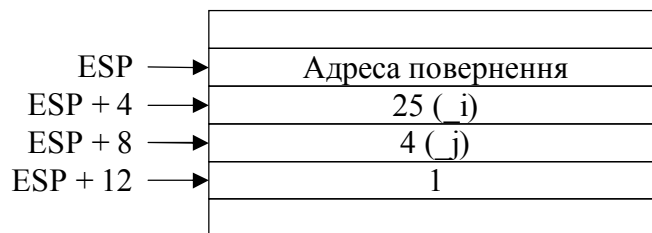


Рис. 2.1. Стан стеку перед виконанням першої інструкції функції `Test`

Параметри функції `Test` мають фіксовані адреси відносно `ESP`, починаючи з комірки, на 4 байти старшої від адреси, за якою зберігається адреса повернення, що занесена туди при виклику. Завантаження регістра `EBP` значенням `ESP` дає можливість звертатися до цих параметрів відносно `EBP`. Однак, спочатку необхідно зберегти у стеку значення `EBP` для того, щоб забезпечити його відновлення при поверненні з функції, тому що викликаюча програма передбачає, що при поверненні `EBP` змінений не буде. Занесення в стек `EBP` змінює всі зміщення в стеку. На рис. 2.2 показано стан стеку після виконання наступних рядків коду:

```

...
push    ebp
mov     ebp, esp
...

```

Організація передачі параметрів функції через стек і використання його для динамічних локальних змінних - це стандартний прийом для мови C. Як можна помітити, неважливо, скільки параметрів має програма мовою C: крайній лівий параметр завжди зберігається в стеку за адресою, що безпосередньо слідує за збереженою у стеку адресою повернення, наступний параметр, що передається, зберігається безпосередньо після крайнього лівого параметра і т.д. Оскільки порядок і тип переданих параметрів відомі, їх завжди можна знайти в стеку.

Простір для локальних змінних можна зарезервувати, віднімаючи від ESP необхідну кількість байт. Наприклад, простір для локального масиву розміром у 100 байт можна зарезервувати, якщо почати функцію `Test` з інструкцій:

```
...
push  ebp
mov    ebp, esp
sub    esp, 100
...
```

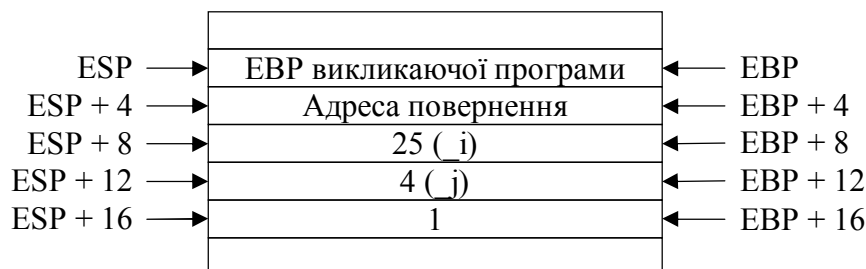


Рис. 2.2 Стан стеку після інструкцій PUSH і MOVE

Тоді, до локальних даних у зарезервованому адресному провторі в стеку можна і далі звертатися через регістр EBP, але уже з від’ємним зміщення. При виході з підпрограми також необхідно відкоректувати вказівник стеку.

Повернення значень

Програми, які викликаються з C і написані на Асемблері можуть повертати значення. Значення функцій повертаються через напередвизначені регістри, що відображені у табл. 2.2.

Таблиця 2.2.

Розташування резултратів виконання функцій

Тип значення, що повертається функцією	Розміщення значення, що повертається функцією
unsigned char	EAX
char	EAX
enum	EAX
unsigned short	EAX
short	EAX
unsigned int	EAX
int	EAX
unsigned long	EAX
long	EAX

float	ST(0), вершина стеку співпроцесора 8087
double	ST(0), вершина стеку співпроцесора 8087
long double	ST(0), вершина стеку співпроцесора 8087
near*	EAX

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке змішане програмування?
2. Яких узгоджень необхідно дотримуватися при змішаному програмуванні на C/Асемблер?
3. Як передаються параметри функціям, що написані на мові C?
4. Як передаються результати з функцій, що написані на мові C?
5. Хто має очищати стек після виклику функції, що написана на мові C?
6. Як передати параметри з Асемблера в функцію, що написана на мові C?
7. Особливості взаємодії C-Асемблер.
8. Особливості взаємодії C-Асемблер-C.

ЛІТЕРАТУРА

1. Джордейн Р.Справочник программиста персональных компьютеров типа IBM PC XT и AT. - М."Финансы и статистика",1992,стор.13-31.
- 2.Березко Л.О., Троценко В.В. Особливості програмування в турбо-асемблері. - Київ, НМК ВО, 1992.
- 3.Дао Л. Программирование микропроцессора 8088.Пер. с англ. -М.: "Мир",1988.
- 4.Абель П.Язык ассемблера для IBM PC и программирования. Пер. з англ.- М., "Высшая школа",1992.

ЗАВДАННЯ

1. Створити дві програми. Прша програма реалізує взаємовиклики **C – ASM** та здійснює обчислення, заданого виразу, згідно варіанту (Табл. 2.3). Програма повинна складатися з кількох модулів, передача параметрів між якими здійснюється через стек. Константа передається через спільну пам'ять.

Основний модуль – створюється мовою C. Він повинен забезпечувати:

- ввід даних з клавіатури;
- виклик підпрограми обчислення виразу;
- вивід на екран результату обчислення виразу.

Модуль безпосередніх обчислень – здійснює всі обчислення виразу. Створюється мовою Assembler.

2. Друга програма реалізує взаємовиклики **C – ASM – C** та здійснює обчислення, заданого виразу, згідно варіанту. Програма повинна складатися з кількох модулів, передача параметрів між якими здійснюється через стек.

Основний модуль – створюється мовою C. Він повинен забезпечувати:

- ввід даних з клавіатури;
- виклик підпрограми обчислення виразу;

Модуль безпосередніх обчислень – здійснює всі обчислення і вивід на екран результату обчислення виразу викликом стандартної функції **printf()**. Створюється мовою Assembler.

3. Відлагодити та протестувати програми. Результати роботи програм продемонструвати викладачу.
4. Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї, а також результатів її роботи.
5. Дати відповідь на контрольні запитання.

Таблиця 2.3.

Варіанти завдань

№	Вираз	К
1.	$X = A_2 + C_1 - D_2 / 2 + K$	1254021
2.	$X = A_4 + C_2 + D_1 * 4 - K$	202
3.	$X = K - B_2 * 5 + C_2 - E_1$	37788663
4.	$X = A_4 + C_1 - D_4 / 5 + K$	45694
5.	$X = B_4 - A_2 * 2 - E_2 + K$	505
6.	$X = K + B_2 / 4 - D_2 * 4 - E_1$	6DD02316
7.	$X = A_4 / 2 - 4 * (D_1 + E_2 - K)$	717
8.	$X = A_4 - B_2 + K - D_2 / 2 + 8 * B_2$	88
9.	$X = 4 * B_2 - C_2 + D_4 / 4$	29
10.	$X = A_4 - B_4 / 2 + K + E_2 * 4$	2310
11.	$X = (A_4 - B_2 - K) * 2 + E_4 / 4$	311
12.	$X = K + B_4 / 2 - 4 * F_2 - E_1$	7055E0AC
13.	$X = A_2 / 2 + 8 * (D_1 + E_2 - K)$	2513
14.	$X = A_4 - B_1 - K - D_2 / 2 + 4 * B_1$	614
15.	$X = A_3 + (4 * C_2) - D_4 / 2 + K$	4569600F
16.	$X = A_4 / 4 + C_2 - D_1 * 2 + K$	616
17.	$X = A_4 - K + C_4 / 2 - E_1 * 8$	1017
18.	$X = 4 * (B_2 - C_1) + D_2 / 4 + K$	56987018

19.	$X=A_2*4+C_1-D_4/2+K$	4019
20.	$X=K+B_4/4-D_1*2-E_2$	18932020
21.	$X=A_4/8+2*(D_2-E_1+K)$	21
22.	$X=K-B_1-C_1-D_2/2+4*B_1$	45781022
23.	$X=A_2*8-C_1+D_4/2+K$	7AA02023
24.	$X=K-B_2/2+D_3+E_2*4$	74569024
25.	$X=(K-B_2-C_1)*2+E_4/4_2$	2B05025
26.	$X=A_2+K+C_2/2-E_1*8$	6C26
27.	$X=A_2*4+(K-E_1*4)$	A77627
28.	$X=K+B_4/2+D_2-E_2/4$	3FF28
29.	$X=K-B_1*4+D_2-F_2/2$	12A0C029
30.	$X=K+B_4-D_2/2+E_1*4$	25630

A, B, C, D, E, F - знакові цілі числа, довжиною в байтах, згідно з індексом, значення константи K подано у 16-му форматі.

ПОРЯДОК ВИКОНАННЯ

1. Запустити середовище розробки Microsoft Visual Studio 2005, для чого потрібно виконати: Start-> All programs->Microsoft Visual Studio 2005 -> Microsoft Visual Studio 2005.

2. Створити новий проект:

2.1. У закладці File вибрати New - > Project (Рис. 2.3).

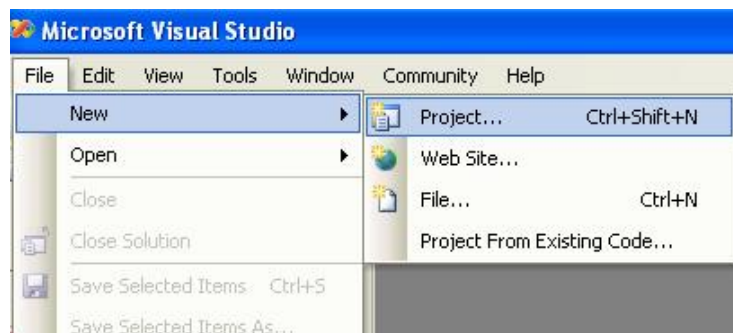


Рис. 2.3. Створення нового проекту в середовищі Microsoft Visual Studio 2005.

2.2. Вибрати тип проекту Visual C++ та шаблон Win32 Console Application (Рис. 2.4) та натиснути кнопку «ОК».

У вікні, що з'явиться натиснути кнопку «NEXT» (Рис. 2.5).

У додаткових опціях проекту встановити тип «Empty project» (Рис. 2.6) та натиснути «Finish».

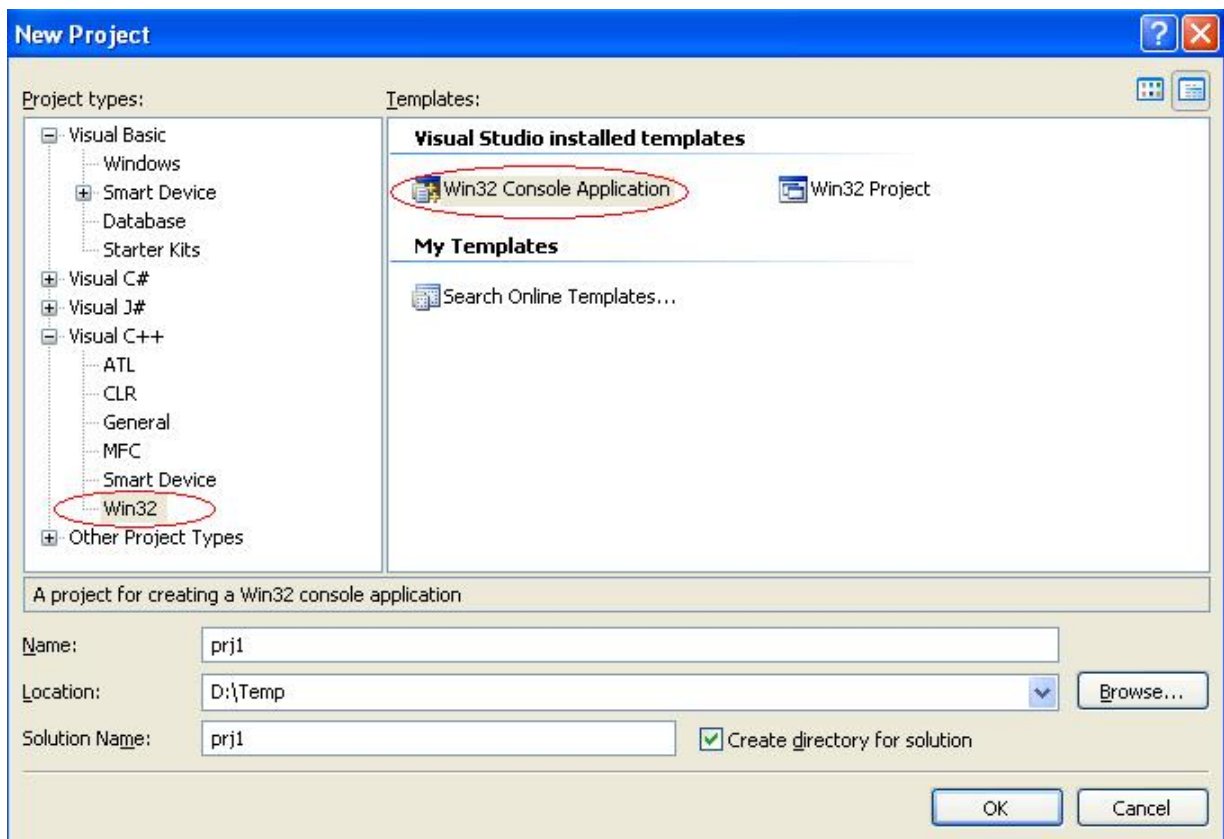


Рис. 2.4. Вибір типу проекту.

