Міністерство транспорту та зв'язку України

Державний департамент з питань зв'язку та інформатизації
Одеська національна академія зв'язку ім. О. С.Попова

Кафедра інформаційних технологій

О.Г. Трофименко

Програмування та алгоритмічні мови

МОДУЛЬ № 4
Організація обчислювальних процесів засобами вбудованого в C++ Builder асемблера

Частина 2

Методичні вказівки для практичних і лабораторних робіт
для студентів академії
зі спеціальності
«Автоматизоване управління технологічними процесами»

ЗАТВЕРДЖЕНО
методичною радою ІМ

Протокол № 10 від 5.04.2007 р.

Одеса 2007

УДК 004.43 План МНВ 2007 р.

Укладач: **О.Г. Трофименко**

Розглянуто основні засоби застосовування асемблера, вбудованого в Borland C++ Builder.

Посібник призначено для підготовки до лабораторних робіт модуля 4 «Організація обчислювальних процесів засобами вбудованого в C++ Builder асемблера» з дисципліни «Програмування та алгоритмічні мови». Передбачено можливість виконання п’яти лабораторних робіт, в кожній з яких, розміщено основні теоретичні відомості й наведено приклади програм з аналізом програмного коду.

УХВАЛЕНО
на засіданні кафедри ІТ

Протокол № 6 від 9.02.2007 р.

Структура модуля 4

Дисципліна “Програмування та алгоритмічні мови” вивчається студентами академії зі спеціальності “Автоматизоване управління технологічними процесами” у II та III семестрах першого та другого курсу навчання.

Вивчення дисципліни розраховано на чотири модулі:

модуль 1 – Програмування задач опрацювання масивів в C++;

модуль 2 – Організація бібліотек функцій опрацювання числових і символьних масивів в C++ Builder;

модуль 3 – Файли в C++. Елементи об’єктно-орієнтованого програмування;

модуль 4 – Організація обчислювальних процесів засобами вбудованого в C++ Builder асемблера.

Модуль 4 присвячено вивченню основних засобів застосовування вбудованого у Borland C++ Builder асемблера, оскільки застосовування асемблера, вбудованого у C++, надає можливість збільшити швидкодію програм і набути доступ до найнижчих рівнів керування обладнанням.

Метою вивчення модуля 4 є формування у студентів таких знань і умінь:

знання з тем:

архітектура мікропроцесора INTEL. Регістри процесора;

формат мови Асемблер. Використання асемблерних вставок в C++ Builder;

арифметичні команди опрацювання регістрів процесора CPU і співпроцесора FPU;

уміння:

використовування асемблерних вставок в C++ Builder-додатках;

розробляння асемблерного програмного коду задля виконування арифметичних операцій.

У відповідності до навчального плану структура модуля має вигляд:

Вид занять	Кількість годин
Лекції	18
Лабораторні роботи	10
Разом аудиторного часу:	28
Індивідуальна й самостійна робота студентів	26
Разом годин:	54

Вступ

Розвиток сучасних інструментальних засобів проектування програм дозволяє розробникові програмного забезпечення здобути готову програму за допомогою кількох клацань миші. На тлі високої популярності мов високого рівня C, Pascal, Basic та ін. мова низького машинного рівня – асемблер – не втрачає своєї актуальності і продовжує широко використовуватись при розроблянні програм, оскільки асемблер є мовою, якою “розмовляє” процесор. А отже, зникнути ця мова може лише разом зі зникненням процесорів. І тому асемблер має фундаментальну перевагу перед мовами високого рівня: він є найшвидший. Більшість програмних додатків, працюючих в режимі реального часу, або написано цілковито на асемблері, або використовують у критичних ділянках коду асемблерні модулі.

Асемблерний код є більш компактний, аніж його аналог мовою високого рівня. У цьому легко переконатись, порівнявши дизасембльовані лістинги однієї й тієї самої програми, написаної асемблером та мовою високого рівня. А оскільки команд є менше, то вони й виконуються швидше. Тому програми, написані асемблером, як правило, мають високу швидкодію.

Можна розробляти як окремі модулі на асемблері, так і долучати їх до програм, створюваних мовами високого рівня. Також можна використовувати широкі можливості вбудованого асемблера, передбачені в багатьох мовах високого рівня. Вбудований асемблер дозволяє досягти максимального ефекту при оптимізуванні математичних виразів, програмних циклів опрацьовування масивів. Провідні фірми-виробники, такі як Microsoft та Borland, невпинно вдосконалюють вбудований асемблер.

Короткі швидкі програми мовою асемблер набувають застосовування там, де розміри коду та його швидкодія є критичними параметрами. Сферами застосовування таких програм є системи реального часу, системні утиліти та програми, а також драйвери пристроїв. Програми на асемблері керують як периферійними пристроями персонального комп’ютера (ПК), так і нестандартними пристроями, приєднаними до ПК.

В посібнику розглядаються основні засоби застосовування вбудованого в Borland C++ Builder асемблера. Сама мова C++ є вельми популярна через високу ефективність об’єктних кодів програм як за швидкодією, так і за місткістю пам’яті. Слід нагадати, що мова C бере свій початок від мови BCPL, яка була мовою макроасемблер. Для більшості програм компілятори C++ генерують компактний та швидкий машинний код, а розумне застосовування вбудованого в C++ асемблера надає можливість збільшувати швидкодію програм і набути доступу до найнижчих рівнів керування обладнанням.

Посібник призначається для підготовки до лабораторних робіт з дисципліни “Програмування та алгоритмічні мови”, які провадяться в межах модуля 4 “Організація обчислювальних процесів засобами вбудованого в C++ Builder асемблера”. Передбачено можливість виконання п’яти лабораторних робіт. Перед кожною лабораторною роботою розміщено докладні теоретичні відомості й наведено приклади програм з аналізом програмного коду.

До виконання лабораторної роботи допускається студент, який має самостійно підготовлений протокол лабораторної роботи. В протоколі студент має відповісти на контрольні запитання та розробити програму для розв’язання поставленого індивідуального завдання конкретної лабораторної роботи.

Результати виконаних обчислень заносяться студентом до протоколу. Правильність роботи програми та остаточних результатів перевіряється викладачем.

Лабораторна робота № 1

Арифметичні операції над цілими числами у вбудованому в C++ Builder асемблері

Мета роботи: Вивчення арифметичних команд мови асемблер та набуття навичок створювання програмних додатків C++ Builder із вбудованим асемблером.

1 Основні теоретичні відомості

1.1 Початкові відомості про вбудований в C++ Builder асемблер

Мова низького машинного рівня – асемблер – це мова, якою „розмовляє" процесор. Тому її фундаментальною перевагою перед мовами високого рівня є швидкість. Більшість програмних додатків, які працюють у режимі реального часу, або написано мовою асемблер, або використовують у критичних ділянках програмного коду асемблерні модулі.

Асемблер, вбудований в мови високого рівня, використовують для досягнення високої продуктивності роботи додатків. Зручність, вигідність та легкість використовування вбудованого асемблера важко переоцінити, оскільки він не потребує окремого компілювання, компонування блоків асемблерного коду. Це є надто важливо для швидкого розробляння та налагоджування програм.

Вбудована в програмний додаток C++ послідовність асемблерних команд розташовується поміж фігурних дужок {} після ключового слова asm. Схематично блок команд на асемблері виглядає так:

```
asm
{
    < команди асемблера >
}
```

1.2 Формат мови асемблер

1.2.1 Формат команд

Команди мови асемблер складаються з таких полів:

[ім'я або позначка] операція операнди [; коментар]

Ім'я або позначка може складатися з латинських літер, цифр та спеціальних знаків (., ?, _, @, $), але не може розпочинатися з цифри. Якщо використовується крапка (.), то вона повинна бути першим знаком. Позначка у полі команди відокремлюється від неї двокрапкою (:). Значенням позначки є її адреса.

Операція – мнемонічний код операції асемблера.

Операнди – один чи кілька операндів, які відокремлюються один від одного комами. Операнди можна задавати у три способи:

- безпосередньо задавати в самій команді числовим виразом чи константою.
- регістром процесора (AH, AL, BH, BL, CH, CL, DH, DL – 8-бітові регістри загального призначення; AX, BX, CX, DX – 16-бітові регістри загального призначення; EAX, EBX, ECX, EDX – 32-бітові регістри загального призначення; SP, BP, SI, DI – базові й індексні регістри; SS, DS, CS, ES – 16-бітові сегментні регістри).
- коміркою пам'яті – це або позначка, або змінна, або вираз, який містить змінні й безпосередньо операнди.

Коментар в асемблері відокремлюється від команди крапкою з комою. В асемблері, вбудованому в інші мови, наприклад в додатки C++ Builder, на запис коментарів поширюються правила цієї мови та програмного середовища.

Кожна команда записується в окремому рядку і може розпочинатися не з першої позиції. Поля команди відокремлюються одне від одного прогалинами чи знаками табуляції.

При написанні асемблерного коду великі й маленькі літири не відрізнюються, на відміну від мови C++. Тобто, наприклад, до регістру акумулятора можна звертатися і як AX, і як ax.

1.2.2 Форми подавання чисел

В асемблері припускаються такі форми подавання чисел:

- двійкове число – це послідовність 0 та 1, яка завершується літерою B. Наприклад, 1001B, 111011001B;
- десяткове ціле – послідовність цифр 0...9, яка м о ж е завершуватись літерою D. Наприклад, 1992 чи 1992D;
- шістнадцяткове число – послідовність 16-кових цифр 0, …, 9, A, ..., F, яка завершується літерою H. Якщо 16-кове число розпочинається з символу A … F, то перед ним слід обов'язково записати 0 для того, щоби відрізнити число 0AH від позначення регістра AH чи число 0ABCH – від символічного імені ABCH;
- десяткове число з плаваючою крапкою записується у вигляді мантиси і порядку. Порядок відокремлюється від мантиси літерою E. Наприклад, –15.693E2, 0.00567E-8;
- символьна константа – рядок символів поміж апострофами (') чи лапками (").

1.2.3 Змінні

Зазвичай змінні в асемблері оголошуються і набувають початкових значень у сегменті даних. Адреса цього сегмента зберігається в сегментному регістрі DS або ES. У вбудованому асемблері розмірність змінної та тип, тобто кількість байтів займаної нею пам'яті, визначають за директивами DB (Define Byte – визначити байт), DW (Define Word – визначити слово) і DD (Define Double Word – визначити подвійне слово). В табл. 1.1 наведено директиви оголошування даних із зазначенням розмірності й типу даних.

Таблиця 1.1 – Директиви оголошення даних

Директиви	Тип	Розмір (байт)	Використовування у співпроцесорі
DB	BYTE	1	8-розрядне ціле число або символьна змінна
DW	WORD	2	16-розрядне ціле число
DD	DWORD	4	32-розрядне ціле або дійсне число

Далі наведено приклад використовування директив DB, DW та DD:

```
asm
{ var1  DB   121        // Змінна var1  розмірністю в 1 байт із початковим значенням 121
  var2  DB   0FFH       // Змінна var2  розмірністю в 1 байт із початковим значенням 255
  var3  DB   1001b      // Змінна var3  розмірністю в 1 байт із початковим значенням 9
  var4  DW   0AH        // Змінна var4  розмірністю в 2 байти із початковим значенням 10
  var5  DD   7          // Змінна var5  розмірністю в 4 байти із початковим значенням 7
}
```

1.2.4 Оператори для запису виразів у операндах

В асемблері існують оператори чотирьох типів: арифметичні, відносин, логічні й атрибутивні. Вирази в операндах обчислюються при транслюванні з урахуванням пріоритетів операцій та використаних дужок. Оператори з однаковим пріоритетом обчислюються зліва направо.

Встановлено такий пріоритет операторів:

1) елементи в круглих, квадратних (непряма адресація) чи кутових дужках(<>); структурні змінні, оператори LENGTH (кількість елементів даних), SIZE (розмір даних у байтах);

2) атрибутивні оператори PTR (перевизначання типу), OFFSET (зсув адреси операнда), SEG (сегментна адреса). Наприклад, MOV AX, WORD PTR [BX][SI] – переслати до AX слово за адресою [BX+SI];

3) атрибутивні оператори HIGH, LOW – видають старший (молодший) байт значення виразу. Наприклад, MOV AH, HIGH A – видає старший байт змінної;

4) множення та поділ: *, / (поділ націло), MOD (остача від поділу націло, має знак діленого), SHL (зсув ліворуч, є еквівалентний до множення на 2 у відповідному ступені), SHR (зсув праворуч, є еквівалентний до поділу на 2 у відповідному ступені);

5) + (додавання), – (віднімання);

6) оператори відносин: EQ *(рівність), NE (нерівність), LT (менше), GT (більше), LE (менше чи дорівнює), GE (більше чи дорівнює);

7) NOT (не) – логічний оператор, який інвертує всі біти операнда;

8) AND (логічне "І") ;

9) OR (логічне "ЧИ"), XOR (виняткове "ЧИ" чи сума за модулем 2);

10) SHORT ("короткий") – для модифіковування атрибута NEAR, який вказує, що значення позначки перебуває в інтервалі (–128, 127) байтів від адреси команди, яка слідує за командою JMP. Це дозволяє здобути більш економний код.

Формат оператора: SHORT <позначка>

1.2.5 Команди пересилання даних

MOV <операнд1>, <операнд2>

Базова команда пересилання даних копіює вміст операнда2 в операнд1, операнд2 не змінюється. Команда MOV AX,BX є аналогічна до оператора BX=AX в С. Як операнд2 можуть використовуватись: число (безпосередній операнд), регістр чи змінна; як операнд1: регістр чи змінна. Не можна виконувати пересилання даних безпосередньо з однієї змінної до іншої чи з одного сегментного регістра до іншого: ці операції виконують двома командами MOV через регістр загального призначення чи за допомогою стека командами PUSH/POP.

PUSH <операнд>

Завантажити операнд до стека. За операнд можуть слугувати регістр, змінна чи безпосередній операнд.

POP <операнд>

Завантажити дані стека до операнда, тобто виконати дію, зворотну до команди PUSH. За операнд можуть слугувати регістр чи змінна.

PUSHA чи **PUSHAD**

Команда PUSHA завантажує до стека вміст восьми регістрів у такому порядку: AX, CX, DX, BX, SP, BP, SI, DI; команда PUSHAD – EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

POPA чи **POPAD**

Завантажити дані зі стека до восьми регістрів, тобто виконати дії, зворотні командам PUSHA/POPAD.

XCHG <операнд1>, <операнд2>

Обмін вмістом поміж операндами, який можна виконувати над двома регістрами чи над регістром та змінною. Команда XCHG AX, BX є аналогічна до трьох операторів у С: temp = BX; BX = AX; AX = temp;

LEA <регістр>, <змінна>

Завантажити виконавчу адресу змінної до регістру.

IN <акумулятор>, <порт>

Завантажити до регістру акумулятора (AL, AX чи EAX) вміст порту введення/виведення, номер якого або задається числом від 0 до 255, або адресується регістром DX.

OUT <порт>, <акумулятор>

Завантажити до порту вміст акумулятора. На командах IN та OUT будується все спілкування процесора з пристроями введення/виведення – клавіатурою, вінчестером, різними контролерами тощо, і використовуються ці команди, першою чергою, у драйверах пристроїв.

1.2.6 Арифметичні команди

ADD < операнд1 >, < операнд2 >

Додати до операнда1 операнд2 і записати суму до операнда1, не змінюючи операнд2. Операнд2 може бути числом, регістром чи змінною, операнд1 – регістром чи змінною.

ADC < операнд1 >, < операнд2 >

Додати до операнда1 операнд2 і прапорець CF регістру прапорців процесора й записати суму до операнда1, не змінюючи операнд2. Пара команд ADD/ADC використовується для до-

давання чисел підвищеної точності. Приміром, для складання двох 64-бітових цілих чисел, одне з яких розташоване у парі регістрів EDX:EAX (молодше подвійне слово (біти 0 … 31) – в EAX і старше (біти 32 … 63) – в EDX), а решта – у парі регістрів EBX:ECX:

```
add eax,ecx
adc edx,ebx
```

Якщо при додаванні молодших подвійних слів відбулося перенесення зі старшого розряду (прапорець CF =1), то це буде враховано командою ADC.

SUB < операнд1 >, < операнд2 >

Відняти від операнда1 операнд2 і записати результати до операнда1, не змінюючи операнд2. Операнд2 може бути числом, регістром чи змінною, операнд1 – регістром чи змінною.

SBB < операнд1 >, < операнд2 >

Відняти від операнда1 операнд2 і прапорець CF регістру прапорців процесора й записати суму до операнда1, не змінюючи операнд2. Команду можна використовувати для віднімання 64-бітових чисел в EDX:EAX та EBX:ECX аналогічно до ADD/ADC:

```
sub eax,ecx
sbb edx,ebx
```

INC < операнд >

Збільшити операнд (регістр чи змінну) на 1 (інкремент).

DEC < операнд >

Зменшити операнд (регістр чи змінну) на 1 (декремент).

NEG < операнд >

Зміна знаку операнду на протилежний.

MUL < операнд >

Множення чисел без знаку. Помножити операнд (регістр чи змінну) на вміст акумулятора (AL, AX чи EAX, залежно від розміру операнда) й записати результат до AX, DX:AX, EDX:EAX відповідно.

Множення чисел зі знаком має три форми, які відрізнюються кількістю операндів:

IMUL < операнд >

Помножити операнд (регістр чи змінну) на вміст акумулятора (AL, AX чи EAX, залежно від розміру операнда) й записати результат до AX, DX:AX, EDX:EAX відповідно.

IMUL < операнд1 >, < операнд2 >

Помножити операнд1 (регістр) на операнд2 (регістр, змінна чи число) й записати результат до операнда1.

IMUL < операнд1 >, < операнд2 >, < операнд3>

Помножити операнд2 (регістр чи змінна) на операнд3 (число) й записати результат до операнда1 (регістр).

DIV < операнд >

Поділ цілих чисел без знаку. Поділити вміст акумулятора (AL, AX чи EAX, залежно від розміру операнда) на операнд (регістр чи змінну) й записати результат до AL, AX, EAX, а остачу – до AH, DX, EDX відповідно. Результат завжди округлюють у бік нуля, знак остачі збігається зі знаком діленого.

IDIV < операнд >

Поділ цілих чисел зі знаком. Поділити вміст акумулятора (AL, AX чи EAX, залежно від розміру операнда) на операнд (регістр чи змінну) і записати результат до AL, AX, EAX, а остачу – до AH, DX, EDX відповідно.

CMP < операнд1 >, < операнд2 >

Порівняти операнд1 з операндом2 і встановити прапорці регістру прапорців. Дія виконується шляхом віднімання операнда2 (регістру, змінної чи числа) з операнда1 (регістру чи змінної; операнди не можуть бути змінними водночас), причому результат віднімання нікуди не записується, лише встановлюються прапорці CF, OF, SF, ZF, AF та PF. Зазвичай команду CMP використовують разом з командами умовного переходу.

Внаслідок виконання більшості арифметичних операцій здійснюється встановлення в 0 чи 1 шести прапорців регістру прапорців процесора:

- прапорець перенесення CF=1, якщо результат попередньої операції не вмістився в операнді й відбулось перенесення одиниці зі старшого біта результату (при додаванні) чи позичання одиниці до старшого розряду (при відніманні);
- прапорець напівперенесення чи допоміжного перенесення AF=1, якщо відбулось перенесення одиниці з молодших 4-х розрядів результату (при додаванні) чи позичання одиниці до цих розрядів (при відніманні);
- прапорець нуля ZF=1, якщо результат попередньої операції дорівнює 0;
- прапорець знаку SF завжди дорівнює старшому бітові результату;
- прапорець парності PF=1, якщо результат є парний;
- прапорець переповнення OF=1, якщо результат попередньої арифметичної операції над числами зі знаком виходить за припустимі для них межі. Приміром, якщо при додаванні двох додатних чисел результатом є число зі старшим бітом, дорівнюваним одиниці, тобто від'ємним, і навпаки.

1.2.7 Команди зсуву

SHR < операнд >, < лічильник >

Логічний побітовий зсув операнда (регістру чи змінної) праворуч на значення лічильника (числа чи регістра CL) (враховуються лише 5 молодших бітів, які набирають значень від 0 до 31). Старший біт операнда обнулюється, а молодший – заноситься до CF. Операція є еквівалентна до б е з з н а к о в о г о поділу на 2 у степені лічильника. Приміром, число 0110B (6) після зсуву на 1 праворуч дає число 011B (3), а число 1000B (8) після зсуву на 2 праворуч – число 010B (2).

SAR < операнд >, < лічильник >

Арифметичний побітовий зсув операнда (регістру чи змінної) праворуч на значення лічильника. Операція є аналогічна до SHR, але знак операнда зберігається, оскільки старший біт операнда не обнулюється, а зберігає попереднє значення.