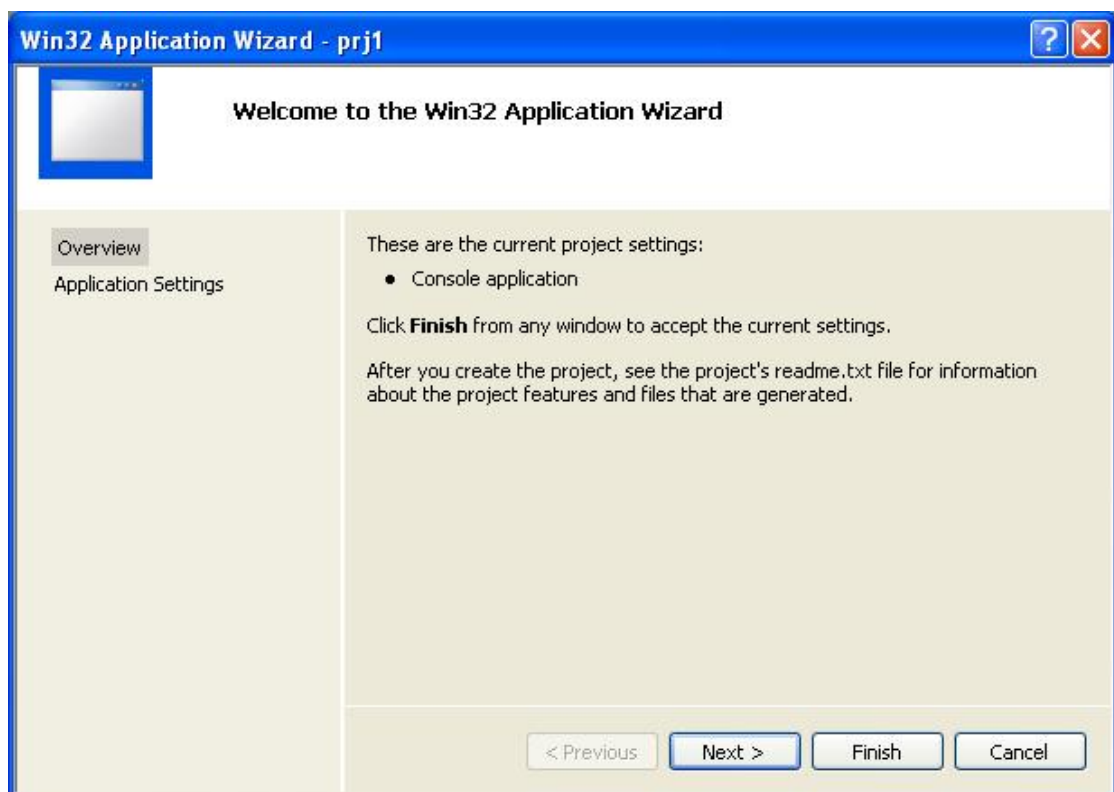


Рис. 2.5. Інформаційне вікно, що вказує тип вибраного проекту.

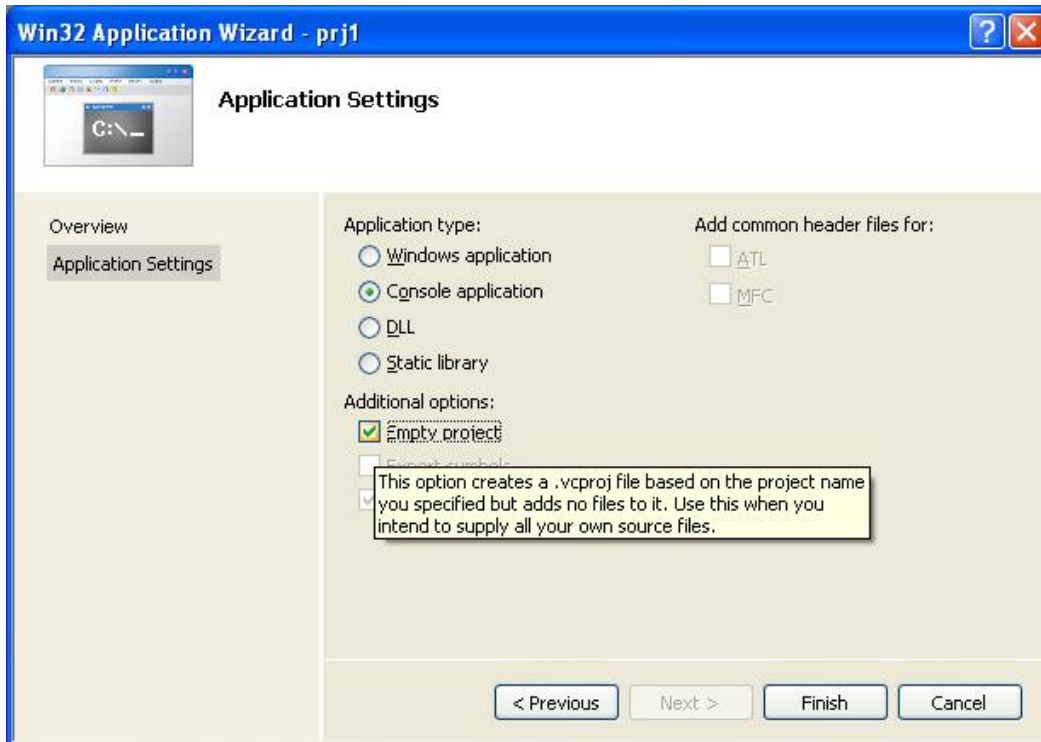


Рис. 2.6. Встановлення параметрів проекту.

У вікні менеджера проекту буде відображатись структура створеного проекту, зображена на рис. 2.7.

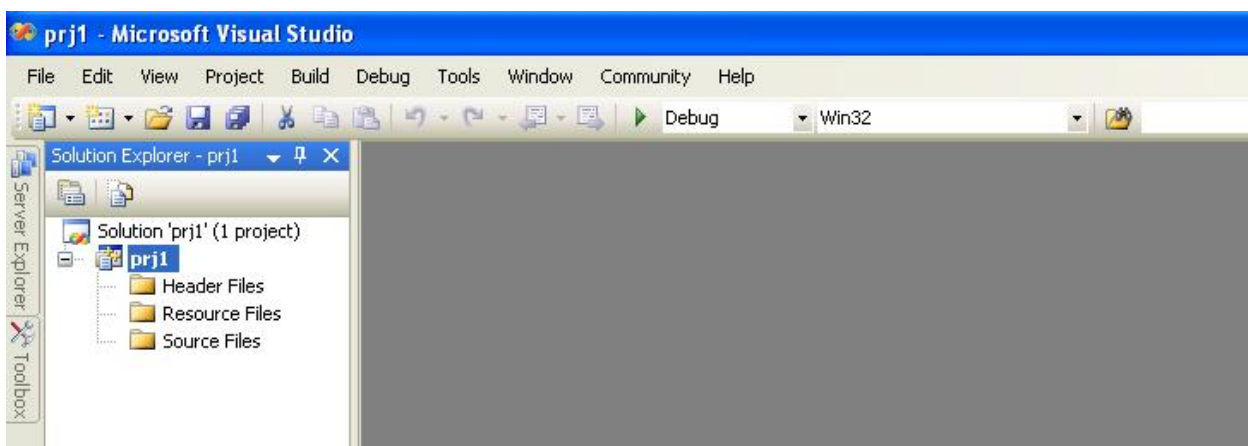


Рис. 2.7. Менеджер проекту.

3. До створеного проекту додати файли вихідних текстів програми. Для цього потрібно натиснути праву клавішу миші на іконці проекту та у спливаючому вікні вибрати Add -> New Item (Рис. 2.8).

4. На основі наведених в методичних вказівках кодів програм підготувати два файли з розширеннями .cpp та .asm. Для цього можна використати будь-

який текстовий редактор (наприклад Notepad). **Зауваження:** створенні файли повинні мати різні імена.

5. Додати створенні файли до проекту Microsoft Visual Studio 2005 так, як наведено на рис. 2.8-2.9.

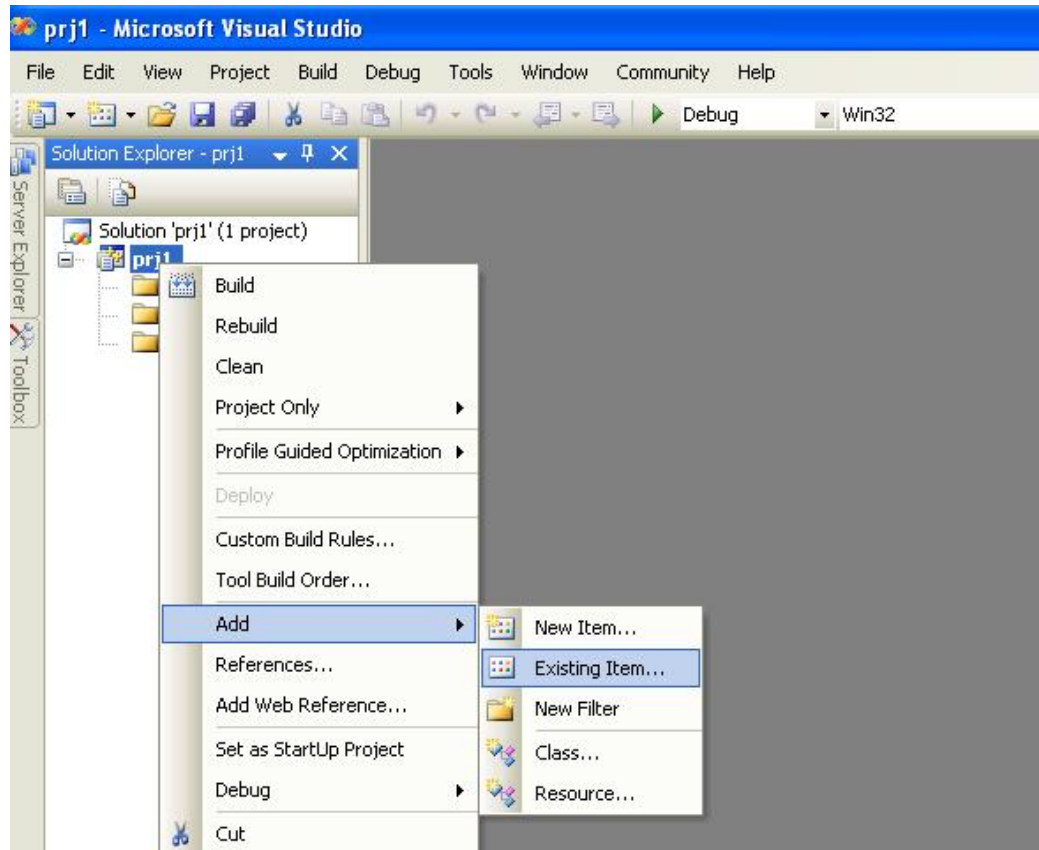


Рис. 2.8. Додавання існуючих файлів з вихідними кодами.

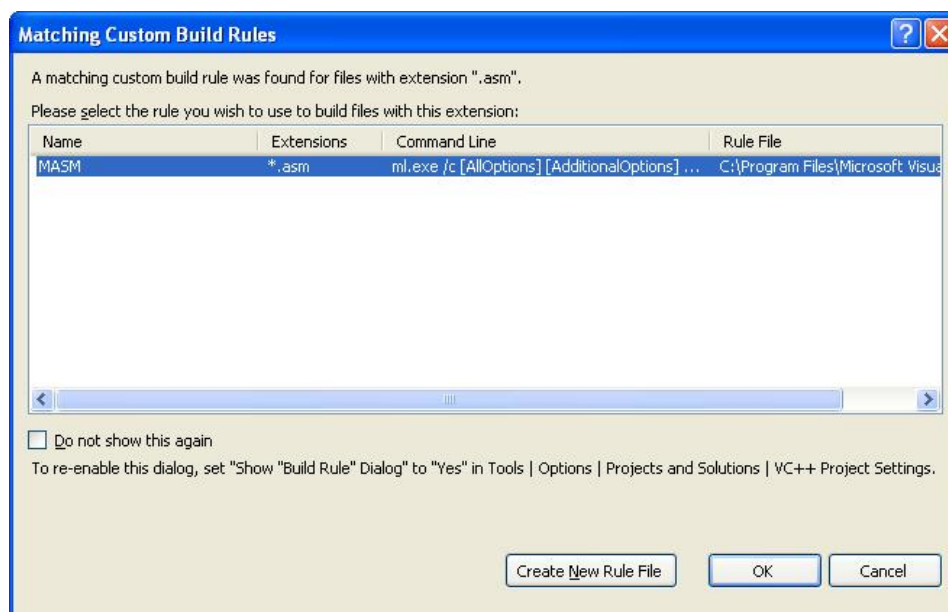


Рис. 2.9. Вибір правила компіляції асемблерних файлів.

При додавання файлу з розширенням .asm у Microsoft Visual Studio 2005 з'явиться вікно, що дозволяє встановити правила компіляції програм, написаних мовою програмування assembler (Рис. 2.9). Вибрати єдине доступне правило та натиснути кнопку «ОК».

6. Скомпілювати створений проект, використавши вкладку «build».

7. Модифікувати тестову програму відповідно до заданого варіанту, скомпілювати та запустити на виконання.

Приклад програми

Програма, що виконує додавання двох цілих чисел
та ілюструє з взаємодією C-ASM

main.cpp

calc.asm

```
#include <stdio.h>
extern "C" int calc(int, int);

int main()
{
    int a=0;
    int b=0;
    int res=0;
    printf("Enter numbers:\n");
    printf("A = ");
    scanf("%d", &a);
    printf("B = ");
    scanf("%d", &b);
    res=calc(a, b);
    printf("Result is: %d\n",res);
    return 0;
}
```

```
.386
.model flat,c
.data
.code
calc PROC
    push ebp
    mov ebp,esp
    mov eax, dword ptr [ebp+8]
    add eax, dword ptr [ebp+12]
    pop ebp
    ret
calc ENDP
END
```

ЛАБОРАТОРНА РОБОТА №3.

ВИКОРИСТАННЯ МАТЕМАТИЧНОГО СПІВПРОЦЕСОРА

Мета: ознайомитися з принципами роботи математичного співпроцесора і використати його можливості для обчислення арифметичних виразів з числами з плаваючою комою.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Математичний співпроцесор запускається центральним процесором. Після запуску він виконує всі обрахунки самостійно і паралельно з роботою центрального процесора. Якщо центральний процесор направляє наступну команду співпроцесору в той момент, коли співпроцесор ще не закінчив виконання попередньої команди, центральний процесор переводиться в стан очікування. Якщо ж співпроцесор не зайнятий, центральний процесор, направивши команду співпроцесору, продовжує свою роботу, не очікуючи завершення обрахунку. Послідовне розташування команд співпроцесора і центрального процесора в коді програми створюють ілюзію послідовного їх виконання. Простіше кажучи можлива така ситуація, що коли центральний процесор звернеться до комірки пам'яті, куди арифметичний співпроцесор повинен був записати результат своїх обчислень і якщо співпроцесор ще не закінчив обрахунки то результату там не буде. Однак є спеціальні засоби синхронізації (команда FWAIT), що дозволять центральному процесорові дочекатися результатів роботи співпроцесора.

Префікс команд та адресація операндів

Команди які призначенні для виконання співпроцесором, записуються в програмі як звичайні машинні команди центрального процесора, але всі вони починаються з байта, який відповідає команді центрального процесора ESC. Зустрівши таку команду, процесор передає її співпроцесору, а сам продовжує виконання програми з наступної команди.

Асемблерна мнемоніка всіх команд співпроцесора починається з букви F, наприклад: FADD, FDIV, FSUB и так далі. Команди співпроцесора можуть адресувати операнди, аналогічно звичайним командам центрального процесора. Операндами можуть бути або дані, які розміщуються в основній пам'яті комп'ютера, або внутрішні регістри співпроцесора.

Для команд арифметичного співпроцесора можливі всі види адресації даних, які використовуються центральним процесором.

Формати даних

Математичний співпроцесор може обробляти дійсні (у форматі з плаваючою комою) та цілі числа.

Дійсні числа. Дійсні числа в загальних обрахунках можна записати наступним чином:

(знак)(мантиса)*10(знак)(порядок)

Наприклад: $-1.35 \cdot 10^5$ (мінус одна ціла тридцять п'ять сотих, помножені на десять в п'ятому степені). Тут знак: мінус, мантиса: 1.35, порядок: 5. Порядок теж може мати знак.

Важливим є таке поняття, як **нормалізоване представлення** чисел: якщо ціла частина мантиси числа складається з однієї цифри, не рівної нулю, то число з плаваючою крапкою називається нормалізованим. Перевага використання нормалізованих чисел полягає в тому, що для фіксованої розрядної сітки числа (тобто для фіксованої кількості цифр в числі) нормалізовані числа мають найбільшу точність. Крім того, нормалізоване представлення виключає неоднозначність, яка може виникнути через те, що кожне число з плаваючою крапкою може бути представлено різними (ненормалізованими) способами (наприклад: $123.5678 \cdot 10^5 = 12.35678 \cdot 10^6 = 1.235678 \cdot 10^7 = 0.1235678 \cdot 10^8$).

При програмуванні на мовах високого рівня зустрічається наступне представлення **чисел з плаваючою крапкою**:

(знак)(мантиса)E(знак)(порядок)

Наприклад, $-5.35E-2$ означає число $-5.35 \cdot 10^{-2}$ (мінус п'ять цілих тридцять п'ять сотих, помножені на десять в мінус другому ступені).

Арифметичний співпроцесор може працювати з дійсними числами в трьох форматах:

- **одинарної точності** (4 байти);
- **подвійної точності** (8 байт);
- **розширеної точності** (10 байт).

При будь-якому представленні старший біт визначає знак дійсного числа: 0 - додатне, 1 – від'ємне. Числа, що рівні за модулем, відрізняються лише цим бітом, оскільки для представлення від'ємних чисел доповняльний код не використовується, на відміну від центрального процесора.

Арифметичний співпроцесор працює з нормалізованими двійковими числами. Двійкове число з плаваючою крапкою називається нормалізованим, якщо ціла частина мантиси рівна 1. З метою розширення розрядної сітки, ця одиниця не зберігається у форматах одинарної і подвійної точності. У форматі з розширеною точністю зберігається і "зайвий" біт цілої частини

нормалізованого числа. Основна причина використання формату з розширеною точністю - запобігання можливій втраті точності обчислень при значній різниці в порядках чисел, що беруть участь в арифметичних операціях.

Поле порядку - це степінь числа 2, на який множиться мантиса, плюс зсув, рівний 127 для одинарної точності, 1023 - для подвійної точності і 16383 - для розширеної точності.

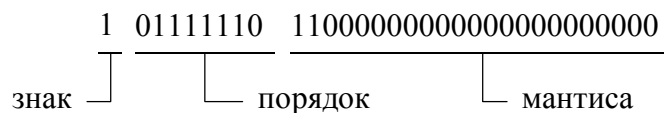
Для того, щоб визначити абсолютне значення числа з плаваючою крапкою, можна скористатися наступними формулами:

- **одинарна точність:** $1.(\text{цифри мантиси}) * 2^{P-127}$;
- **подвійна точність:** $1.(\text{цифри мантиси}) * 2^{P-1023}$;
- **розширена точність:** $1.(\text{цифри мантиси}) * 2^{P-16383}$.

Знак числа визначається старшим бітом.

Наприклад:

Маємо таке число представлено з одинарною точністю:



Для цього числа знаковий біт рівний 1 (від'ємне число), порядок рівний 126, мантиса - 11 (у двійковій системі числення).

Значення цього числа рівне:

$$1.11 * 2^{(126-127)} = -1.75 * 2^{-1} = -0,875$$

Цілі числа. Арифметичний співпроцесор разом з дійсними числами здатний обробляти і цілі числа. Він має команди, що виконують перетворення цілих чисел в дійсні і назад.

Можливі чотири формати цілих чисел:

- **ціле число;**
- **коротке ціле число;**
- **довге ціле число;**
- **упаковане двійково-десятькове число.**

Ціле число займає два байти. Його формат повністю відповідає формату, що використовується центральним процесором. Для представлення від'ємних чисел використовується доповняльний код.

Коротке ціле і довге ціле мають аналогічні формати, але займають, відповідно 4 і 8 байт.

Упаковане двійково-десятькове число займає 10 байт. Це число містить 18 десяткових цифр, розташованих по дві в кожному байті. Знак упакованого BCD числа знаходиться в старшому біті найлівішого байта. Решта біт старшого байта повинна бути рівні 0.

Існують команди співпроцесора, які перетворюють числа у формат упакованих двійково-десяткових чисел з внутрішнього представлення в розширеному дійсному форматі. Якщо програма робить спробу перетворення в упакований формат ненормалізованих чисел, нечисел, нескінченності і т.ін., в результаті виходить невизначеність. Невизначеність в упакованому BCD форматі є числом, в якому два старші байти містять одиниці у всіх розрядах. Вміст решти восьми байтів довільний. При спробі використовувати таке упаковане число в операціях - фіксується помилка.

Регістри співпроцесора

Арифметичний співпроцесор має наступну систему регістрів:

- 8 регістрів загального призначення для роботи з даними (R0–R7, кожний розміром 10 байт), до яких можна звертатися тільки як до елементів стеку (де ST(0) – його вершина);
- регістр стану SR (два байта);
- регістр управління CR (два байта);
- регістр тегів TW (два байта);
- регістр вказівника команди FIP (шість байтів);
- регістр вказівника операнду команди FDIP (шість байтів).

Числові регістри

Числові регістри, як правило, в технічній літературі позначаються ST0 - ST7. Вони організовані за принципом стеку.

Регістр стану в полі ST містить номер числового регістра, що є вершиною стеку. При виконанні команд як операнди можуть виступати числові регістри. У цьому випадку номер вказаного в команді регістра додається до вмісту поля ST регістра стану і таким чином визначається регістр, що використовується. Більшість команд після виконання збільшують поле ST регістра стану, записуючи результати своєї роботи в стек числових регістрів.

Система команд математичного співпроцесора

Можливі три формати команд співпроцесора, аналогічні форматам команд центральних процесорів фірми Intel. Це команди зі звертанням до оперативної пам'яті, команди зі звертанням до одного з числових регістрів і команди без операндів, заданих явним чином. Команди зі звертанням до пам'яті можуть займати від двох до чотирьох байт, залежно від способу адресації. Всі асемблерні мнемоніки команд співпроцесора починаються з букви F, тому їх легко відрізнити від команд центрального процесора.

Команди співпроцесора можна розділити на декілька груп:

- команди пересилки даних;

Таблиця 3.1.

Основні команди математичного співпроцесора

Команда	Функція	Операнд 1	Операнд 2	Результат
Команди пересилки даних				
FLD	Завантажити змінну дійсного типу до стеку співпроцесора	відправник	–	ST(0)
FILD	Завантажити змінну цілого типу до стеку співпроцесора	відправник	–	ST(0)
FST	Копіювати значення зі стеку (регістр ST(0)) у змінну дійсного типу	приймач	–	приймач
FSTP	Перенести значення зі стеку (регістр ST(0)) у змінну дійсного типу	приймач	–	приймач
FIST	Копіювати значення зі стеку (регістр ST(0)) до приймача (змінна цілого типу)	приймач	–	приймач
FISTP	Перенести значення зі стеку (регістр ST(0)) у змінна цілого типу	приймач	–	приймач
FXCH	Обмін значеннями регістрів ST(0) та відправника (інший регістр)	відправник	–	–
FCMOVxx	Група команд умовної пересилки даних	приймач	відправник	ST(0)
Група команд умовної пересилки даних FCMOVxx				
Команда	Реальна умова	Умова для команди FCOM		
FCMOVE	ZF = 1	Якщо рівні		
FCMOVNE	ZF = 0	Якщо нерівні		
FCMOVB	CF = 1	Якщо менше		
FCMOVBE	CF = 1 та ZF = 1	Якщо менше або рівні		
FCMOVNB	CF = 0	Якщо не менше		
FCMOVNBE	CF = 0 та ZF = 0	Якщо не менше або рівні		
Команди управління FPU				
FINIT	Ініціювання роботи математичного співпроцесора FPU	–	–	–
FSTCW	Копіювання вмісту регістру CR до приймача (змінна розміром 2 байта або регістр)	приймач	–	змінна
FLDCW	Копіювання відправника (змінна розміром 2 байта) до регістру CR	відправник	–	CR
FSTSW	Копіювання вмісту регістру SR до приймача (змінна розміром 2 байта або регістр AX)	приймач	–	змінна або AX

- арифметичні команди;
- команди порівнянь чисел;
- трансцендентні команди;
- команди керування.

Команди пересилки даних призначені для завантаження чисел з оперативної пам'яті в числові регістри, записи даних з числових регістрів в оперативну пам'ять, копіювання даних з одного числового регістра в інший.

Арифметичні команди виконують такі операції, як додавання, віднімання, множення, ділення, знаходження квадратного кореня, знаходження часткового залишку, округлення і т.п.

Команди порівняння порівнюють дійсні і цілі числа, виконують аналіз чисел.

Трансцендентні команди призначені для обчислення різних тригонометричних, логарифмічних, показникових і гіперболічних функцій - \sin , \cos , \lg і т.ін.

Команди керування забезпечують установку режиму роботи арифметичного співпроцесора, його скидання і ініціалізацію, перехід співпроцесора в захищений режим роботи і т.д.

Арифметичні команди математичного співпроцесора

Співпроцесор використовує шість основних форматів арифметичних команд:

Fxxx	перший операнд - вершина стеку, другий - наступний елемент стеку, результат операції записується у вершину стеку
Fxxx пам'ять	перший операнд - пам'ять, другим та приймачем є вершина стеку ST(0). Показчик стеку ST не змінюється, команда для дійсних чисел з одинарною і подвійною точністю
Fіxxx пам'ять	аналогічно попередньому формату команди, але операндами можуть бути 16- або 32-розрядні цілі числа
Fxxx ST, ST(i)	для цього формату регістр ST(i) є джерелом, а ST(0) - верхівка стеку - приймачем. Показчик стеку не змінюється
Fxxx ST(i), ST	для цього типу регістр ST(0) є джерелом, а ST(i) - приймачем. Показчик стеку не змінюється
FxxxP ST(i), ST	регістр ST(i) - приймач, регістр ST(0) - джерело. Після виконання команди джерело ST(0) витягується із стеку

Символи "xxx" можуть приймати наступні значення:

ADD - додавання;

SUB - віднімання;
SUBR - зворотне віднімання, тобто операнди міняються місцями;
MUL - множення;
DIV – ділення;
DIVR - зворотне ділення, ділене і дільник міняються місцями.
Окрім основних арифметичних команд є додаткові арифметичні команди:
FSQRT - знаходження квадратного кореня;
FSCALE - масштабування на ступінь числа 2;
FPREM - обчислення часткової остачі;
FRNDINT - заокруглення до цілого;
FXTRACT - виділення порядку числа і мантиси;
FABS - знаходження абсолютного значення (модуля) числа;
FCHS - зміна знаку числа.

КОНТРОЛЬНІ ПИТАННЯ

1. Для чого використовується математичний співпроцесор?
2. Які формати даних може обробляти математичний співпроцесор?
3. Які числові регістри має співпроцесор?
4. Які регістри керування має співпроцесор?
5. Які групи команд має співпроцесор?
6. Які команди пересилки даних співпроцесора ви знаєте?
7. Які команди управління співпроцесора ви знаєте?
8. Які арифметичні команди FPU ви знаєте?

ЛІТЕРАТУРА

- 1.Березко Л.О., Троценко В.В. Особливості програмування в турбо-асемблері. - Київ, НМК ВО, 1992.
- 2.Абель П.Язык ассемблера для IBM PC и программирования. Пер. з англ.- М., "Высшая школа", 1992.
3. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2 (2A & 2B): Instruction Set Reference, A-Z. – 2011. Режим доступу:
<http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-2b-instruction-set-a-z-manual.html>

ЗАВДАННЯ

1. Створити *.exe програму, яка реалізовує обчислення, заданого варіантом виразу. Вхідні дані повинні вводитися з клавіатури, під час виконання

програми, як дійсні числа зі знаком. Програма повинна складатися з двох модулів:

головний модуль – створюється мовою C і має забезпечити ввід необхідних даних, виклик асемблерної процедури для обчислення виразу та вивід результату обчислень;

модуль безпосередніх обчислень – здійснює всі необхідні арифметичні дії з використанням математичного співпроцесора.

2. Переконалися у правильності роботи кожного модуля зокрема та програми загалом.
3. Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї.
4. Дати відповідь на контрольні запитання.

Таблиця 3.1.