SHL < операнд >, < лічильник >

SAL < операнд >, < лічильник >

Команди виконують одну й ту саму дію: побітовий зсув операнда (регістру чи змінної) ліворуч на значення лічильника. Старший біт операнда заноситься до CF, а молодший – обнулюється. Операція є еквівалентна до множення на 2 у степені лічильника. Приміром, число 0110B (6) після зсуву на 1 праворуч дає число 1100B (12), а число 0100B (4) після зсуву на 2 праворуч – число 0001 0000B (16).

1.3 Покрокове виконування додатків та застосовування дизасемблера при налагоджуванні програмного коду

Застосовування точок зупину (breakpoint – переривання виконання) є потужним засобом при налагоджуванні програм, які дозволяють виконувати програму фрагментами від однієї точки переривання до іншої. Щоби поставити точку переривання, достатньо у вікні редактора коду клацнути мишею на смужці ліворуч коду потрібного рядка, після чого рядок набуде червоного кольору і ліворуч від нього з'явиться червона точка. При повторному клацанні мишею точка переривання зникне.

Вікно вбудованого дизасемблера відкривається за допомогою команд меню **View** (Вид) – **Debug Windows** (Вікна наладника) – **CPU** (Процесор) за запущеного на виконання проекту (рис. 1.1).

Задля покрокового проходження фрагмента програми можна використовувати команди меню Run, наведені у табл. 1.2.

Таблиця 1.2 – Команди меню Run

Команда меню Run	„Гаряча” клавіша	Пояснення
Step Over (по кроках без заходу до ...)	F8	Покрокове виконання рядків програми, вважаючи виклик функції чи процедури за один рядок, тобто входження до функцій та процедур не провадиться
Trace Into (трасування із заходом до ...)	F7	Покрокове виконання рядків програми із заходом до функцій та процедур, що викликаються
Trace to Next Source Line (Трасування до наступного рядка)	Shift + F7	Перехід до наступного рядка
Run to Cursor (виконати до курсора)	F4	Виконати фрагмент програми до оператора, на якому розташовано курсор у вікні редактора коду

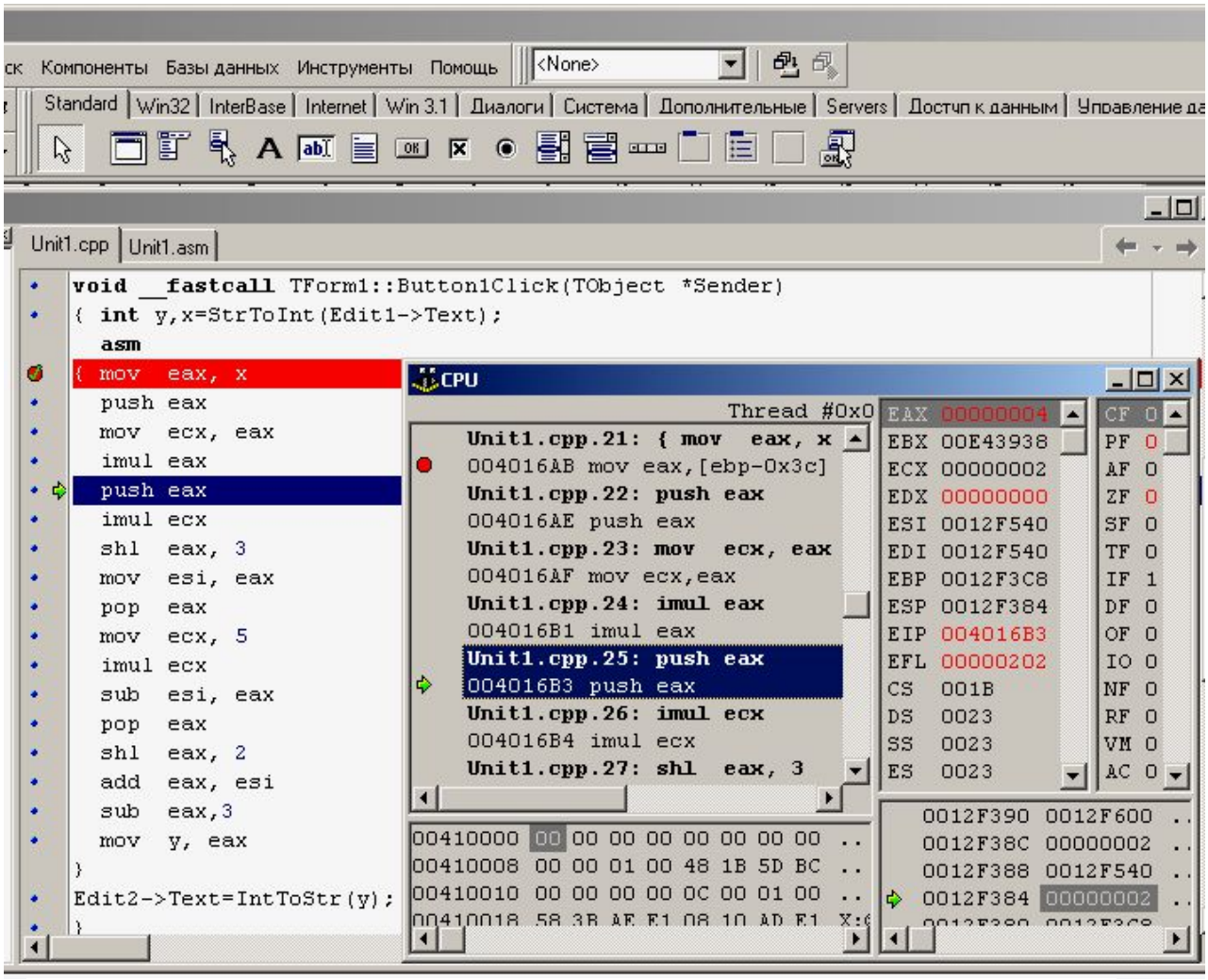


Рис. 1.1 – Вигляд вікна дизасемблера для прикладу 1.1

Задля покрокового виконування програми у дизасемблері доцільно послуговуватись такою послідовністю дій:

1) поставити точку переривання у першому виконуваному рядку;

2) запустити покрокове виконування програми за допомогою F8 або F7;

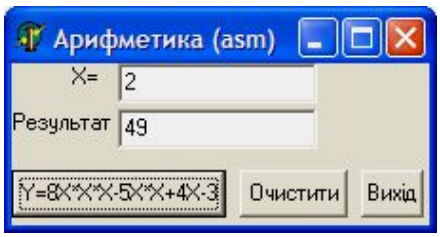
3) відкрити вікно дизасемблера **View – Debug Windows – CPU**;

4) здійснювати покрокове виконування програми за допомогою F8 або F7, переглядаючи асемблерні коди програми та вміст усіх регістрів мікропроцесора.

1.4 Приклади використовування вбудованого асемблера задля виконування арифметичних операцій

Приклад 1.1 Обчислити значення виразу $y = 8x^3 - 5x^2 + 4x - 3$ для різних значень x.

Вигляд вікна вбудованого дизасемблера за покрокового налагоджування даного прикладу наведено на рис. 1.1.



За результатами покрокового виконування команд цієї програми за $x = 2$ стан використовуваних регістрів наведено в табл. 1.3. Стан регістрів наводиться у шістнадцятковій системі, а в дужках подано їхнє десяткове значення для двознакових чисел.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int y, x = StrToInt(Edit1->Text);
  asm
    { mov  eax, x
      push eax
      mov  ecx, eax
      imul  eax
      push eax
      imul  ecx
      shl    eax, 3
      mov  esi, eax
      pop   eax
      mov  ecx, 5
      imul  ecx
      sub   esi, eax
      pop  eax
      shl    eax, 2
      add  eax, esi
      sub  eax,3
      mov  y, eax
    }
Edit2->Text = IntToStr(y);
}
```

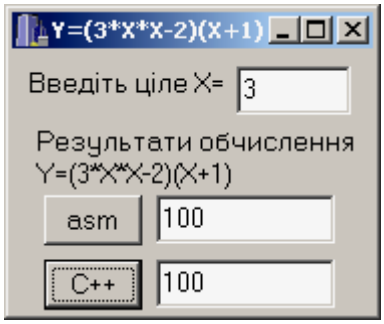
Таблиця 1.3 – Стан регістрів процесора за покрокового налагоджування

Команди	Стан використовуваних регістрів				Стек
	EAX	EDX	ECX	ISI	
mov eax, x	2				
push eax	2				2
mov ecx, eax	2		2		
imul eax	4	0	2		
push eax	4	0	2		4, 2
imul ecx	8	0			4, 2
shl eax, 3	40 (=64)	0	2		4, 2
mov esi, eax	40	0	2	40	4, 2
pop eax	4	0	2	40	2
mov ecx, 5	4	0	5	40	2
imul ecx	14 (=20)	0	5	40	2
sub esi, eax	14	0	5	2C (=44)	2
pop eax	2	0	5	2C	
shl eax, 2	8	0	5	2C	
add eax, esi	34 (=52)	0	5	2C	
sub eax,3	31 (=49)	0	5	2C	
mov y, eax	31	0	5	2C	

Приклад 1.2 Обчислити значення виразу $y = (3x^2 - 2)(x + 1)$ для різних значень x.

Для зручності перевірки правильності обчислень в асемблері створимо додаткову командну кнопку й розробимо програмний код для обчислення завдання на C++.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{  int y, x = StrToInt(Edit1->Text);
   asm
   { mov  eax, x
     imul  eax
     imul  eax,3
     sub   eax, 2
     mov  ecx, x
     inc    ecx
     imul  ecx
     mov  y, eax
    } Edit2->Text = IntToStr(y);
}
```




```
void __fastcall TForm1::Button2Click(TObject *Sender)
{ int y, x = StrToInt(Edit1->Text);
  y = (3 * x * x – 2) * ( x + 1);
  Edit3->Text = IntToStr(y); }
```

2 Контрольні запитання

1 Наведіть два способи збільшування вмісту регістру EAX учетверо й опишіть переваги того чи іншого способу.

2 Переведіть числа 39 та 279 у шістнадцяткову та двійкову системи обчислення.

3 Наведіть команду асемблера для оголошення змінної X розміром 2 байти з початковим значенням 1256.

4 Наведіть три способи обміну вмістом двох регістрів – EAX та в ECX – й опишіть переваги того чи іншого способу.

3 Лабораторне завдання

1 Розробити програмний проект C++ Builder для введення початкових даних і розв'язання засобами вбудованого асемблера індивідуального завдання відповідно до варіанта з табл. 1.4. Як елементи керування використовувати командні кнопки: для запускання процедур обчислення; для очищування текстових полів форми; для завершення роботи з програмою.