Варіанти завдань

№	Вираз	К
1.	$X=A_4+B_2*C_1-D_2/E_1+K$	1254021
2.	$X=A_4/B_2+C_3-D_1*E_1-K$	202
3.	$X=K-B_4+C_2/D_1-E_1*F_2$	37788663
4.	$X=A_1*(B_2+C_1)-D_1/E_2+K$	45694
5.	$X=A_2*B_2-A_2*C_1-D_1/E_2+K$	505
6.	$X=K+B_2/C_1-D_2*F_2-E_1$	6DD02316
7.	$X=A_4/B_2-C_1*(D_1+E_2-K)$	717
8.	$X=A_2-B_1+K-D_2/E_1+F_1*B_2$	88
9.	$X=A_1*B_2-A_1*C_2+D_2/(E_1+K)$	29
10.	$X=A_2-B_1/C_2+K+E_2*F_1$	2310
11.	$X=(A_2-B_1-K)*D_1+E_4/F_2$	311
12.	$X=K+B_1/C_2-D_2*F_2-E_1$	7055E0AC
13.	$X=A_2/B_1+C_1*(D_1+E_2-K)$	2513
14.	$X=A_2-B_1-K-D_2/E_1+F_1*B_1$	614
15.	$X=A_2+(B_1*C_2)-D_4/E_2+K$	4569600F
16.	$X=A_1/B_2+C_3-D_1*E_1+K$	616
17.	$X=A_1-K+C_1/D_2-E_1*F_1$	1017
18.	$X=A_1*(B_2-C_1)+D_2/E_1+K$	56987018
19.	$X=A_2*B_2+A_2*C_1-D_2/E_1+K$	4019
20.	$X=K+B_1/C_2-D_1*F_1-E_2$	18932020
21.	$X=A_1/B_2+C_1*(D_2-E_1+K)$	21
22.	$X=K-B_2-C_1-D_2/E_1+F_2*B_1$	45781022

23.	$X=A_2*B_2-C_1+D_1/E_2+K$	7AA02023
24.	$X=K-B_2/C_1+D_1+E_2*F_2$	74569024
25.	$X=(K-B_2-C_1)*D_1+E_4/F_2$	2B05025
26.	$X=A_2+K+C_2/D_1-E_1*F_1$	6C26
27.	$X=A_2*B_1+C_1/(K-E_1*F_1)$	A77627
28.	$X=K+B_1/C_2+D_2-E_2/F_1$	3FF28
29.	$X=K-B_1*C_1+D_2-F_2/E_1$	12A0C029
30.	$X=K+B_2-D_2/C_1+E_1*F_2$	25630

A, B, C, D, E, F - дійсні числа, де індекс визначає точність внутрішнього подання (1 - одинарної, 2 – подвійної точності). Константу K подано у 16-му форматі.

ПРИКЛАД ВИКОНАННЯ

Завдання: Написати програму, яка обчислює арифметичний вираз над дійсними числами, введеними в десятковій системі і результат виводить на екран в десятковій формі зі знаком.

Заданий вираз:

$$X=A+B$$

Оскільки дані, що вводяться повинні бути знаковими і зберігати заданий розмір, проаналізуємо які значення можуть бути введені, а отже і отримані в результаті обчислень.

Текст програми, яка повністю реалізовує завдання, приведений нижче. Програма передбачає ввід/вивід знакових даних з використанням функції C.

Далі, застосовуються можливості арифметичного співпроцесора для обчислень всіх арифметичних дій. Для цього в сегменті даних передбачаємо змінні, що будуть використовуватися для обчислень. Вони повинні бути 4 або 8 байтними (згідно форматів даних для співпроцесора).

Програма з взаємодією C-ASM, що виконує додавання двох дійсних чисел з використанням математичного співпроцесора

main.cpp

```
#include <stdio.h>

extern "C" float
calc(float, float);

int main()
{
    float a=0;
```

calc.asm

```
.386
.model flat, c
.data
.code
calc PROC
    push ebp
    mov ebp, esp
    fld dword ptr [ebp+8]
```



```

float b=0;                                fadd dword ptr [ebp+12]
float res=0;                              pop ebp
printf("Enter numbers:\n");              ret
printf("A = ");                          calc ENDP
scanf("%f",&a);                          END
printf("B = ");
scanf("%f",&b);
res=calc(a,b);
printf("Result is: %f\n",res);
return 0;
}

```

ЛАБОРАТОРНА РОБОТА №4.

ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ НА МАТЕМАТИЧНОМУ СПІВПРОЦЕСОРІ

Мета: познайомитися з принципами роботи математичного співпроцесора та оволодіти навиками використання вбудованих елементарних математичних функцій та реалізації розгалужень.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Команди порівняння чисел

У центральному процесорі команди умовних переходів виконуються відповідно до значень окремих бітів регістра прапорців процесора. У арифметичному співпроцесорі існують спеціальні команди порівнянь, за наслідками виконання яких, встановлюються біти кодів умов в регістрі стану:

FCOM – порівняння;

FICOM – порівняння цілих чисел;

FCOMP - порівняння дійсних чисел і вилучення зі стеку;

FICOMP - порівняння цілих чисел і вилучення зі стеку;

FCOMPP - порівняння і подвійне вилучення зі стеку (ST(0), ST(1));

FTST - порівняння операнда з нулем;

FXAM - аналіз операнда на тип числа (скінчене число, денормалізоване число, нуль, некінечність, ...).

Команда FCOM віднімає вміст операнда, розміщеного в оперативній пам'яті, від значення у вершині стеку ST(0). Результат віднімання нікуди не записується і покажчик вершини стеку ST не змінюється.

Позначимо операнд команди порівняння як "x". Нижче, приведемо значення бітів кодів умови після виконання команди "FCOMx":

$C3 = 0, C0 = 0 \quad ST(0) > x$

$C3 = 0, C0 = 1 \quad ST(0) < x$

$C3 = 1, C0 = 0 \quad ST(0) = x$

$C3 = 1, C0 = 1 \quad ST(0) \text{ і } x \text{ непорівнювані}$

Остання комбінація виникає при спробі порівняння не чисел, невизначеностей або нескінченості, а також в деяких інших випадках.

Операндами команди FICOM є 16- або 32-розрядні цілі числа, а в решті - аналогічна команді FCOM.

Команди FCOMP і FICOMP аналогічні, відповідно, командам FCOM і FICOM, за винятком того, що після виконання операнд вилучається зі стеку.

Команда FCOMPP виконує ті ж дії, що і FCOM, але вона після виконання вилучає зі стеку обидва операнди, що брали участь в порівнянні.

Команда FTST призначена для порівняння операнду з нулем. Після її виконання коди умов встановлюються згідно з наведеним нижче:

$C3 = 0, C0 = 0$ ST(0) > 0

$C3 = 0, C0 = 1$ ST(0) < 0

$C3 = 1, C0 = 0$ ST(0) = 0

$C3 = 1, C0 = 1$ ST(0) і 0 непорівнювані

Команда FXAM аналізує вміст ST(0). Після її виконання встановлюються коди умов, згідно яких можна визначити знак числа, його скінченність або нескінченність, нормалізованість і т.д. Біт C1 містить знак числа, що аналізується: 0 - додатний, 1 - від'ємний. За допомогою біта C0 можна визначити, є число скінченим або нескінченим: 0 - скінчене число, 1 - нескінчене. Для скінчених чисел подальша класифікація може проводитися за вмістом кодів умов C2 і C3:

$C3 = 0, C0 = 0$ Ненормалізоване число

$C3 = 0, C0 = 1$ Нормалізоване число

$C3 = 1, C0 = 0$ Нульове число

$C3 = 1, C0 = 1$ Число денормалізоване

Аналогічно, для нескінчених чисел коди умов C2 і C3 мають наступні значення:

$C3 = 0, C0 = 0$ Нечисло

$C3 = 0, C0 = 1$ Нескінчене число

$C3 = 1, C0 = 0$ Порожнє число

$C3 = 1, C0 = 1$ Порожнє число

Для реалізації порівняння чисел необхідно за допомогою команди "FSTSW AX" переписати вміст регістра стану співпроцесора в регістр AX центрального процесора. Далі вміст регістра AH переписати в регістр прапорів центрального процесора за допомогою команди SAHF. Біти кодів умов співпроцесора відображаються (проектуються) на регістр прапорів центрального процесора так, що без додаткових дій, можна використовувати команди умовних переходів, але тільки беззнакового типу.

Наприклад, в наступному фрагменті програми виконується перехід до мітки compute, якщо операнди рівні:

```
.586
```

```
...
```

```
fcom
```

```
fstsw ax
sahf
je compute
...
```

Трансцендентні команди

Трансцендентні команди призначені для обчислення наступних функцій:

- тригонометричні (\sin , \cos , tg ...)
- зворотні тригонометричні (\arcsin , \arccos ...)
- показникові (x^y , 2^x , 10^x , e^x)
- гіперболічні (sh , ch , th ...)
- зворотні гіперболічні (arsh , arch , arth ...)

Ось список всіх трансцендентних команд математичного співпроцесора:

FPTAN - обчислення часткового тангенса;

FPATAN - обчислення часткового арктангенса;

FYL2X - обчислення $y \cdot \log_2(x)$;

FYL2XP1 - обчислення $y \cdot \log_2(x+1)$;

F2XM1 - обчислення $2x-1$;

FCOS - обчислення $\cos(x)$;

FSIN - обчислення $\sin(x)$;

FSINCOS - обчислення $\sin(x)$ і $\cos(x)$ одночасно.

Команда FPTAN обчислює частковий тангенс $ST(0)$, розміщуючи в стеку два числа x та y , такі що $y/x = \operatorname{tg}(ST(0))$. Для сучасних співпроцесорів $x = 1$, а $y = \operatorname{tg}(ST(0))$.

Після виконання команди число y розташовується в $ST(0)$, а число x заноситься у вершину стеку (тобто записується в $ST(1)$). Для старих співпроцесорів (Intel 8087, 80287) аргумент команди FPTAN повинен бути нормалізованим і знаходиться в межах: $0 \leq ST(0) \leq \pi/4$.

Таблиця 4.1.

Результати обчислень інструкції FPTAN

ST(0) джерело	ST(0) приймач
$-\infty$ або $+\infty$	виняткова ситуація
$-R$ або $+R$	від $-R$ до $+R$
-0	-0
$+0$	$+0$
нечисло	Нечисло
Примітка: R – скінченне дійсне число	

Для нових співпроцесорів аргумент команди FPTAN повинен бути поданий в радіанах в межах $\pm 2^{63}$. Результати обчислень в залежності від вхідних даних подані у таблиці нижче.

Якщо аргумент є більшим або меншим, ніж $\pm 2^{63}$, то тангенс необхідно шукати сумісним використанням команди FPREM і FPTAN перевіряючи прапорець C2 до повного знаходження значення тангенсу.

Користуючись знайденим значенням часткового тангенса, можна обчислити інші тригонометричні функції за наступними формулами:

$$\begin{aligned} \sin(z) &= 2 \cdot (y/x) / (1 + 2 \cdot (y/x)); \\ \cos(z) &= (1 - 2 \cdot (y/x)) / (1 + 2 \cdot (y/x)); \\ \operatorname{tg}(z/2) &= y/x; \\ \operatorname{ctg}(z/2) &= x/y; \\ \operatorname{cosec}(z) &= (1 + 2 \cdot (y/x)) / 2 \cdot (y/x); \\ \sec(z) &= (1 + 2 \cdot (y/x)) / (1 - 2 \cdot (y/x)). \end{aligned}$$

Де z - значення, що знаходилося в ST(0) до виконання команди FPTAN, x і y - значення в регістрах ST(0) і ST(1), відповідно.

Команда FPATAN обчислює частковий арктангенс $z = \operatorname{arctg}(\operatorname{ST}(1)/\operatorname{ST}(0)) = \operatorname{arctg}(x/y)$. Перед виконанням команди числа x і y розташовуються в ST(1) і ST(0), відповідно. Аргументи команди FPATAN повинні знаходитися в межах: $0 \leq x < y < +\infty$. Результат записується в ST(1), а операнд в ST(0) вилучається зі стеку. Таким чином результат після виконання інструкції розміщується у ST(0).

Команда FYL2X обчислює вираз $y \cdot \log_2(x)$, операнди x і y розміщуються, відповідно в ST(0) та ST(1). Операнди вилучаються зі стеку, а результат записується в стек. Параметр x повинен бути додатнім числом.

Користуючись результатом виконання цієї команди, можна обчислити логарифмічні функції, наступним чином:

Логарифм за основою два: $\log_2(x) = \operatorname{FYL2X}(x)$.

Натуральний логарифм:

$$\log_e(x) = \log_e(2) \cdot \log_2(x) = \operatorname{FYL2X}(\log_e(2), x) = \operatorname{FYL2X}(\operatorname{FLDLN2}, x).$$

Десятковий логарифм:

$$\log_{10}(x) = \log_{10}(2) \cdot \log_2(x) = \operatorname{FYL2X}(\log_{10}(2), x) = \operatorname{FYL2X}(\operatorname{FLDLG2}, x).$$

Функція FYL2XP1 обчислює вираз $y \cdot \log_2(x + 1)$, де x відповідає ST(0), а y - ST(1). Результат записується в ST(0), обидва операнди вилучаються зі стеку та втрачаються. На операнд x накладається обмеження: $0 < x < 1 - 1 / \sqrt{2}$.

Команда F2XM1 обчислює вираз $2x-1$, де x - ST(0). Результат записується в ST(0), параметр повинен знаходитися в наступних межах: $0 \leq x \leq 0,5$.

Команда FCOS обчислює $\cos(x)$. Параметр x повинен знаходитися в ST(0), туди ж записується результат виконання команди.

Команда FSIN аналогічна команді FCOS, але обчислює значення синуса ST(0).

Команда FSINCOS обчислює одночасно значення синуса і косинуса параметра ST(0). Значення синуса записується в ST(1), косинуса - в ST(0).

Команди керування

Команди, керування, призначені для роботи з нечисловими регістрами співпроцесора. Деякі команди мають альтернативні варіанти. Мнемоніки цих команд можуть починатися з FN або F. Перший варіант відповідає командам “Без очікування”. Для таких команд процесор не перевіряє, чи зайнятий співпроцесор виконанням команди, тобто біт зайнятості B не перевіряється. Особливі випадки також ігноруються.

Варіанти команд “З очікуванням” діють так само, як і звичайні команди співпроцесора.

Ось список команд керування для співпроцесора:

FNSTCW (FSTCW) - записати управляюче слово (записує вміст регістра управління в оперативну пам'ять).

FLDCW - завантажити управляюче слово (завантажує регістр управління з оперативної пам'яті і, як правило, використовується для зміни режиму роботи співпроцесора).

FNSTSW (FSTSW) - записати слово стану (записує вміст регістра стану в оперативну пам'ять).

FNSTSW AX (FSTSW AX) - записати слово стану в AX (записує вміст регістра стану в регістр AX центрального процесора, де можливий аналіз вмісту за допомогою команд умовних переходів).

FNCLX (FCLEX) - скинути особливі випадки (скидає прапорці особливих випадків в регістрі стану співпроцесора, також скидаються біти ES і B).

FNINIT (FINIT) - ініціалізувати співпроцесор (ініціалізує регістр стану, регістр управління, і регістр тегів таким чином:

регістр управління - проектна нескінченність, округлення до найближчого, розширена точність, всі особливі випадки замасковані;

регістр стану - B=0 (біт зайнятості скинутий), код умови не визначений, ST=ES=0, прапорці особливих випадків встановлені в нуль;

регістр тегів - усі поля регістру тегів містять значення 11 (порожній регістр));

FNSTENV (FSTENV) - записати оточення (записує в пам'ять вміст всіх регістрів, окрім числових, у визначеному форматі. Команда корисна при обробці особливих випадків);

FLDENV - завантажити оточення (завантажує регістри, збережені командою FNSTENV);

FNSAVE (FSAVE) - записати повний стан (діє аналогічно команді FNSTENV, але додатково зберігає вміст числових регістрів);

FRSTOR - відновити повний стан (діє аналогічно команді FLDENV, але додатково відновлює вміст числових регістрів);

FINCSTP - збільшити показчик стека SP на 1;

FDECSTP - зменшити показчик стека SP на 1;

FFREE - звільнити регістр (визначає числовий регістр ST, вказаний як операнд, як порожній, записуючи у відповідне поле регістра тегів значення 11);

FNOP - порожня команда, немає операції (не робить жодних дій);

FSETPM - встановлює захищений режим роботи (переводить співпроцесор в захищений режим роботи).

КОНТРОЛЬНІ ПИТАННЯ

1. Для чого використовується математичний співпроцесор?
2. Які формати даних може обробляти математичний співпроцесор?
3. Які числові регістри має співпроцесор?
4. Які регістри керування має співпроцесор?
5. Які групи команд має співпроцесор?
6. Які арифметичні команди співпроцесора ви знаєте?
7. Які команди керування співпроцесора ви знаєте?
8. Які трансцендентні команди FPU ви знаєте?

ЛІТЕРАТУРА

1. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC XT и AT. - М. "Финансы и статистика", 1992, стор. 13-31.
2. Березко Л.О., Троценко В.В. Особливості програмування в турбо-асемблері. - Київ, НМК ВО, 1992.
3. Дао Л. Программирование микропроцессора 8088. Пер. с англ. - М.: "Мир", 1988.
4. Абель П. Язык ассемблера для IBM PC и программирования. Пер. з англ. - М., "Высшая школа", 1992.
5. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2 (2A & 2B): Instruction Set Reference, A-Z. – 2011. Режим доступу: <http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-2b-instruction-set-a-z-manual.html>

ЗАВДАННЯ

- Створити *.exe програму, яка реалізовує обчислення, заданого варіантом виразу. Вхідні дані повинні вводитися з клавіатури, як дійсні числа. Програма повинна складатися з двох модулів:
головний модуль – створюється мовою C і має забезпечити ввід необхідних даних, виклик асемблерної процедури для обчислення виразу та вивід результату обчислень;
модуль безпосередніх обчислень – здійснює всі необхідні арифметичні дії з використанням математичного співпроцесора.
- Переконаватися у правильності роботи кожного модуля зокрема та програми загалом.
- Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї.
- Дати відповідь на контрольні запитання.

Таблиця 4.1.

Варіанти завдань

варіант	Завдання	Варіант	Завдання
1.	$X_i = \begin{cases} \frac{2 * c - d + \sqrt{23 * a_i}}{\frac{a_i}{4} - 1} & c > d \\ \frac{c + 4 * d - \sqrt{123 * c}}{1 - \frac{a_i}{2}} & c \leq d \end{cases}$	16.	$X_i = \begin{cases} \frac{a_i + \frac{c}{d} - \sqrt{28 * d}}{4 * b * a + 1} & c > d \\ \frac{\frac{c}{d} - \sqrt{24 + d} + a_i}{2 * a_i * c - 4} & c \leq d \end{cases}$
2.	$X_i = \begin{cases} \frac{-2 * c + d * 82}{tg(\frac{a_i}{4} - 1)} & c > d \\ \frac{lg(2 * c - a_i) + d - 152}{\frac{a_i}{4} + c} & c \leq d \end{cases}$	17.	$X_i = \begin{cases} \frac{2 * c - ln(a_i + d) * c}{c/4 - 1} & c > d \\ \frac{41 - d/4 - 1}{c/tg(d + a_i) - d} & c \leq d \end{cases}$
3.	$X_i = \begin{cases} \frac{tg(a_i + c/4) - 12 * d}{a_i * b - 1} & c > d \\ \frac{-2 * c - sin(a_i/d) + 53}{\frac{a_i}{4} - b} & c \leq d \end{cases}$	18.	$X_i = \begin{cases} \frac{a - c * 4 - 1}{c/31 + tg(a_i * d)} & c > d \\ \frac{tg(d/a_i + 4) + d}{c - d + 1} & c \leq d \end{cases}$
4.	$X_i = \begin{cases} \frac{2 * c - lg(d/4)}{a_i * a_i - 1} & c > d \\ \frac{tg(c) - d * 23}{2 * a_i - 1} & c \leq d \end{cases}$	19.	$X_i = \begin{cases} \frac{lg(21 - a_i) * c/4}{1 + c/a_i + d} & c > d \\ \frac{c - ln(33 + d)/4}{a_i - c/d - 1} & c \leq d \end{cases}$

5.	$X_i = \begin{cases} \frac{2*c - d/23}{\ln(1 - \frac{a_i}{4})} & c > d \\ \frac{4*c + d - 1}{c - tg \frac{a_i}{2}} & c \leq d \end{cases}$	20.	$X_i = \begin{cases} \frac{2*b - 38*c}{arctg(d + a_i)/c + 1} & c > d \\ \frac{arctg(c/4 + 28)*d}{a_i/d - c - 1} & c \leq d \end{cases}$
6.	$X_i = \begin{cases} \frac{2*c - d*\sqrt{\frac{42}{d}}}{c + a_i - 1} & c > d \\ \frac{\sqrt{\frac{25}{c} - d + 2}}{d + a_i - 1} & c \leq d \end{cases}$	21.	$X_i = \begin{cases} \frac{a_i*c/4 - 1}{\sqrt{41 - d} - c*a_i} & c > d \\ \frac{1 + a_i - d/2}{\sqrt{24 + d - c} + a_i} & c \leq d \end{cases}$
7.	$X_i = \begin{cases} \frac{arctg(c - d/2)}{2*a_i - 1} & c > d \\ \frac{4*lg(c) - d/2 + 23}{a_i*a_i - 1} & c \leq d \end{cases}$	22.	$X_i = \begin{cases} \frac{\ln(a_i*d + 2)*c}{41 - d/c + 1} & c > d \\ \frac{lg(4*d - c)*a_i}{c/d - 1} & c \leq d \end{cases}$
8.	$X_i = \begin{cases} \frac{c*tg(d + 23)}{a_i/2 - 4*d - 1} & c > d \\ \frac{c/d + \ln(3*a_i/2)}{c - a_i + 1} & c \leq d \end{cases}$	23.	$X_i = \begin{cases} \frac{2*c + tg(a_i - 21)}{c/a_i + d + 1} & c > d \\ \frac{4/c + tg(3*a_i)}{c/a_i - d - 1} & c \leq d \end{cases}$
9.	$X_i = \begin{cases} \frac{2*c + lg(d)*51}{d - a_i - 1} & c > d \\ \frac{2*c + \ln(d/4) + 23}{a_i*a_i - 1} & c \leq d \end{cases}$	24.	$X_i = \begin{cases} \frac{8*lg(d - 1) - c}{a_i*2 + d/c} & c > d \\ \frac{4*\ln(a_i/d) + 1}{c - d + a_i} & c \leq d \end{cases}$
10.	$X_i = \begin{cases} \frac{42*c - d/2 + 1}{a_i*a_i - \ln(d - 5)} & c > d \\ \frac{arctg(2*c)/d + 2}{d - a_i - 1} & c \leq d \end{cases}$	25.	$X_i = \begin{cases} \frac{4*\ln(d)/c + 1}{2*c + a_i - c} & c > d \\ \frac{arctg(d - c)/d + a_i/4}{a_i + c - 1} & c \leq d \end{cases}$
11.	$X_i = \begin{cases} \frac{arctg(12/c) + 73}{a_i*a_i - 1} & c > d \\ \frac{2*c/a_i - d*d}{d + tg(a_i - 1)} & c \leq d \end{cases}$	26.	$X_i = \begin{cases} \frac{arctg(a_i - c)*d + 28}{4*c/a_i + 1} & c > d \\ \frac{c*d - 24 + a_i}{d - lg(3*c)} & c \leq d \end{cases}$
12.	$X_i = \begin{cases} \frac{\sqrt{\frac{53}{a_i} + d - 4*a_i}}{1 + a_i*c} & c > d \\ \frac{\sqrt{15*a_i + b} - \frac{a_i}{4}}{c*d - 1} & c \leq d \end{cases}$	27.	$X_i = \begin{cases} \frac{a_i + d/c}{a_i - c/4 + 1} & c > d \\ \frac{\sqrt{\frac{41*d}{a_i}} + 1}{a_i - c/d} & c \leq d \end{cases}$

13.	$X_i = \begin{cases} \frac{-25/a_i + c - \lg(c)}{1 + c * d / 2} & c > d \\ \frac{\lg(4 * a_i - 1) + d / 2}{d * c - 5} & c \leq d \end{cases}$	28.	$X_i = \begin{cases} \frac{a_i + \lg(d / 4 - 1)}{c / 3 - a_i * d} & c > d \\ \frac{c * a_i + c / 2}{c - \lg(d + 1)} & c \leq d \end{cases}$
14.	$X_i = \begin{cases} \frac{8 * \lg(d + 1) - c}{a_i / 2 + d * c} & c > d \\ \frac{4 * c - \ln(d - 1)}{a_i / d + 18} & c \leq d \end{cases}$	29.	$X_i = \begin{cases} \frac{\lg(25 + 2 * a_i) / d}{c + a_i - 1} & c > d \\ \frac{c + 23 - d + 4}{a_i - \ln(a_i + c / d)} & c \leq d \end{cases}$
15.	$X_i = \begin{cases} \frac{\arctg(4 * d) / c - 1}{12 + a_i - c} & c > d \\ \frac{\arctg(c) + c * d - a_i / 4}{a_i * d - 1} & c \leq d \end{cases}$	30.	$X_i = \begin{cases} \frac{d / 2 - 53 / c}{\arctg(d - a_i) * c + 1} & c > d \\ \frac{c * 4 + 28 * d / c}{5 - \arctg(a_i * d)} & c \leq d \end{cases}$

Примітка: a_i – елементи масиву дійсних чисел подвійної точності; c та d – дійсні числа одинарної точності.

ПРИКЛАД ВИКОНАННЯ

Завдання: Написати програму, яка обчислює арифметичний вираз над знаковими даними, введеними в десятковій системі і результат виводить на екран в десятковій формі зі знаком.

Знайти результат обчислення виразу:

$$y = \begin{cases} \cos(3\pi \cdot x), & x > 3 \\ \sin(3\pi \cdot x), & x \leq 3. \end{cases}$$

Текст програми, яка реалізовує завдання, приведений нижче. Програма передбачає ввід/вивід дійсних чисел з використанням функції C. Далі, застосовуються можливості арифметичного співпроцесора для обчислень всіх арифметичних дій. Для цього в сегменті даних передбачаємо змінні, що будуть використовуватися для обчислень. Вони повинні бути 4 або 8 байтними (згідно форматів даних для співпроцесора).

Програма обчислення трансцендентних функцій з використанням математичного співпроцесора

main.cpp	calc.asm
<code>#include <stdio.h></code>	<code>.386</code>
<code>extern "C" float calc(float);</code>	<code>.model flat, c</code>
<code>int main()</code>	<code>.data</code>
<code>{</code>	<code>const_3 dd 3.0</code>
<code>float x=0;</code>	<code>.code</code>
	<code>calc PROC</code>
	<code>push ebp</code>

```

float res=0;
printf("Enter numbers:\n");
printf("X = ");
scanf("%f",&x);
res=calc(x);
printf("Result is: %f\n",res);
return 0;
}

mov ebp, esp
finit
fld dword ptr[ebp+8]
fcomp const_3
fstsw ax
sahf
fldpi
fmul
fmul const_3
ja __cos
fsin
jmp __next
__cos:
fcos
__next:
pop ebp
ret
calc ENDP
END

```

ЛАБОРАТОРНА РОБОТА №5.

ОСОБЛИВОСТІ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ФУНКЦІЙ WIN32 API

Мета: Ознайомитись з можливостями та набути навиків програмування на Асемблері в ОС Windows та засвоїти навички використання функцій API Win32.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Незважаючи на те що Windows 95/NT/XP є складнішими операційними системами у порівнянні з DOS, програмувати для них на асемблері набагато простіше. Windows-задача запускається в 32-бітному режимі з плоскою (flat) моделлю пам'яті та не передбачає низькорівневого програмування різних пристроїв комп'ютера так як це було у DOS. В існуючих операційних системах програми користуються тільки системними викликами Win32 API функцій, число яких тут перевищує 2000 (близько 2200 для Windows 95 і 2434 для Windows NT).

Всі Windows-задачі використовують спеціальний формат виконавчих файлів - формат PE (Portable Executable). Такі файли починаються як звичайні exe-файл старого зразка (їх також називають MZ за першими двома символами заголовку). Якщо такий файл запустити в DOS, він виконається й видасть повідомлення про помилку (текст повідомлення залежить від використовуваного компілятора), у той час як Windows помітить, що після звичайного MZ-заголовка файлу йде PE-заголовок, і запустить програму. Це буде означати лише те, що для компіляції програм будуть потрібні інші параметри в командному рядку.