2 Створити таблицю на зразок табл. 1.3 та заповнити значеннями вмісту використовуваних у проекті регістрів і стека для кожної команди програми. Для заповнення і перевірки значень таблиці у процесі налагодження програми використовувати покрокове виконування команд й відстежувати стан стека та регістрів мікропроцесора у вікні дизасемблера.

3 Задля зручності перевірки правильності обчислень в асемблері створити додаткову командну кнопку й написати програмний код для обчислення індивідуального завдання на C++ на зразок прикладу 1.2.

4 Оформити протокол лабораторної роботи, в якому, окрім відповідей на всі контрольні запитання і тексту програми, записати таблицю і заповнити її. Занести остаточні результати обчислень до протоколу.

Таблиця 1.4 – Варіанти індивідуальних завдань

№ вар.	Індивідуальне завдання	№ вар.	Індивідуальне завдання
1	$Y = 4X(X^2 + 3) - 7X - 1$	16	$Y = X(X^2 - 1) + 11X$
2	$Y = 2(X^4 - X^2) + 1$	17	$Y = (X^2 - 2X)(4X + 1)$
3	$Y = 8X(1 - X^2) - 7$	18	$Y = 2X(1 - X^2) + 11$
4	$Y = 4(X^4 - 3X^2) + 2X + 1$	19	$Y = 8(X^2 - 3)(2X + 1)$
5	$Y = (X^2 - 2)(2X - 1) + 5$	20	$Y = (9 - 2X^2)(2 - X) + 1$
6	$Y = (3 - X)^2(2X - 1)$	21	$Y = (11 + X)^2 - 2X + 1$
7	$Y = (8X + 1)(X^2 - 6) - 1$	22	$Y = (X + 1)(4X^2 - 9)$
8	$Y = 2X^3 + 3X^2 - 15X + 7$	23	$Y = 4X^3 - 6X^2 + X + 1$
9	$Y = X^3 - 11X^2 + 8X + 1$	24	$Y = X^3 - 3X^2 + 8(X + 1)$
10	$Y = X^3 - X^2 + 3X - 1$	25	$Y = (X^3 + 2)(8X^2 - 3) - 1$
11	$Y = 4X^3 - 8X^2 + X + 1$	26	$Y = 2X(5X^2 - 9) + 3$
12	$Y = (X^2 - 1)(2 + X) + 3$	27	$Y = 3X^2(4X - 5) + 15$
13	$Y = (8 - X)^2(4X - 1)$	28	$Y = (11X - 5)(X^2 + 1) - 1$
14	$Y = (8X + 1)(X^2 - 6) - 1$	29	$Y = (X^3 - 7)(9 - 2X^2) + 1$
15	$Y = (X^4 + 1)(3 - 4X^2) + 1$	30	$Y = 3X^3 - 8X^2 - X + 1$

Лабораторна робота № 2

Арифметичні операції над дійсними числами у вбудованому в C++ Builder асемблері

Мета роботи: Вивчення асемблерних арифметичних команд опрацьовування дійсних чисел та набуття навичок роботи з регістрами співпроцесора при створюванні додатків C++ Builder із вбудованим асемблером.

1 Основні теоретичні відомості

1.1 Робота в асемблері з дійсними числами

У процесорах Intel усі операції з дійсними числами (числами з плаваючою комою) виконує співпроцесор (FPU – Floating Point Unit) – пристрій з власними регістрами та набором команд. Окрім роботи з дійсними числами, співпроцесор підтримує роботу з цілими числами. FPU виконує усі обчислення у 80-бітовому розширеному форматі, а 32- й 64-бітові числа використовуються для обміну даними з основним процесором та пам'яттю.

1.2 Регістри FPU

FPU надає вісім 80-бітових регістрів для зберігання даних і п'ять допоміжних регістрів. Ці регістри не адресуються за іменами, як регістри головного процесора, а розглядаються як стек, вершина якого називається ST(0); більш глибокі елементи – ST(1), ST(2) тощо до ST(7). До регістрів не можна звертатися безпосередньо за іменами.

1.3 Деякі команди FPU

1.3.1 Команди пересилання даних

FLD <змінна> // Завантажити дійсну змінну до стека
FILD <змінна> // Завантажити цілу змінну до стека
FST <змінна> // Скопіювати ST(0) до дійсної змінної
FSTP <змінна> // Завантажити (виштовхнути) ST(0) до дійсної змінної
FIST <змінна> // Скопіювати ST(0) до цілої змінної
FISTP <змінна> // Завантажити (виштовхнути) ST(0) до цілої змінної
FXCH <регістр> // змінити місцями два регістри (ST(0) і регістр)
// Для FXCH (без операндів) – ST(0) та ST(1)

1.3.2 Константи FPU

FLD1 // Завантажити до стека 1.0
FLDZ // Завантажити до стека +0.0
FLDPI // Завантажити до стека число π

1.3.3 Арифметичні команди

FADD <операнд1>,<операнд2> // Додавання значень операнда1 та операнда2
// і розміщення результату в операнді1.

Команда може мати такі форми:

1) FADD <змінна> підсумовує значення ST(0) та змінної й розміщує результат у ST(0);
2) FADD ST(0),ST(n), FADD ST(n),ST(0), коли обидва операнди є регістрами;
3) FADD без операндів є аналогічна до FADD ST(0),ST(1),

FIADD <ціла змінна> // Додавання до ST(0) цілої змінної, результат у ST(0)
FSUB <операнд1>,<операнд2> // Віднімання з операнда1 значення операнда2
// і розміщення результату в операнді1.

Команда може мати такі форми:

1) FSUB <змінна> віднімає із ST(0) значення змінної й розміщує результат у ST(0)
2) FSUB ST(0),ST(n), FSUB ST(n),ST(0), коли обидва операнди є регістрами
3) FSUB без операндів є аналогічна до FSUB ST(0),ST(1)

FISUB <ціла змінна> // Віднімання від ST(0) значення цілої змінної, результат у ST(0)
FSUBR <операнд1>,<операнд2> // Обернене віднімання з операнда2 значення операнда1
FISUBR <ціла змінна> // Обернене віднімання цілих чисел
FMUL <операнд1>,<операнд2> // Множення дійсних чисел операнда1 та операнда2
// Результат записується до операнда1

Команда може мати такі форми:

1) FMUL <змінна> множення ST(0) на значення змінної. Результат у ST(0)
2) FMUL ST(0),ST(n), FMUL ST(n),ST(0), коли обидва операнди є регістрами
3) FMUL без операндів є аналогічна до FMUL ST(0),ST(1)

FIMUL <ціла змінна> // Множення цілого числа змінної та ST(0). Результат у ST(0)
FDIV <операнд1>,<операнд2> // Поділити операнда2 на операнд1. Результат у операнд2.

Команда може мати такі форми:

1) FDIV <змінна> поділити ST(0) на значення змінної. Результат у ST(0)
2) FDIV ST(0),ST(n), FDIV ST(n),ST(0), коли обидва операнди є регістрами
3) FDIV без операндів є аналогічна до FDIV ST(0),ST(1)

FIDIV <ціла змінна> // Поділити ST(0) на цілу змінну. Результат у ST(0)
FDIVR <операнд1>,<операнд2> // Поділити операнда1 на операнд2. Результат у операнд1
FIDIVR <ціла змінна> // Обернене ділення ST(0) на цілу змінну. Результат у ST(0)
FABS // Абсолютне значення ST(0)
FCHS // Змінити знак значення ST(0) на протилежний
FSCALE // Помножити ST(0) на 2 у степені ST(1)
FSQRT // Квадратний корінь з ST(0)
FSIN // Синус числа ST(0), заданого в радіанах
FCOS // Косинус числа ST(0), заданого в радіанах
FSINCOS // Синус та косинус числа ST(0), заданого в радіанах. Значення косинуса –
// в ST(0), синуса – в ST(1), тобто наступною командою FDIV можна здобути
// **котангенс** в ST(0)
FPTAN // Тангенс числа ST(0), заданого в радіанах. Результат – в ST(1), в ST(0) – 1,
// тобто наступною командою FDIVR можна здобути **котангенс** в ST(0)
FPATAN // Арктангенс числа, здобутого при діленні ST(1) на ST(0), тобто обчислюєть-
// ся кут поміж віссю абсцис і лінією, проведеною з центра координат
// до точки ST(1), ST(0). Результат – в ST(0).

1.4 Приклади використовування команд FPU у вбудованому асемблері

Приклад 2.1 Створювання програмного проекту для обчислювання різних математичних функцій засобами вбудованого асемблера.

```
float x, y, z;
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  asm
  { fld x
    fchs          // – x
    fstp x  }
Edit3->Text = FloatToStr(x);
}
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{ y = StrToFloat(Edit2->Text);
  asm
  { fld y
    fchs          // –y
    fstp y  }
Edit3->Text = FloatToStr(y);
}
```

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  y = StrToFloat(Edit2->Text);
asm
{ fld x
  fadd y        // x + y
  fstp z  }
Edit3->Text = FloatToStr(z);
}
```

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  y = StrToFloat(Edit2->Text);
asm
{ fld x
  fsub y        // x – y
  fstp z  }
Edit3->Text = FloatToStr(z);
}
```

```
void __fastcall TForm1::Button5Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  y = StrToFloat(Edit2->Text);
   asm
    { fld x
      fmul y     //  x * y
      fstp z  }
  Edit3->Text = FloatToStr(z);   }
```

```
void __fastcall TForm1::Button6Click(TObject *Sender)
 { x = StrToFloat(Edit1->Text);
   y = StrToFloat(Edit2->Text);
    asm
     { fld x
       fdiv y     //  x / y
       fstp z  }
 Edit3->Text = FloatToStr(z);   }
```

```
void __fastcall TForm1::Button7Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
   asm
   {  fld x
      fsqrt          // sqrt(x)
      fstp z   }
Edit3->Text = FloatToStr(z);  }
```

```
void __fastcall TForm1::Button8Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  asm
   { fld x
     fsin     //  sin(x)
     fstp z  }
Edit3->Text = FloatToStr(z);  }
```

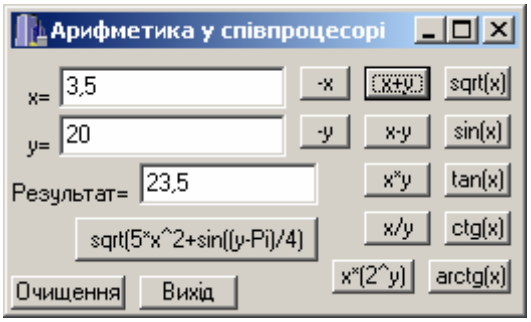
```
void __fastcall TForm1::Button9Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  asm        // tan(x)
   { fld x     // или
     fptan   // fsincos
     fstp z  // fdivr
     fstp z
 }
Edit3->Text = FloatToStr(z);  }
```

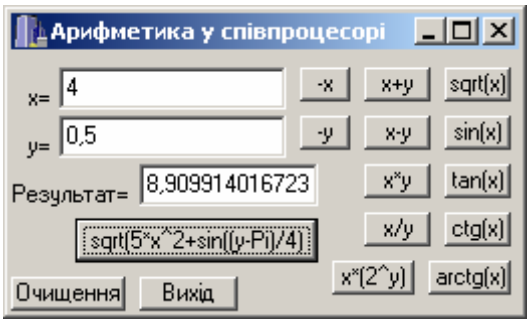
```
void __fastcall TForm1::Button10Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  asm
   { fld x
     fptan
     fxch          // ctg(x)
     fdiv
     fstp z }
Edit3->Text=FloatToStr(z);      }
```

```
void __fastcall TForm1::Button11Click(TObject *Sender)
{ x = StrToFloat(Edit1->Text);
  asm
   { fld x
     fld1
     fpatan   // arctan(st(1)/st(0))
     fstp z  }
 Edit3->Text = FloatToStr(z);
}
```

```
void __fastcall TForm1::Button12Click(TObject *Sender)
{  x = StrToFloat(Edit1->Text);
   y = StrToFloat(Edit2->Text);
 asm
 { fld y
   fld x
   fscale   // x * (2 ^ y)
   fstp z  }
Edit3->Text = FloatToStr(z); }
```

```
void __fastcall TForm1::Button13Click(TObject *Sender) // sqrt(5*x^2+sin((y-Pi)/4)
{ int s;    x = StrToFloat(Edit1->Text);   y = StrToFloat(Edit2->Text);
  asm
 { fld x
   fmul x          //  x * x
   mov s, 5
   fild s
   fmul           // 5* x ^ 2
   fld y
   fldpi
   fsub           // y - pi
   mov s, 4
   fild s
   fdiv            // (y – pi)/4
   fsin
   fadd            // 5*x^2+sin((y-Pi)/4
   fabs
   fsqrt
   fstp z          //z=sqrt(5*x^2+sin((y-Pi)/4)
 } Edit3->Text = FloatToStr(z);  }
```





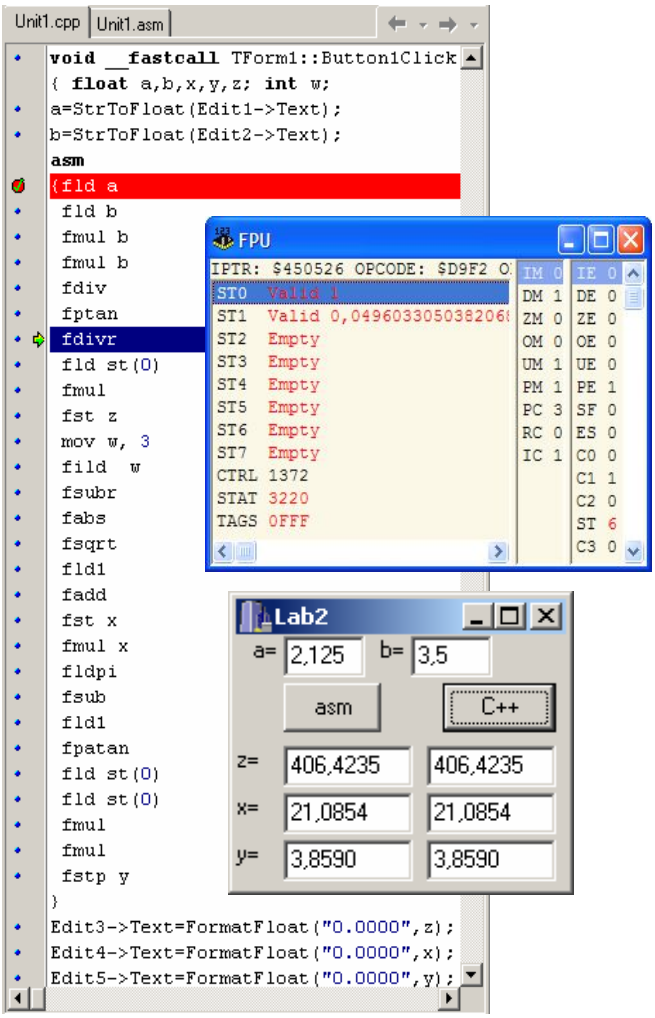

```
void __fastcall TForm1::Button14Click(TObject *Sender)
{  Edit1->Clear();  Edit2->Clear();  Edit3->Clear(); }
```

```
void __fastcall TForm1::Button15Click(TObject *Sender)
{  Close(); }
```

Приклад 2.2 Створити програмний проект для обчислення засобами вбудованого асемблера $y = \text{arctg}^3\left(x^2 - \pi\right);\ x = 1 + \sqrt{|3 - z|};\ z = \text{ctg}^2\, a/b^3$ для різних значень змінних дійсного типу a та b, які вводять з екрана.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float a, b, x, y, z;  int w;
a=StrToFloat(Edit1->Text); b = StrToFloat(Edit2->Text);
asm
```

{	ST(0)	ST(1)	ST(2)
fld a	*a*		
fld b	*b*	*a*	
fmul b	*b*^2	*a*	
fmul b	*b*^3	*a*	
fdiv	*a*/ *b*^3		
fptan	1	tg(*a*/ *b*^3)	
fdivr	ctg(*a*/ *b*^3)		
fld st(0)	ctg(*a*/ *b*^3)	ctg(*a*/ *b*^3)	
fmul	ctg^2(*a*/ *b*^3)		
fst z	*z*=ctg^2(*a*/ *b*^3)		
mov w, 3	*z*		
fild w	3	*z*	
fsubr	*z*-3		
fabs	|*z*-3|		
fsqrt	sqrt|*z*-3|		
fld1	1	sqrt|*z*-3|	
fadd	1+sqrt|*z*-3|		
fst x	x=1+sqrt|*z*-3|		
mul x	x*x		
fldpi	Pi	x*x	
fsub	x*x-Pi		
fld1	1	x*x-Pi	
fpatan	arctan((x*x-pi)/1)		
fld st(0)	arctan((x*x-pi)/1)	arctan((x*x-pi)/1)	
fld st(0)	arctan((x*x-pi)/1)	arctan((x*x-pi)/1)	arctan(..)
fmul	arctan(...)^2	arctan((x*x-pi)/1	
fmul	arctan(...)^3		
fstp y			




```
} Edit3->Text=FormatFloat("0.0000", z);
   Edit4->Text=FormatFloat("0.0000", x); Edit5->Text=FormatFloat("0.0000", y);  }
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{ float a, b, x, y, z;   a=StrToFloat(Edit1->Text); b = StrToFloat(Edit2->Text);
  z=pow(1/tan(a/(b*b*b)),2);        x=1+sqrt(fabs(3-z));              y=pow(atan(x*x-M_PI),3);
  Edit6->Text=FormatFloat("0.0000", z);
  Edit7->Text=FormatFloat("0.0000", x); Edit8->Text=FormatFloat("0.0000", y);  }
```

1.5 Переглядання стану регістрів співпроцесора за покрокового виконування додатків

Задля покрокового виконування програми й переглядання регістрів співпроцесора доцільно послуговуватись такою послідовністю дій:

1) поставити точку переривання у першому рядку асемблерної вставки;

2) запустити проект на виконання і ввести початкові дані у відповідні вікна форми, після чого натиснути відповідну командну кнопку, яка запустить програму до першої точки переривання;

3) відкрити вікно стану регістрів співпроцесора **View** – **Debug Windows** – **FPU** та процесора **View** – **Debug Windows** – **CPU**;

4) продовжити покрокове виконування програми за допомогою F8 або F7, переглядаючи асемблерні коди програми та вміст усіх регістрів мікропроцесора й співпроцесора.

2 Контрольні запитання

1 В чому полягає призначення FPU?
2 Які кількість та місткість регістрів має співпроцесор для зберігання даних?
3 Наведіть три способи збільшення вмісту регістру ST(0) учетверо.
4 Наведіть два способи обчислення котангенса числа у регістрі ST(0).

3 Лабораторне завдання

1 Розробити програмний проект C++ Builder для введення необхідних вихідних даних і розв'язання засобами вбудованого асемблера індивідуального завдання відповідно до варіанта з табл. 2.1. Як елементи керування використовувати командні кнопки: для запускання процедур обчислень, для очищення текстових полів форми і для завершення роботи з програмою. Вивести результати всіх трьох обчислюваних змінних до відповідних текстових полів.

2 Створити таблицю на зразок таблиці прикладу 2.2 та заповнити значеннями вмісту використовуваних у проекті регістрів співпроцесора для кожної команди програми. Для заповнення і перевірки значень таблиці використовувати покрокове виконування команд й відстежувати стан регістрів співпроцесора у вікні дизасемблера.

3 Для зручності перевірки правильності обчислень в асемблері створити додаткову командну кнопку й написати програмний код для обчислення індивідуального завдання на C++ на зразок поданого у прикладі 2.2.

4 Оформити протокол лабораторної роботи, в якому, окрім тексту програми, записати таблицю і заповнити її. Занести остаточні результати обчислень до протоколу.

Таблиця 2.1 – Варіанти індивідуальних завдань

№ вар.	Індивідуальне завдання	№ вар.	Індивідуальне завдання
1	$y = a\sin^2 b + b/\cos a;\ a = \sqrt{\lvert b-c\rvert};\ b = \operatorname{arctg}(x)$	16	$y = \cos\lvert x + t^2\rvert;\ x = t^2 - 4p;\ t = \operatorname{arctg}(\sqrt{3}/p)$
2	$y = \sqrt{a^2 + b^2};\ a = \operatorname{tg}\lvert x\rvert;\ b = \sin(2-a)$	17	$y = \operatorname{arctg}(1-a);\ a = \operatorname{tg}\lvert 4t + b^2\rvert;\ t = b^2 - \sqrt{3b}$
3	$y = \operatorname{ctg} a/c;\ c = a^2 + \sqrt{b};\ a = b^3 - \cos\lvert b\rvert$	18	$y = \sqrt{x^2 + 8c};\ x = \lvert c - a\rvert a;\ c = \cos^2 a + 1$
4	$y = \sqrt{\lvert a-b\rvert};\ a = \operatorname{tg}(x);\ b = x + \sin(1/x)$	19	$y = \sin p + v^3;\ p = \operatorname{tg}\lvert x\rvert;\ v = \sqrt{x+1}/(x^2-5)$
5	$y = a^3/b^2;\ a = \sqrt{\lvert x\rvert + 1};\ b = \sin x^2 - 3$	20	$y = 4x^3/t^2;\ x = \operatorname{tg} 2^a;\ t = \sin\sqrt{6 - a^3}$
6	$y = \cos^2(p+\pi);\ p = x^2 - \sqrt{\lvert x\rvert};\ x = \operatorname{arctg}(1/b)$	21	$y = c^2 + \sqrt{\lvert a\rvert};\ c = \operatorname{ctg}(a/x);\ a = 1 - 8x$
7	$y = c^3/\cos c;\ c = a^2 + b^2;\ a = \sqrt{\lvert x-1\rvert}$	22	$y = \operatorname{arctg}^2\lvert x\rvert;\ x = \sqrt{t+8};\ t = \cos(1-b)$
8	$y = \sin^3(a-b);\ a = t^3 + \sqrt{b};\ b = \operatorname{tg}^2\lvert x\rvert$	23	$y = v + \operatorname{tg}(w-\pi);\ v = \cos^2 a;\ w = \sqrt{1 + \lvert 5a\rvert}$
9	$y = \operatorname{arctg} x^2;\ x = p - k;\ k = \sqrt{p + t^2}$	24	$y = x^2 + \sqrt{\lvert 2x-1\rvert};\ x = \cos^2 b;\ b = \sqrt{8 + t^2}$
10	$y = \cos^2(a - \sin b);\ a = \sqrt{\lvert x\rvert};\ b = x^4 + m^2$	25	$y = 1 - \cos x^2;\ x = \operatorname{ctg} c/a;\ c = \sqrt{\lvert a-3\rvert}$
11	$y = \sin^3 a + \cos^2 x;\ a = c - k^2;\ c = \operatorname{arctg}\lvert x\rvert$	26	$y = \operatorname{ctg}^2\lvert x - \pi\rvert;\ x = \sqrt{a+b};\ a = 2^{1+b}$
12	$y = \cos\sqrt{x-1};\ x = a + b^2;\ a = \sin^2(b-\pi)$	27	$y = \operatorname{arctg}^3\lvert p-1\rvert;\ p = \sqrt{x^2 + 4a};\ x = \sin(\pi/a)$
13	$y = a\cos x - b\sin x;\ x = \sqrt{\lvert a-b\rvert};\ a = t^2 b$	28	$y = \sin^2(p+\pi);\ p = 2^{\sqrt{t}};\ t = x^2 + \sqrt{\lvert x-1\rvert}$
14	$y = \sqrt{\lvert x\rvert}\sin a + 5;\ a = \operatorname{tg}^2\lvert 2x\rvert;\ x = b/3$	29	$y = \cos^3 x + \lvert a\rvert;\ x = \operatorname{tg}(1/b);\ b = \sqrt{7 + a^2}$
15	$y = 2a/\operatorname{arctg}(b-1);\ a = \sqrt{x^2 + b^2};\ x = \operatorname{tg}(b+\pi)$	30	$y = \sin(a^2 - 1);\ a = \sqrt{\lvert b + t\rvert};\ t = \operatorname{ctg}(b^2 + 1)$

Лабораторна робота № 3

Опрацьовування числових масивів засобами вбудованого в C++ Builder асемблера

Мета роботи: Вивчення асемблерних умовних та циклічних команд та набуття навичок опрацьовування масивів цілих чисел засобами вбудованого в C++ Builder асемблера.

1 Основні теоретичні відомості

1.1 Логічні команди асемблера

AND <операнд1>, <операнд2>

Логічне побітове множення (логічне „І") операнда1 (регістру чи змінної) та операнда2 (регістру, змінної, числа; операнди не можуть бути змінними одночасно). Результат записується в операнд1. Принцип виконання операції AND для різних значень бітів двох операндів наведено у таблиці праворуч. Основним використанням цієї операції є обнулення окремих бітів. Наприклад, команда

X	Y	X and Y
0	0	0
0	1	0
1	0	0
1	1	1