Win32 API (розшифровується як інтерфейс прикладного програмування) - це множина підпрограм-функцій, на яких побудована операційна система Windows, яка використовує 32х-бітну адресацію, починаючи з Windows 95 і закінчуючи Windows XP. Розробники Windows зробили багато зусиль щоб стандартизувати як назви функцій, так і їх параметри. Тому використовувати їх не так важко, якщо засвоїти деякі загальні концепції.

Більшість функцій доступні для програм користувача, які написані для Windows на будь-якій мові програмування (у тому числі і на асемблері). Множина цих функцій розширюється при переході до наступної версії Windows, таким чином, забезпечується сумісність розроблених раніше програм із новими версіями операційної системи. Існують і функції, які не відображені

в документації, або для свого застосування вимагають від програми спеціальних прав доступу до пам'яті.

Суть функцій API зрозуміти значно легше, якщо уявити, з яких файлів вони викликаються і на які групи ці функції поділяються. Асемблер - це як раз той зручний і простий засіб, який дозволить вам звертатись безпосередньо до будь-якої функції API, що знаходиться у DLL-файлі.

Секрет пізнання операційної системи через програмування на асемблері полягає у тому, що сам асемблер не накладає жодних обмежень на програму та дані, з якими вона працює. Це повинен робити сам програміст з метою захисту операційної системи від своїх некоректних дій. Таким чином, основною метою системного програмування є написання коректних програм з необмеженими можливостями (в рамках операційної системи). Для збереження коректності необхідно користуватися певними правилами програмування, які будуть зрозумілі на конкретних прикладах.

Типи програм. Процесори сімейства Intel можуть працювати в трьох основних режимах: реальному, віртуальному і захищеному. При включенні комп'ютера його процесор працює в реальному режимі. Після завантаження операційної системи (ОС) процесор може бути переключений програмами ОС в інші режими. В реальному та віртуальному режимах використовується 16-бітна адресація з фіксованими сегментами по 64К. У захищеному режимі використовується 32х-бітна адресація з необмеженими сегментами, і адреса звертання до пам'яті формується (на апаратному рівні) за допомогою дескрипторних таблиць, в яких задаються початкові адреси сегментів, їх довжина, та права доступу до пам'яті і до портів для процесів, які їх використовують. Крім того, в захищеному режимі реалізоване апаратне переключення між задачами за допомогою спеціальних таблиць.

Особливості виклику функцій API. Найбільш перспективним з точки зору програмування є захищений режим, тому що він використовує всі апаратні можливості комп'ютера. Отже, функції API для Windows відіграють ту ж саму роль, що і переривання INT 21h для DOS в реальному або віртуальному режимі, але, відмінності між ними досить суттєві, серед них:

- функції API не відміняють, а замінюють програмні переривання. Механізм обробки апаратних переривань залишається на рівні драйверів пристроїв;
- стандарт виклику функцій API ґрунтується на передачі параметрів через стек (а не через регістри);
- значення кожної функції повертається в регістрі EAX. Якщо функція повертає структуру даних, то регістр EAX містить логічну ознаку виконання, а адресу структури необхідно передати до функції як параметр;

- функції API працюють у захищеному режимі процесора, а переривання DOS - у реальному чи віртуальному режимі.

Функції API зберігаються у різних бібліотеках динамічного компонування, які знаходяться у файлах із розширенням DLL, наприклад, kernel32.dll, user32.dll, gdi32.dll та ін. Ці файли знаходяться у системному каталозі Windows (наприклад, "C:\Windows\System"). У разі необхідності, програміст може створити DLL-файл з набором своїх функцій.

Програми Windows звертаються до функцій API за допомогою команд апаратного виклику CALL, наприклад: `call MessageBoxA`, де `MessageBoxA` - 32х-бітна адреса функції. Саме ця назва функції фігурує у файлі user32.dll (подивіться редактором цей файл). Перелік можливих функцій є у файлі /tasm/lib/Import32.lib, який називається бібліотекою імпорту.

Параметри для виконання будь-якої функції API перед її викликом повинні засилатися в стек, починаючи з останнього параметра (див. текст програми). Тому кожний параметр є 32х-бітним числом (в якому можуть використовуватись не всі біти). Параметрами досить часто бувають спеціальні дескриптори - хендли (handle) та атоми.

Дескриптори (хендли та атоми) - це унікальні цілі числа, які Windows використовує для ідентифікації об'єктів, які створюються або використовуються в системі. Хендли займають по 4 байти, а атоми - по 2 байти. Хендли ідентифікують вікна, меню, блоки пам'яті, екземпляри програми, пристрої виводу, файли, аудіо та відео потоки, та інші об'єкти. Атоми ідентифікують стандартні іконки, курсори та об'єкти, які не змінюються при наступному завантаженні системи.

Більшість дескрипторів є значеннями індексів внутрішніх таблиць, які Windows використовує для доступу та керування своїми об'єктами. Звичайно, програми користувача (ужитки) в захищеному режимі не мають прав доступу до цих таблиць. Тому, коли необхідно отримати чи змінити дані, що пов'язані з певним об'єктом Windows, застосування використовує відповідну функцію API з параметром хендла цього об'єкту. Таким чином Windows забезпечує захист своїх даних при роботі у багатозадачному режимі.

КОНТРОЛЬНІ ПИТАННЯ

1. Який формат виконавчих файлів у програм написаних під Windows?
2. Що таке Win32 API?
3. Особливості виклику функцій API?
4. Які функції API ви знаєте?
5. Що таке дескриптор, хендл, атом?

ЛІТЕРАТУРА

1. Эпплман Д. Win32 API и Visual Basic. Для профессионалов (+CD). - СПб.: Питер, 2001. - 1120 с.: ил.
2. Юров В. Assembler: учебник. - СПб.: Питер, 2001. - 624 с.: ил.

ЗАВДАННЯ

Реалізувати програму, що викликає Win32 API функцію згідно варіанту заданого табл. 5.1 і виводить результат її роботи на екран.

Таблиця 5.1.

Варіанти завдань

№ варіанту	Функція API
1.	GetUserNameA
2.	GetWindowsDirectoryA
3.	GetSystemDirectoryA
4.	GetTempPathA
5.	GetCurrentDirectoryA
6.	GetDriveTypeA
7.	GetVersionEx
8.	GetOEMCP
9.	GetOACP
10.	DeleteFileA
11.	GetDiskFreeSpaceA
12.	GetFileAttributesA
13.	GetFileSize
14.	GetFullPathNameA
15.	GetShortPathNameA
16.	MoveFileA
17.	RemoveDirectoryA
18.	GetDoubleClickTime
19.	CharLowerA
20.	CharUpperA
21.	IsCharLowerA
22.	IsCharUpperA
23.	GetSystemDirectoryA
24.	GetTempPathA
25.	GetCurrentDirectoryA
26.	GetUserNameA

27.	GetSystemTime
28.	GetLocalTime
29.	GetSysColor
30.	GetComputerNameA

ПОРЯДОК ВИКОНАННЯ

1. Набрати подану нижче програму для визначення імені комп'ютера (яке задається системним адміністратором при установці операційної системи), зберегти її у файлі з розширенням “.asm”.
2. Відкомпілювати за допомогою MASM32 набрану програму та запустити одержаний exe-файл на виконання. Записати ім'я комп'ютера, яке отримала програма.
3. Розглянути текст програми, вивчити загальну структуру програми із застосуванням функцій Win32 API.
4. Замінити виклик функції API GetComputerNameA на виклик однієї з функцій (згідно варіанту), змінивши відповідним чином параметри.
5. Про параметри функцій можна дізнатися з довідкового файлу GUIDE, який є додатком до Лабораторної роботи. Для отримання довідки необхідно запустити файл GUIDE, вибрати розділ “index” (вказівник), набрати назву функції без останньої букви “A”. Буква “A” вказує лише, що дана функція працює з символами, які представляються в пам'яті комп'ютера одним байтом. Всі параметри, які будуть вказані в довіднику, необхідно засилати в стек, починаючи з останнього. Якщо назва параметру починається з наступних букв, то його довжина вказана у таблиці:

Перші символи	Довжина параметра	Зміст параметра
H	4	Хендл об'єкту
Lp	4	Адреса об'єкту (offset)
N	4	Змінна або адреса змінної (4 байти)
U або B	4	Ідентифікатори типу BOOL або прапорці
W	2	Ціле 16-ти бітне число
D	4	Ціле 32х бітне число
Short	2	Ціле 16ти бітне число
Long	4	Ціле 32х бітне число

6. Створити exe-файл модифікованої програми та продемонструвати його роботу.

7. Підготувати та захистити звіт. В звіті *обов'язково* мають бути описані параметри виклику функції.

Програма визначення назви комп'ютера з використанням функції Win32 API

```
.586                                ; для процесора не нижче INTEL-586
.model flat, STDCALL                ; компілювати як програму для WIN32
option casemap :none                ; код чутливий до регістру літер
; Визначення зовнішніх процедур:
include \masm32\include\windows.inc ; завжди першим
include \masm32\macros\macros.asm   ; підтримка макросів MASM
; -----
; підключення файлів з форматами прототипів виклику функцій
include \masm32\include\masm32.inc
include \masm32\include\gdi32.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
; -----
; підключення заголовків бібліотек експортованих функцій
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\gdi32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
.data                                ; вміст сегменту даних:
buflen dd 256                        ; визначення комірки пам'яті
hello_title db ' Лабораторна робота № 5 ', 0
hello_message db 'ComputerName: '    ; рядок байтів
user_name db 256 dup (0)              ; буфер заповнений нулями
.code                                ; вміст сегменту коду:
Start: ; формування параметрів для виклику заданої функції
push offset buflen                    ; 2-й параметр: адреса buflen
push offset user_name                 ; 1-й параметр: адреса user_name
call GetComputerNameA                 ; виклик функції API
; формування параметрів вікна для відображення результату
push 40h                              ; стиль вікна - одна кнопка "OK"
                                        ; з піктограмою "i"

push offset hello_title                ; адреса рядка із заголовком
push offset hello_message              ; адреса рядка з повідомленням
push 0                                ; хендл програми-власника вікна
call MessageBoxA                       ; виклик функції API
push 0                                  ; код виходу з програми
call ExitProcess                       ; завершення програми
end Start                              ; закінчення сегменту коду
```

ЛАБОРАТОРНА РОБОТА №6.

СТВОРЕННЯ DLL ТА ЇХ ВИКОРИСТАННЯ ПРИ НЕЯВНОМУ ЗВ'ЯЗУВАННІ НА МОВІ C

Мета: Ознайомитись з технологією та оволодіти навиками створення та використання бібліотек динамічного компонування з використанням неявного зв'язування.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Завершальним етапом створення програмного продукту є процес збирання (компонування) завантажувального модуля (.exe – файлу). *Компонуванням* (linking) називають процес створення фізичного або логічного виконуваного файлу (модуля) із набору об'єктних файлів бібліотек для подальшого виконання або під час виконання і вирішення проблеми неоднозначності імен, що виникає при цьому.

У разі створення фізичного виконуваного файлу для подальшого виконання компонування називають *статичним*, коли у такому файлі міститься все потрібне для виконання програми. У випадку створення логічного виконуваного файлу під час виконання програми компонування називають *динамічним*, у цьому випадку образ виконуваного модуля збирають “на ходу”.

Статичне компонування виконуваних файлів має низку недоліків:

- якщо кілька застосунків використовують спільний код (наприклад, код бібліотеки мови C), кожний виконуваний файл міститиме окрему копію цього коду в результаті такі файли займатимуть значне місце на диску і у пам'яті;
- під час кожного оновлення застосування, його потрібно наново перекомпонувати і перевстановити;
- неможливо реалізувати динамічне завантаження програмного коду під час виконання.

Для вирішення цих і подібних проблем було запропоновано концепцію *динамічного компонування* із використанням динамічних або розділюваних бібліотек.

Динамічна бібліотека (англ. Dynamic-Load Library — динамічно завантажувана бібліотека) - набір функцій, скомпонованих разом у вигляді бінарного файлу, який може бути динамічно завантажений в адресний простір

процесу, що використовує ці функції. *Динамічне завантаження*- завантаження під час виконання процесу. *Динамічне компонування* - компонування образу виконуваного файла під час виконання процесу із використанням динамічних бібліотек.

До переваги використання динамічних бібліотек, слід віднести:

- оскільки бібліотечні функції містяться в окремому файлі, розмір виконуваного файла стає меншим;
- якщо динамічну бібліотеку використовують кілька процесів, у пам'ять завантажують лише одну її копію, після чого сторінки коду бібліотеки відображаються в адресний простір кожного з цих процесів;
- оновлення застосування може бути зведене до встановлення нової версії динамічної бібліотеки без необхідності перекомпонування тих його частин, які не змінилися;
- динамічні бібліотеки дають змогу застосуванню реалізовувати динамічне завантаження модулів на вимогу;
- динамічні бібліотеки дають можливість спільно використовувати ресурси застосування;
- оскільки динамічні бібліотеки є двійковими файлами, можна організувати спільну роботу бібліотек, розроблених із використанням різних мов програмування.

Використання динамічних бібліотек не позбавлене недоліків:

- динамічне зв'язування сповільнює завантаження застосування. Що більше таких бібліотек потрібно процесу, то більше файлів треба йому відобразити у свій адресний простір під час завантаження, а відображення кожного файла забирає час;
- при відсутності спільного використання динамічної бібліотеки іншими застосування зовнішня пам'ять може використовуватися не ефективно. На відміну від статичного зв'язування, коли зі загальної бібліотеки вибираються тільки ті функції, що використовуються застосування, при використанні динамічного зв'язування до застосування необхідно додавати повну версію бібліотеки, навіть, якщо використовуються тільки декілька функцій. При значних обсягах бібліотеки втрати пам'яті відчутні;
- найбільшою проблемою у використанні динамічного компонування є проблема зворотної сумісності динамічних бібліотек. Ця проблема виникає в ситуації, коли застосування встановлює нову версію DLL поверх попередньої. Якщо нова версія не має зворотної сумісності із попередніми, застосування, розраховані на використання попередніх версій бібліотеки, можуть припинити роботу;

- ускладнюється процес інсталювання програмного продукту, в процесі якого необхідно досліджувати які з DLL вже інстальовано в ОС і які їх версії. Інстальована нова версія DLL з некоректною зворотною сумісністю, може негативно вплинути на виконання інших програм, що її використовують і навіть не підозрюють про підміну бібліотеки.

Використання DLL у Windows

Загальні бібліотеки функцій в ОС Windows реалізовані компанією Microsoft за DLL технологією. Як правило, ці бібліотеки мають розширення файлу *.dll, *.ocx (для бібліотек, що містять елементи керування ActiveX) або *.drv (драйвери старих версій ОС). Структура DLL така сама, як і в PE-файлів (Portable Executable) для 32-, 64-розрядних Windows, та New-Executable (NE) для 16-бітових Windows.

DLL може містити 2 типи функцій: експортні та внутрішні. Експортні функції можуть бути викликані зі зовнішніх прикладних та визначаються за допомогою ключового слова `__declspec(dllexport)`. Внутрішні – це функції, які використовуються в середині DLL і не можуть бути викликані ззовні.

DLL є модулем (module). Тобто, вона складається з: сегментів коду, сегментів ресурсів та одного сегменту даних. Крім цього DLL може містити точку входу. Точка входу – це функція `DllMain`, яка викликається при завантаженні або вивантаженні бібліотеки потоком або процесом. Ця функція має наступний прототип:

```
BOOL WINAPI DllMain(
    HANDLE hModule,           // Хендл DLL модуля
    DWORD ul_reason_for_call, // Причина виклику
    LPVOID lpReserved );     // Зарезервовано
```

Якщо `DllMain` повертає `FALSE`, то бібліотека вважається такою, що не завантажилася. При неявному зв'язуванні це призведе до відмови запуску програми, а при явному – помилки завантаження лише цієї бібліотеки.

У процесі виконання вміст бібліотеки залишається незмінним (сегменти коду та сегменти ресурсів), що дозволяє завантажувати її в пам'ять в єдиному примірнику і використовувати багатьма завданнями одночасно. Використання `dll` дозволяє економити пам'ять, забезпечити модульність програм, полегшити процес встановлення програм.

Можливі 2 способи використання динамічних бібліотек. Вони називаються “явним” та “неявним” зв'язуванням. “Явне” та “неявне” зв'язування бібліотеки з програмою мають суттєві відмінності в процесі написання та компіляції програми.

Неявне зв'язування бібліотеки з програмою (Load-time dynamic linking) полягає в тому, що бібліотека (яка міститься у файлі з розширенням .dll) завантажується в пам'ять в момент завантаження програми. При відсутності бодай однієї з бібліотек при запуску програми відбудеться збій та припинення виконання програми.

Щоб реалізувати неявне зв'язування необхідно до проекту програми включити прототипи функцій, що містяться в бібліотеці та бібліотеку імпорту (має розширення .lib). На даному етапі наявність файлу з розширенням .dll не є необхідною. При компоновці створюється виконавчий файл, який містить код, що забезпечує систему інформацією, яка необхідна для автоматичного завантаження бібліотеки з .dll файлу та інформацією, яка необхідна для зв'язування імен функцій у програмі з їх адресами у бібліотеці.

Неявне зв'язування дозволяє здійснювати виклик функцій з бібліотеки написанням коду програми в стилі притаманному мовам C/C++.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке DLL?
2. Що може міститися у DLL?
3. Переваги та недоліки статичного та динамічного зв'язування.
4. Що таке неявне зв'язування?
5. Як реалізувати неявне зв'язування?
6. Що необхідно мати для реалізації неявного зв'язування?

ЛІТЕРАТУРА

1. Эпплман Д. Win32 API и Visual Basic. Для профессионалов (+CD). - СПб.: Питер, 2001. - 1120 с.: ил.
2. Юров В. Assembler: учебник. - СПб.: Питер, 2001. - 624 с.: ил.
3. Джонсон М. Харт Системное программирование в среде Windows. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 592 с.

ПОРЯДОК ВИКОНАННЯ

Написання програми складається з 2-ох етапів. На першому етапі створюється бібліотека. На другому етапі – створюється програма, яка викликатиме функції з створеної бібліотеки.

1. Для створення бібліотеки створюємо новий проект типу Win32 Project та в налаштуваннях вибираємо тип проекту DLL, так як показано на рис. 6.1.

2. Створюємо заголовний файл бібліотеки (з розширенням .h) для забезпечення експортування інтерфейсів функцій відповідно до варіанту завдання. Приклад файлу наведено нижче.

Заголовний файл (dlltest.h)

```
#ifndef _DLLTEST_H_
#define _DLLTEST_H_

#include <iostream>
#include <stdio.h>
#include <windows.h>
using namespace std;

extern "C" __declspec(dllexport) void NumberList();
extern "C" __declspec(dllexport) void LetterList();
#endif
```

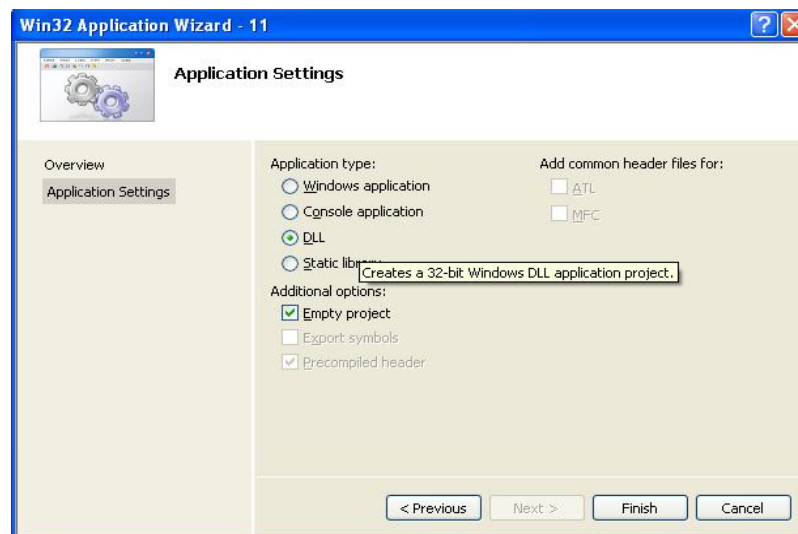


Рис.6.1. Вибір типу проекту для створення DLL засобами MSVS.

3. Створюємо .cpp файл з вмістом коду бібліотеки відповідно до варіанту завдання. Приклад файлу наведено нижче.

Код бібліотеки (dlltest.cpp)

```
#include "dlltest.h"

#define MAXMODULE 50
char module[MAXMODULE];
extern "C" __declspec(dllexport) void NumberList()
{
    GetModuleFileName(NULL, (LPTSTR)module, MAXMODULE);
    cout << "This function was called from " << module << endl;
    cout << "NumberList():";
    for(int i=0; i<10; i++)
    {
        cout << i << " ";
    }
}
```

```

    cout << endl << endl;
}

extern "C" __declspec(dllexport) void LetterList()
{
    GetModuleFileName(NULL, (LPTSTR)module, MAXMODULE);
    cout << "This function was called from " << module << endl;
    cout << "LetterList(): ";
    for(int i=0; i<26; i++)
    {
        cout << char(97 + i) << " ";
    }
    cout << endl << endl;
}

```

Бібліотека містить 2 функції, які виводять текст на консоль. Рядок

```
extern "C" __declspec(dllexport)
```

означає, що функцію буде видно зовні DLL (тобто її можна викликати з іншої програми).

Після компіляції ми одержимо DLL – бібліотеку, яка складатиметься з 2-ох файлів: dlltest.dll, dlltest.lib. Зауважимо, що дана бібліотека, хоч і не має явно оголошеної точки входу, проте функція DllMain все ж таки буде створена неявно, хоч нічого і не робитиме.

4. На другому етапі створюємо новий проект типу Win32 Project та в налаштуваннях вибираємо тип проекту Console Application. Додаємо до проекту заголовочний файл бібліотеки, який створений на кроці 2. Підключаємо створену бібліотеку імпорту в властивостях проекту як показано на рис. 6.2.

5. Пишемо програму в стилі C/C++, яка здійснюватиме виклик функцій створених відповідно до варіанту завдання з бібліотеки. Компілюємо і запускаємо програму. Приклад виклику функцій при неявному зв'язуванні для тестового прикладу наведено нижче.

```

#include <conio.h>
#include "dlltest.h"

void main()
{
    NumberList();
    LetterList();

    getch();
}

```

6. Готуємо та захищаємо звіт.

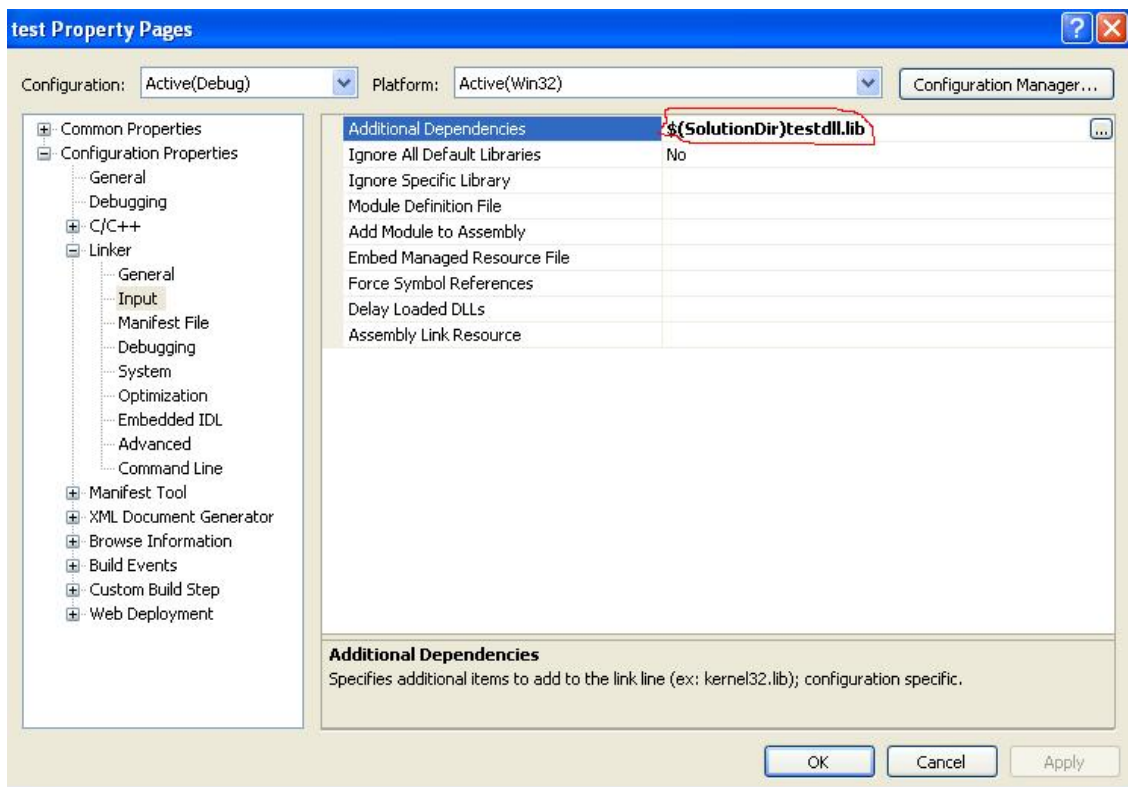


Рис. 6.2. Підключення бібліотеки імпорту в проект.

ВАРІАНТИ ЗАВДАНЬ

1. Підрахувати кількість слів у заданому рядку тексту.
2. Підрахувати кількість букв а в останньому слові заданого рядка тексту.
3. Знайти кількість слів у тексті, що починаються з заданої літери.
4. Знайти кількість слів у тексті, в яких перший і останній символи співпадають між собою.
5. Знайти довжину найдовшого слова в тексті.
6. Циклічно переставити букви в словах тексту так, що i -а буква слова стала $i+1$ -ою, а остання - першою.
7. У кожному слові тексту замінити букву "а" на букву "е", якщо "а" стоїть на парному місці, і замінити букву "б" на поєднання "ак", якщо "б" стоїть на непарному місці.
8. Здійснити заміну заданого слова тексту іншим (врахувати, що слова можуть мати різну довжину!).
9. Задано текст, що містить від 2 до 30 слів, в кожному з яких від 2 до 10 латинських букв; між сусідніми словами - не менше одного пропуску. Надрукувати всі слова, відмінні від останнього слова, заздалегідь

- перетворивши кожне з них за наступним правилом: 1) перенести першу букву в кінець слова; 2) перенести останню букву в початок слова.
10. Відредагувати заданий текст, видаляючи з нього всі слова з непарними номерами.
 11. Відредагувати заданий текст, перевертаючи слова з парними номерами. Наприклад, HOW DO YOU DO -> HOW OD YOU OD.
 12. Надрукувати всі слова тексту, відмінні від останнього слова.
 13. Перетворити текст залишити в словах тільки перші входження кожної букви.
 14. Перетворити текст видаливши середню букву в словах непарної довжини.
 15. Підрахувати суму місць, на яких в словах тексту стоїть задана буква.
 16. Скласти таблицю слів заданого тексту, що починаються з букви "A", з вказівкою числа повторень кожного слова.
 17. Перетворити текст шляхом викреслювання із слів всіх букв, що стоять на непарних місцях.
 18. Створити функцію для перетворення арабських чисел в римські та навпаки. Наприклад, 255 = CCLV = сто + сто + п'ятдесят + п'ять. Зауваження. Подібними алгоритмами перекладу чисел з однієї системи в іншу ми користуємося по декілька разів на дню, коли ведемо грошові розрахунки. Сума грошей - це арабське число, якому відповідає певний набір банкнот і монет (аналоги римських цифр).
 19. Підрахувати, скільки букв треба виправити в слові X, щоб вийшло слово Y (X, Y - слова однакової довжини).
 20. Підрахувати число однакових букв в словах X і Y рівної довжини, що стоять на одних і тих же місцях.
 21. Визначити, скільки заданих символів міститься в тексті, що вводиться з клавіатури.
 22. З клавіатури вводиться текст. Підрахувати і вивести на друк кількість слів тексту, що починаються з голосної.
 23. Для заданого тексту визначити довжину максимальної серії символів, відмінних від латинських букв, що міститься в ньому.
 24. Створити програму, що з'ясовує, чи можна з букв слова X скласти слово Y.
 25. Ввести два рядки тексту та ціле число N, порівняти між собою перші N символів в рядках, ігноруючи пробіли.
 26. Ввести два рядки тексту, вилучити з довшого рядка всі входження коротшого рядка. Вивести новий рядок на екран.
 27. Ввести два рядки тексту, провести "склейку" першого рядка до другого та вивести новий рядок на екран.