```
AND AL, 00001111b
```

старші чотири біти регістру AL перетворить на нулі, а молодші залишить без зміни. Аналогічна команда (&) є в мові C++.

OR <операнд1>, <операнд2>

Логічне побітове додавання (логічне „ЧИ") операнда1 (регістру чи змінної) й операнда2 (регістру, змінної, числа; операнди не можуть бути змінними одночасно). Результат записується в операнд1. Принцип виконання операції OR для різних значень бітів двох операндів наведено у таблиці праворуч. Основним використанням цієї операції є встановлення в 1 окремих бітів. Наприклад, команда

X	Y	X or Y
0	0	0
0	1	1
1	0	1
1	1	1

```
OR  AL, 00001111b
```

молодші чотири біти регістру AL встановить у 1, а старші залишить без зміни. Аналог команди в C++ – (|).

XOR <операнд1>, <операнд2>

Логічне побітове виняткове „ЧИ" операнда1 (регістру чи змінної) й операнда2 (регістру, змінної, числа; операнди не можуть бути змінними одночасно). Результат записується до операнда1. Принцип виконання операції логічне виняткове „ЧИ" для різних значень бітів двох операндів наведено у таблиці праворуч. Тобто біт результату дорівнює 1, якщо відповідні біти операндів розрізнюються, і 0 – в інших випадках. Аналогічна команда в C++ записується ^. XOR використовується для різних цілей, наприклад:

X	Y	X xor Y
0	0	0
0	1	1
1	0	1
1	1	0

```
    1)   XOR  AX, AX        // обнулення регістру AX
або 2)   XOR  AX, BX
         XOR  BX, AX
         XOR  AX, BX        // змінює місцями вміст  AX та BX
```

Обидва приклади виконуються швидше, аніж відповідні очевидні MOV AX,0 та XCHG AX,BX.

NOT <операнд>

Інверсія кожного з бітів операнда (регістру чи змінної) на протилежний: 0 встановлюється в 1, а 1 – в 0. Аналогічна команда в C++ записується ~.

TEST <операнд1>, <операнд2>

Логічне порівняння двох операндів у відповідності з результатами виконання операції логічне AND і встановлення прапорців SF, ZF та PF без встановлення самого результату. TEST, так само як і CMP, використовується переважно разом з командами умовного переходу.

1.2 Деякі асемблерні команди передавання керування

JMP <позначка>

Безумовний перехід на операнд, перед яким встановлено позначку.

Розрізнюють типи переходу:

- **short** (короткий перехід) – якщо адреса переходу перебуває в межах –128...+127 байт від команди JMP;
- **near** (ближній перехід) – якщо адреса переходу перебуває у тому самому сегменті пам'яті, що й команда JMP;
- **far** (дальній перехід) – якщо адреса переходу перебуває у другому сегменті пам'яті.

JCC < позначка>

Набір команд умовного переходу (типу short чи near) за виконання певної умови. Перевірка умови здійснюється за рахунок перевірки відповідних прапорців регістру прапорців, які встановлюються попередньою командою, зазвичай CMP чи TEST. Основні варіанти команди JCC наведено у табл. 3.1.

Таблиця 3.1 – Команди умовного переходу

Код	Перевірка прапорців	Умова	Код	Перевірка прапорців	Умова
JA JG	ZF=0 і SF=OF	> (для чисел без знаку) > (для чисел зі знаком)	JAE JGE	CF=0 SF=OF	≥ (для чисел без знаку) ≥ (для чисел зі знаком)
JB JL	CF=1	< (для чисел без знаку) < (для чисел зі знаком)	JBE JLE	CF=1 чи ZF=1 ZF=1чи SF≠OF	≤ (для чисел без знаку) ≤ (для чисел зі знаком)
JE JNE	ZF=1 ZF=0	= ≠	JS JNS	SF=1 SF=0	якщо є знак якщо немає знаку
JC JNC	CF=1 CF=0	якщо було перенесення не було перенесення	JP JNP	PF=1 PF=0	якщо результат парний якщо результат непарний
JO	OF=1	якщо є переповнення	JNO	OF=0	якщо немає переповнення

Приклади: 1)
```
CMP EAX,0
JNE M1
...
M1: INC EAX
```
2)
```
M2: NEG EAX
    JS  M2
```

У прикладі 1 виконується перехід на позначку M1, якщо значення регістру EAX ≠ 0. Приклад 2 дозволяє здобути абсолютне значення числа в EAX, застосовуючи лише дві команди – змінення знаку і перехід на попередню команду повторно, якщо знак є від'ємний.

LOOP <позначка>

Команда циклу, яка виконує зменшення регістру CX на 1 і перехід типу SHORT на позначку, якщо CX ≠ 0. Тобто до регістру CX перед циклом слід занести кількість разів виконування циклу. Наприклад, у поданому нижче фрагменті команда ADD виконується 10 разів і підсумовуються усі числа від 10 до 0:

```
        XOR AX, AX
        MOV CX, 0AH
Loop_Star: ADD  AX, CX
        LOOP Loop_Star
```

Команда LOOP є еквівалентна до пари команд DEC CX та JNZ <позначка>, але її код є коротше на 1 байт і стан регістрів не змінюється.

1.3 Приклади опрацьовування одновимірних масивів в асемблері

У даному програмному проекті наведено шість найбільш поширених алгоритмів опрацьовування елементів одновимірного масиву з 10 цілих чисел.

З метою оптимізування тлумачення алгоритмів розв'язування та підтвердження правильності здобутих результатів розв'язування кожного із завдань наведено засобами двох мов – C++ та асемблера.