28. Ввести рядок тексту, змінити регістр кожного символу, що є літерою і вивести новий рядок на екран.
29. Ввести рядок тексту та два окремі символи. Вилучити з вхідного рядка всі другі символи після того як зустрінеться перший символ. Вивести новий рядок на екран.
30. Ввести рядок тексту та один символ. Вилучити з вхідного рядка всі слова, що починаються на заданий символ. Вивести новий рядок на екран.

ЛАБОРАТОРНА РОБОТА №7.

СТВОРЕННЯ DLL ТА ЇХ ВИКОРИСТАННЯ ПРИ ЯВНОМУ ЗВ'ЯЗУВАННІ НА МОВІ АСЕМБЛЕР

Мета: Ознайомитись з технологією та оволодіти навиками створення та використання бібліотек динамічного компонування з використанням явного зв'язування на мові Асемблера.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Динамічне компонування образу задачі в процесі її виконання надає ряд переваг розробникам програмного забезпечення в порівнянні зі статичним копуванням. До переваг відносяться:

- зменшується розмір виконуваного файлу;
- у пам'ять завантажують лише одну копію динамічної бібліотеки;
- оновлення бібліотек не веде до перекомпилювання застосування;
- реалізовується динамічне завантаження модулів на вимогу;
- можливість спільно використовувати ресурси застосування;
- можна організувати спільну роботу бібліотек, розроблених із використанням різних мов програмування.

Однак воно не позбавлене недоліків:

- сповільнює завантаження застосування;
- не ефективно використовуватися зовнішня пам'ять;
- проблема є зворотної сумісності;
- ускладнюється процес інсталювання програмного застосування.

Однак грамотно володіючи технологією створення та підтримки динамічних бібліотек можна зменшити вплив деяких з цих недоліків на програмний продукт, а деякі подолати.

Можливі 2 способи використання динамічних бібліотек. Вони називаються “явним” та “неявним” зв'язуванням. “Явне” та “неявне” зв'язування бібліотеки з програмою мають суттєві відмінності в процесі написання та компіляції програми.

Неявне зв'язування бібліотеки з програмою (Load-time dynamic linking) полягає в тому, що бібліотека (яка міститься у файлі з розширенням .dll) завантажується в пам'ять в момент завантаження програми. До переваг “неявного” зв'язування в порівнянні з “явним” відноситься:

- простота програмування. Розробник не вникає в проблеми зв'язування назв функцій з адресами за якими завантажена їх реалізація;
- прогнозованість поведінки застосування. Якщо застосування успішно завантажено, то усі проблеми перехрестних зв'язків уже вирішено; Однак у “неявного” зв'язування існує ряд недоліків:
- при відсутності бодай однієї з бібліотек при запуску програми відбудеться збій та припинення виконання програми;
- значні затрати часу на завантаження та старт застосування, пов'язані з необхідністю завантаження усіх динамічних бібліотек;
- відсутня можливість вивантаження непотрібних в даний час динамічних бібліотек;
- на час компонування необхідно мати додаткові файли з прототипами функцій та бібліотеку імпорту (.lib).

Отже, основними перевагами “явного” зв'язування, є можливість тонко керувати процесами завантаження та вивантаження динамічних бібліотек, а отже і використовуваною пам'яттю, хоча і за рахунок складності програмування.

“Явне” зв'язування бібліотеки з програмою (Run-time dynamic linking) полягає в тому, що бібліотека (яка міститься у файлі з розширенням .dll) завантажуються в пам'ять в момент часу, що визначається розробником, за допомогою виклику API функцій LoadLibrary або LoadLibraryEX. При успішному виконанні функція повертає адресу точки входу. При відсутності бібліотеки, яку необхідно завантажити, або при помилках її завантаження функція поверне NULL, а сама програма, може продовжити виконання. Звичайно, якщо функції, що містяться у відсутній бібліотеці не є критичними для її подальшої роботи.

Для виклику бібліотечної функції необхідно оголосити вказівник на функцію, та присвоїти йому адресу бібліотечної функції. Для цього необхідно використати API функцію GetProcAddress, яка повертає адресу вказаної їй у параметрі бібліотечної функції.

Після завершення роботи з функціями бібліотек, програмі необхідно вивантажити бібліотеки за допомогою функції FreeLibrary.

Для успішної компіляції необхідно мати лише dll файл бібліотеки. Запуск програми відбудеться навіть за відсутності бібліотечного файлу, оскільки його наявність при використанні “явного” зв'язування не перевіряється.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке DLL?

2. Що може міститися у DLL?
3. Що таке “явне” зв’язування?
4. Які переваги “явного” зв’язування у порівнянні з “неявним”?
5. Як реалізувати “явне” зв’язування?
6. Що необхідно мати для реалізації “явного” зв’язування?

ЛІТЕРАТУРА

1. Эпплман Д. Win32 API и Visual Basic. Для профессионалов (+CD). - СПб.: Питер, 2001. - 1120 с.: ил.
2. Юров В. Assembler: учебник. - СПб.: Питер, 2001. - 624 с.: ил.
3. Джонсон М. Харт Системное программирование в среде Windows. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 592 с.

ПОРЯДОК ВИКОНАННЯ

Написання програми складається з 2-ох етапів. На першому етапі створюється Labor07.dll бібліотека. На другому етапі – створюється програма, яка викликатиме функції зі створеної бібліотеки.

1. Для створення бібліотеки створюємо новий проект типу Win32 Project та в налаштуваннях вибираємо тип проекту DLL, так як показано на рис. 7.1:

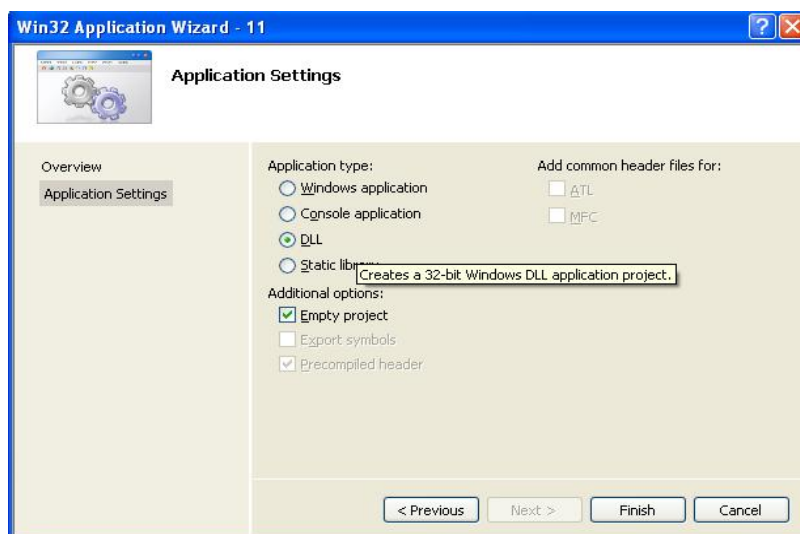


Рис.7.1. Вибір типу проекту для створення DLL засобами MSVS

2. Створюємо файл з описом бібліотеки (з розширенням *.def) для забезпечення експортування інтерфейсів функцій. Приклад файлу наведено нижче.

Файл опису бібліотеки (Labor07.def)

```
;-----
;                               Labor06.def
```

```

;-----
LIBRARY    Labor06          ; назва бібліотеки
EXPORTS
    TestFunction          ; назви функцій її інтерфейсу

```

3. Створюємо .asm файл з вмістом коду бібліотеки. Приклад файлу наведено нижче.

Код бібліотеки (Labor07.asm)

```

.data
    tit db "Labor #6, asm dll function",0

.code
;----- Стартова функція -----
DllEntry PROC hInstDLL:DWORD, reason:DWORD, reserved:DWORD
    mov  eax, 1
    ret
DllEntry ENDP

;----- Тестова функція в бібліотеці -----
TestFunction PROC msg:DWORD
    invoke MessageBoxA, 0, msg, addr tit, 0
    ret
TestFunction ENDP

End DllEntry

```

Бібліотека містить 2 функції. Перша – точка входу в бібліотеку, друга виводять текст у вікно повідомлень.

Після компіляції ми одержимо DLL – бібліотеку, яка складатиметься з 2-ох файлів: Labor07.dll, Labor07.lib.

4. На другому етапі створюємо новий проект типу Win32 Project та в налаштуваннях вибираємо тип проекту Console Application.

5. Створюємо програму, яка здійснюватиме виклик функцій з бібліотеки як показано нижче. Компілюємо і запускаємо програму.

```

;-----
;                               Labor07_e.asm
;-----
.386
.model flat, stdcall
option casemap:none
.data

```

```

LibName db "Labor06.dll",0
FunctionName db "TestFunction",0
DllNotFound db "Cannot load library",0
AppName db "Load explisit Library",0
NotFound db "TestFunction function not found",0
msg db "hello it explisit dll function call",0

.data?
hLib dd ?
TestFunctionAddr dd ?

.code
start:
    invoke LoadLibrary,addr LibName    ; завантаження бібліотеки
    .if eax == NULL                    ; якщо завантаження не вдалося
        invoke MessageBox,NULL,addr DllNotFound,addr AppName,MB_OK
    .else
        mov hLib,eax                  ; збереження заголовку бібліотеки
        ; визначення адреси функції
        invoke GetProcAddress,hLib,addr FunctionName
        .if eax == NULL                ; якщо не вдалося взяти адресу
            invoke MessageBox,NULL,addr NotFound,addr AppName,MB_OK
        .else
            push offset msg
            mov TestFunctionAddr,eax
            call [TestFunctionAddr]      ; виклик функції
        .endif
        invoke FreeLibrary,hLib         ; вивантаження бібліотеки
    .endif
    invoke ExitProcess,NULL
end start
;-----

```

6. Створюємо власну DLL бібліотеку для реалізації функцій згідно варіанту.

7. Створюємо програму, що використовує явне зв'язування DLL.

8. Готуємо та захищаємо звіт.

ВАРІАНТИ ЗАВДАНЬ

1. Підрахувати кількість слів у заданому рядку тексту.
2. Підрахувати кількість букв а в останньому слові заданого рядка тексту.
3. Знайти кількість слів у тексті, що починаються з заданої літери.
4. Знайти кількість слів у тексті, в яких перший і останній символи співпадають між собою.
5. Знайти довжину найдовшого слова в тексті.

6. Циклічно переставити букви в словах тексту так, що i -а буква слова стала $i+1$ -ою, а остання - першою.
7. У кожному слові тексту замінити букву "а" на букву "е", якщо "а" стоїть на парному місці, і замінити букву "б" на поєднання "ак", якщо "б" стоїть на непарному місці.
8. Здійснити заміну заданого слова тексту іншим (врахувати, що слова можуть мати різну довжину!).
9. Задано текст, що містить від 2 до 30 слів, в кожному з яких від 2 до 10 латинських букв; між сусідніми словами - не менше одного пропуску. Надрукувати всі слова, відмінні від останнього слова, заздалегідь перетворивши кожне з них за наступним правилом: 1) перенести першу букву в кінець слова; 2) перенести останню букву в початок слова.
10. Відредагувати заданий текст, видаляючи з нього всі слова з непарними номерами.
11. Відредагувати заданий текст, перевертаючи слова з парними номерами. Наприклад, HOW DO YOU DO -> HOW OD YOU OD.
12. Надрукувати всі слова тексту, відмінні від останнього слова.
13. Перетворити текст залишити в словах тільки перші входження кожної букви.
14. Перетворити текст видаливши середню букву в словах непарної довжини.
15. Підрахувати суму місць, на яких в словах тексту стоїть задана буква.
16. Скласти таблицю слів заданого тексту, що починаються з букви "А", з вказівкою числа повторень кожного слова.
17. Перетворити текст шляхом викреслювання із слів всіх букв, що стоять на непарних місцях.
18. Створити функцію для перетворення арабських чисел в римські та навпаки. Наприклад, 255 = CCLV = сто + сто + п'ятдесят + п'ять. Зауваження. Подібними алгоритмами перекладу чисел з однієї системи в іншу ми користуємося по декілька разів на день, коли ведемо грошові розрахунки. Сума грошей - це арабське число, якому відповідає певний набір банкнот і монет (аналоги римських цифр).
19. Підрахувати, скільки букв треба виправити в слові X, щоб вийшло слово Y (X, Y - слова однакової довжини).
20. Підрахувати число однакових букв в словах X і Y рівної довжини, що стоять на одних і тих же місцях.
21. Визначити, скільки заданих символів міститься в тексті, що вводиться з клавіатури.
22. З клавіатури вводиться текст. Підрахувати і вивести на друк кількість слів тексту, що починаються з голосної.

23. Для заданого тексту визначити довжину максимальної серії символів, відмінних від латинських букв, що міститься в ньому.
24. Створити програму, що з'ясовує, чи можна з букв слова X скласти слово Y.
25. Ввести два рядки тексту та ціле число N, порівняти між собою перші N символів в рядках, ігноруючи пробіли.
26. Ввести два рядки тексту, вилучити з довшого рядка всі входження коротшого рядка. Вивести новий рядок на екран.
27. Ввести два рядки тексту, провести "склейку" першого рядка до другого та вивести новий рядок на екран.
28. Ввести рядок тексту, змінити регістр кожного символу, що є літерою і вивести новий рядок на екран.
29. Ввести рядок тексту та два окремі символи. Вилучити з вхідного рядка всі другі символи після того як зустрінеться перший символ. Вивести новий рядок на екран.
30. Ввести рядок тексту та один символ. Вилучити з вхідного рядка всі слова, що починаються на заданий символ. Вивести новий рядок на екран.

НАВЧАЛЬНЕ ВИДАННЯ

СИСТЕМНЕ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання циклу лабораторних робіт з дисципліни
“Системне програмування” для студентів базового напрямку
6.050102 “Комп’ютерна інженерія”

Укладачі:

Мархивка Василь Степанович
Олексів Максим Васильович
Мороз Іван Володимирович
Акимишин Орест Ігорович

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70х100/16. Папір офсетний. Друк на різнографі
Умовн. друк. арк. Обл.-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул. Ф. Колесси, 2. Львів, 79000