```
// Сума всіх елементів масиву
void __fastcall TForm1::Button3Click(TObject *Sender)
{ int i, s, a[10];
  for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
  asm
  {  pusha
     mov edx, 0
     mov eax, 0
     mov ecx, 10 //кількість елементів масиву
@n1: add eax, dword ptr a+edx
          add edx, 4
          loop @n1
     mov s, eax
     popa
  }  Edit1->Text= IntToStr(s);
}
```

```
//Факторіал першого елемента масиву
void __fastcall TForm1::Button4Click(TObject *Sender)
{ int i, f, a[10];
  for ( i = 0; i < 10; i++)
     a[i] = StrToInt(Memo1->Lines->Strings[i]);
asm
{  pusha
   mov eax, 1
   mov ecx, dword ptr a //для десятого елемента (a+4*9)
@next:
   mul ecx
   loop @next
   mov f, eax
   popa
  }
Edit3->Text = IntToStr(f);
}
```

```
// Сума всіх елементів масиву
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int i, s = 0, a[10];
 for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 for( i = 0; i < 10; i++) s += a[i];
 Edit2->Text = IntToStr(s);
}
```

```
//Факторіал першого елемента масиву
void __fastcall TForm1::Button2Click(TObject *Sender)
{ int i, f = 1, a[10];
 for( i = 0; i < 10; i++)
     a[i] = StrToInt(Memo1->Lines->Strings[i]);
 for( i = 1; i <= a[0]; i++)   f *= i;
 Edit4->Text = IntToStr(f);
}
```

```
// Добуток додатних елементів масиву
void __fastcall TForm1::Button7Click(TObject *Sender)
{ int i, p, k, a[10];
 for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
k = sizeof(a) / 4; //кількість елементів масиву
asm
 { pusha
   mov edi, 0
   mov eax, 1
   mov ecx, k // кількість повторень циклу
 @n2:
   mov esi, dword ptr a+edi
   cmp esi, 0
   jle @net
   mul esi
 @net:
   add edi, 4
   loop @n2
   mov p, eax
   popa
  }
 Edit5->Text = IntToStr(p);
}
```

```
// Сума парних елементів масиву
void __fastcall TForm1::Button8Click(TObject *Sender)
{ int k, i, s, a[10];
 for( i = 0; i < 10; i++)
       a[i] = StrToInt(Memo1->Lines->Strings[i]);
 k = sizeof(a)/4; // кількість елементів  масиву
 asm
{  pusha
   mov edx, 0
   mov eax, 0
   mov ecx, k   // кількість повторень циклу
@m2:
   mov esi, dword ptr a+edx
   and esi,1
   jnp @m1
   add eax, dword ptr a+edx
@m1:
   add edx, 4
   loop @m2
   mov s, eax
   popa
  }
 Edit7->Text = IntToStr(s);
}
```

```
// Добуток додатних елементів масиву
void __fastcall TForm1::Button5Click(TObject *Sender)
{ int i, p = 1, a[10];
 for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 for( i = 0; i < 10; i++)
     if (a[i] > 0) p *= a[i];
 Edit6->Text = IntToStr(p);
}
```

```
// Сума парних елементів масиву
void __fastcall TForm1::Button6Click(TObject *Sender)
{ int i, s = 0, a[10];
 for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 for( i = 0; i < 10; i++)
      if (a[i] % 2 == 0) s += a[i];
Edit8->Text = IntToStr(s);
}
```

```
//  Кількість елементів з діапазону [-2,3]
void __fastcall TForm1::Button11Click(TObject
*Sender)
{ int i, kol, a[10];
  for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
  asm
 { pusha
   mov edi, 0
   mov eax, 0
   mov ecx, 10
@nex:    mov esi, dword ptr a+edi
            cmp esi, -2
            jl @not
            cmp esi, 3
            ja @not
            add eax, 1
@not:    add edi, 4
           loop @nex
   mov kol, eax
   popa
  }
Edit9->Text = IntToStr(kol);
}
```

```
//  Мінімальний елемент
void __fastcall TForm1::Button12Click(TObject *Sender)
{ int i, min, a[10];
  for( i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 asm
 { pusha
   mov esi, dword ptr a
   mov min, esi         // min = a[0];
   mov edi, 4
   mov ecx, 9 // кількість елементів - 1
 @nn:
   mov esi, dword ptr a+edi
   cmp esi, min
   jg @nt
   mov min, esi
 @nt:
   add edi, 4
   loop @nn
   popa
 }
Edit11->Text = IntToStr(min);
}
```

```
//  Кількість елементів з діапазону [-2,3]
void __fastcall TForm1::Button9Click(TObject *Sender)
{ int i, k = 0, a[10];
 for(i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 for(i = 0; i < 10; i++)
      if (a[i] >= -2  &&  a[i] <= 3) k++;
Edit10->Text = IntToStr(k);
}
```

```
//  Мінімальний елемент
void __fastcall TForm1::Button10Click(TObject *Sender)
{ int i, min, a[10];
 for(i = 0; i < 10; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
 min = a[0];
 for(i = 1; i < 10; i++)
       if(min > a[i]) min = a[i];
 Edit12->Text = IntToStr(min);
 }
```

2 Контрольні запитання

1 Наведіть приклади трьох різних операторів, які надають регістрові eax значення 0.

2 Наведіть оператори для перевірки змінної x на парність.

3 Наведіть оператори для перевірки значення регістру ecx: 1) > 0; 2) ≠ 0; 3) < 0.

3 Лабораторне завдання

1 Розробити програмний проект C++ Builder для введення елементів масиву та розв'язання засобами вбудованого асемблера індивідуального завдання відповідно до варіанта з табл. 3.2.

2 Підчас налагодження програми слід використовувати покрокове виконування команд програми і відстежувати стан регістрів мікропроцесора у вікні дизасемблера.

3 Для зручності перевірки правильності обчислень в асемблері створити додаткову командну кнопку й написати програмний код для обчислення індивідуального завдання на C++ на зразок вищенаведеного прикладу.

4 Оформити протокол лабораторної роботи, до якого занести відповіді на всі контрольні запитання і записати тексти програм. Занести остаточні результати обчислень до протоколу.

Таблиця 3.2 – Індивідуальні завдання

№ вар.	Розмір масиву	Індивідуальне завдання
1	15	Обчислити суму від'ємних елементів масиву
2	10	Обчислити кількість додатних елементів масиву
3	8	Обчислити факторіал значення останнього елемента масиву
4	12	Обчислити добуток елементів, значення яких є менше за 6
5	14	Обчислити суму непарних елементів масиву
6	18	Обчислити суму елементів, абсолютне значення яких не перевищує 1…5
7	11	Обчислити кількість від'ємних елементів масиву
8	14	Обчислити факторіал значення третього елемента масиву
9	16	Віднайти індекс мінімального елемента
10	14	Обчислити кількість елементів масиву, значення яких є більше за значення першого елемента
11	17	Обчислити суму максимального й мінімального елементів масиву
12	9	Обчислити суму елементів, значення яких перебувають в діапазоні [3, 6]
13	15	Віднайти максимальний елемент масиву
14	10	Обчислити добуток непарних елементів масиву
15	8	Обчислити суму мінімального елемента масиву та його індексу
16	12	Обчислити різницю поміж максимальним і мінімальним елементами масиву
17	20	Обчислити кількість непарних елементів масиву
18	18	Обчислити суму квадратів тих чисел, модуль яких не перевищує 3
19	11	Обчислити факторіал значення сьомого елемента масиву
20	9	Обчислити добуток від'ємних елементів масиву
21	16	Обчислити добуток максимального елемента масиву та його індексу
22	19	Обчислити окремо кількість додатних та від'ємних елементів масиву
23	17	Обчислити добуток елементів, значення яких перебувають в діапазоні [2, 5]
24	8	Обчислити суму елементів, значення яких є менше за значення першого елемента масиву
25	7	Обчислити добуток ненульових елементів масиву
26	18	Обчислити суму додатних елементів, значення яких не перевищує 4
27	11	Обчислити добуток мінімального й максимального елементів масиву
28	10	Обчислити суму модулів усіх від'ємних елементів масиву
29	12	Обчислити кількість елементів, значення яких перебувають в діапазоні [4, 7]
30	9	Обчислити добуток елементів, значення яких є менше за значення останнього елемента масиву

Лабораторна робота № 4

Опрацьовування рядків та символьних масивів засобами вбудованого в C++ Builder асемблера

Мета роботи: Вивчення асемблерних команд опрацьовування рядків та набуття навичок застосовування їх при створюванні проектів в C++ Builder із вбудованим асемблером.

1 Основні теоретичні відомості

1.1 Деякі асемблерні команди роботи з рядками та блоками даних

Ці команди використовують для певних операцій над рядками байтів, слів та подвійних слів. За замовчуванням процесор використовує регістри ES:DI, щоби вказати рядки результату, і регістри DS:SI, щоби задати початковий рядок. Програміст повинен забезпечити правильне значення цих регістрів. При виконуванні команд опрацьовування рядків автоматично змінюється значення регістрів DI і/чи SI після опрацювання кожного елемента рядка. Значення регістрів DI, SI збільшується чи зменшується залежно від прапорця DF (DF=1 відповідає зменшенню значення). Значення змінюється на 1 при опрацьовуванні байтів, на 2 – при опрацьовуванні слів і на 4 – при опрацьовуванні подвійних слів.

MOVS < операнд>, <рядок> // Копіювання рядка на адресу операнда
MOVSB // Копіювання рядка байтів з [DS:SI] в [ES:DI]
MOVSW // Копіювання рядка слів з [DS:SI] в [ES:DI]
MOVSD // Копіювання рядка подвійних слів з [DS:SI] в [ES:DI]

LODS <рядок> // Зчитування з рядка в акумулятор (AL, AX чи EAX)
LODSB // Зчитування одного байта з рядка в акумулятор (AL)
LODSW // Зчитування одного слова з рядка в акумулятор (AX)
LODSD // Зчитування подвійного слова з рядка в акумулятор (EAX)

STOS <рядок> // Запис значення акумулятора (AL, AX чи EAX) до рядка
STOSB // Запис значення акумулятора (AL) до рядка
STOSW // Запис значення акумулятора (AX) до рядка
STOSD // Запис значення акумулятора (EAX) до рядка

INS <змінна>, DX // Зчитування рядка з порту (номер порту в DX) у змінну
INSB // Зчитування байта з порту (номер порту в DX) в пам'ять на адресу [ES:DI]
INSW // Зчитування слова з порту (номер порту в DX) в пам'ять на адресу [ES:DI]
INSD // Зчитування двох слів з порту (номер порту в DX) в пам'ять на адресу [ES:DI]

INS <змінна>, DX // Зчитування рядка з порту (номер порту в DX) у змінну
INSB // Зчитування байта з порту (номер порту в DX) в пам'ять на адресу [ES:DI]
INSW // Зчитування слова з порту (номер порту в DX) в пам'ять на адресу [ES:DI]
INSD // Зчитування двох слів з порту (номер порту в DX) в пам'ять на адресу [ES:DI]

OUTS DX, <змінна> // Запис рядка до порту (номер порту в DX) змінної
OUTSB // Запис до порту (номер порту в DX) байта пам'яті з адресою [DS:SI]
OUTSW // Запис до порту (номер порту в DX) слова пам'яті з адресою [DS:SI]
OUTSD // Запис до порту (номер порту в DX) двох слів пам'яті з адресою [DS:SI]

CMPS < операнд>, <рядок> // Порівняння операнда з рядком і встановлення прапорців
CMPSB // Порівняння рядка байтів із [DS:SI] з [ES:DI]
CMPSW // Порівняння рядка слів із [DS:SI] з [ES:DI]
CMPSD // Порівняння рядка подвійних слів із [DS:SI] з [ES:DI]

SCAS <рядок> // Порівняння акумулятора (AL, AX чи EAX) з рядком
SCASB // Порівняння AL з рядком і встановлення прапорців
SCASW // Порівняння AX з рядком і встановлення прапорців
SCASD // Порівняння EAX з рядком і встановлення прапорців

REP // Повторювати наступну команду, встановлену в CX(ECX) кількість разів
REPE // Повторювати наступну команду, допоки прапорець ZF=0
REPNE // Повторювати наступну команду, допоки прапорець ZF≠1
REPZ // Повторювати наступну команду, допоки прапорець ZF=0
REPNZ // Повторювати наступну команду, допоки прапорець ZF≠1

Остання команда є префіксом для повторювання наступної команди опрацювання рядків установлену в регістрі CX (ECX) кількість разів, зменшуючи її при кожному виконанні команди на 1. Окрім того, REPE та REPZ припиняють повторювання команди, якщо прапорець ZF=0, а REPNE і REPNZ припиняє повторювання, якщо прапорець ZF=1. Префікс REP зазвичай використовується з командами MOVS, LODS, STOS, INS та OUTS, а префікси REPE, REPZ, REPNE та REPNZ – з командами CMPS та SCAS.

1.2 Типи рядків в C++ Builder

В C++ є відсутній спеціальний тип рядків. Рядки розглядаються як масиви символів, які завершуються нульовим символом (‘\0’), тобто до символів рядка можна звертатися за їхніми порядковими номерами у цьому рядку. Доступ до рядка може здійснюватись через вказівник на перший символ рядка. Значенням рядка є адреса його першого символу.

Рядок може бути оголошено як масив символів або як змінна типу char*.

Наведемо два приклади еквівалентних оголошень масивів з 15-ти символів:

```
char S[ ] = “рядок символів”;  // де S[0]=’р’, S[1]=’я’, S[6]=’ ’, S[7]=’с’, S[14]=’в’, S[15]=’\0’,
char  *S = “рядок символів”;
```

Для опрацьовування текстів у C++ Builder є рядки. Будь-який рядок вважається за одновимірний масив символів, тобто до символів рядка можна звертатися за їхніми порядковими номерами в цьому рядку.

Рядки можуть бути трьох типів:

1) короткий рядок SmallString<n> (де n <= 255). При оголошуванні змінних такого типу виділяється до 256 байтів (по одному байтові для кожного символу). Цей тип має особливість: перший байт містить значення поточної довжини рядка;

2) довгий рядок AnsiString з нульовим символом (‘\0’) наприкінці (або нуль-термінальний рядок). При оголошуванні змінні типу AnsiString ініціюються порожніми рядками;

3) широкий рядок – WideString – з нульовим символом (‘\0’) наприкінці, в якому для розміщення одного символу відводиться два байти пам’яті. Змінні типів AnsiString чи WideString – це динамічно розташовувані масиви символів, максимальна довжина яких обмежується лише наявністю пам’яті.

1.3 Організація асемблерних функцій

C++ Builder підтримує можливість організації функцій (підпрограм) з асемблерними командами. При організації таких функцій слід дотримуватись усіх відповідних правил мови C++. Особливістю асемблерних функцій є можливість передавання результату виконання функції через регістр-акумулятор (eax), тоді команда return із зазначенням змінної-результату не є обов’язковою. Наприклад, функція обчислення суми двох чисел може мати вигляд:

```
int  sum( int a, int b)
{ asm
  { mov eax, a
    add eax, b
} }
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int a = StrToInt(Edit1->Text),
  b = StrToInt(Edit2->Text);
  int c = sum3(a, b);
  Edit3->Text = IntToStr (c);
 }
```

1.4 Приклади опрацьовування рядків в асемблері

Функція (підпрограма) визначання довжини рядка S типу AnsiString може мати вигляд:

```
int KolSimv(AnsiString S)
{ int k;
  asm
  {  pusha
     mov edi, dword ptr[S]  // Завантажити  початок рядка.
     mov esi, edi           // Запам'ятати початок.
     cmp byte ptr[S], 0     // Якщо перший символ =0 (рядок пустий),
     je @exit               // перейти за позначкою  @exit.
@nex:
    inc edi                 // Перейти на наступний символ рядка.
    cmp byte ptr[edi], 0    // Допоки не кінець рядка (тобто поки не 0-символ)
    jne @nex                // перейти на позначку, тобто повторювати перевірку,
@exit:
     sub edi, esi           //  інакше обчислити довжину рядка.
     mov k, edi
     popa
    }
 return k;
}
```

Ця функція при компілюванні займає 21 байт пам’яті. Порівняно з цим, аналогічна стандартна функція C++ Builder S.Length() при компілюванні займає 69 байт пам’яті. Процедура з викликом функції KolSimv може виглядати, наприклад, як:

```
      void __fastcall TForm1::Button1Click(TObject *Sender)
      { AnsiString S = Edit1->Text;
        int k = KolSimv (S);    // Виклик функції KolSimv.
        Edit2->Text = k;
      }
```

Аналогічна функція визначання довжини короткого рядка типу SmallString<255> може мати вигляд:

```
int KolSim(SmallString<255> S)
{  unsigned char k;
  asm
  { mov al, byte ptr[S] // Завантажити перший байт, який містить значення довжини рядка,
    mov k, al   }       //  в регістр al та вивантажити його з  al  до змінної  k.
  return k;
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{ SmallString <255>  S =  Edit1->Text;
  Edit2->Text = S;
  int k = KolSim(S);    // Виклик функції KolSim
  Edit3->Text = k;
}
```

Функція обчислювання кількості прогалин у рядку типу AnsiString чи заміни їх на знак "+":

```
    int kol_pr(AnsiString s, int k)
    { int p;
      asm
      { pusha
        mov edi, dword ptr [s]  // Завантажити  початок рядка.
        mov ecx, dword ptr k    // Завантажити до лічильника обчислений розмір рядка у байтах.
        mov edx, 0
```

```
        cmp ecx, 0
        je @exit          // Перевірка на порожній рядок
        cld
        mov al, ' '
    @next:
        scasb
        jne @skip
        inc edx           //  Чи mov byte ptr [edi-1], '+' якщо слід замінити усі прогалини на "+"
    @skip:
        loop @next
    @exit:
        mov  dword ptr p, edx   // Вивантажити  обчислену кількість прогалин.
        popa
        }
     return p;
    }
```

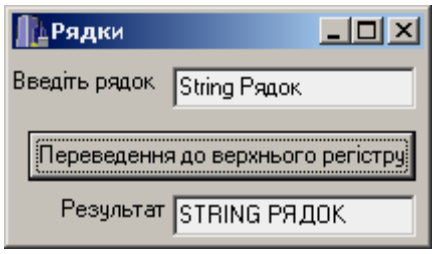
```
    void __fastcall TForm1::Button4Click(TObject *Sender)
    { AnsiString S = Edit1->Text;
      int k = S.Length();
      int p = kol_pr(S,k);
      Edit3->Text = p;
    }
```

Ще один приклад переведення усіх літер рядка до верхнього регістру. У цьому прикладі рядок s оголошено глобально. Оскільки різниця поміж великими й малими літерами дорівнює 32, то перетворювання є можливе в такі способи:

1) командою віднімання sub al,32 чи sub al, 20h

2) командою встановлення 5-го біту в 0 and al,11011111b чи and al,0dfH

```
void U_case_all(AnsiString s)
{ asm
 {  mov edx, dword ptr [s]
 @next1:
   mov al, [edx]      //  Завантажити  символ до регістру al.
   cmp al, "a"
   jb @no             // Код  символу є менше за код "a"?
   cmp al, "z"
   ja @rus       // Якщо символ не є малою латинською літерою, – вийти, інакше –
   sub al, 32    // перетворити символ на велику латинську літеру командою віднімання sub al,32
   jmp @no
 @rus:
   cmp al, "a"
   jb @no
   cmp al, "я"
   ja @no        // Якщо символ не є малою літерою кирилиці, – вийти, інакше – перетворити
   and al,0dfH // малу літеру кирилиці на велику за рахунок встановлення 5-го біта в 0
 @no:            // командою and al,11011111b чи командою sub al, 20h, чи командою  sub al,32.
   mov [edx], al       // Записати перетворену велику літеру на власне місце
   lea edx, [edx+1]  // та перейти до наступного символу, збільшивши адресу на 1,
   cmp al, 0           // допоки ще не кінець рядка, повторювати перевірку.
   jne @next1
  } return ;
}
```




```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ AnsiString S = Edit1->Text;
  U_case_all(S);
  Edit2->Text = S;
}
```

Приклад програми опрацьовування символьної матриці 3х3 й пошуку в ній символів ‘Я’ чи ‘я’:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ char s1[3][3]; unsigned char i, j, k; char cc1;
for (i = 0; i < 3; i++)
for (j = 1; j <= 3; j++) s1[i][j] = Memo1->Lines->Strings[i][j];
asm
{ pusha
  lea edi, s1
  xor ecx, ecx
  mov cl, 9
  mov cc1, 'N'
@next2:
  cmp byte ptr [edi], 'Я'
  jne @NO1
  mov  cc1, 'Y'
@ NO1:
  cmp byte ptr [edi], 'я'
  jne @ NO2
  mov  cc1, 'Y'
@ NO2:
inc edi
  loop @next2
popa
} Edit4->Text = cc1; }
```

2 Контрольні запитання

1 Поясніть, чому не може існувати понад 256 символьних констант.

2 Чому номер 256-ї символьної константи є 255?

3 Запишіть двійковий код символьної константи з десятковим номером 32.

4 Запишіть десятковий номер символьної константи з кодом 01100101.

5 Запишіть команду асемблера, котра надаватиме символьній змінній C значення ‘%’.

6 Запишіть команду асемблера, котра надаватиме символьній змінній S значення ‘–’.

7 Запишіть асемблерні команди перевірки: чи зберігають символьні змінні Y та Z різні символи.

8 Запишіть асемблерні команди перевірки: чи є змінна X цифрою в межах поміж двійкою та сімкою.

3 Лабораторне завдання

1 Розробити програмний проект C++ Builder для введення рядка символів та опрацювання його засобами вбудованого асемблера згідно з індивідуальним завданням відповідно до варіанту.

2 Підчас налагодження програми слід використовувати покрокове виконування команд програми і відстежувати стан регістрів мікропроцесора у вікні дизасемблера.

3 Оформити протокол лабораторної роботи, до якого занести відповіді на всі контрольні запитання і записати тексти програм.

Варіанти індивідуальних завдань

1 Написати програму, котра вводить рядок символів і віднаходить індекс першої коми в цьому рядку.

2 Написати програму, котра вводить символьний рядок символів і замінює в ньому всі коми на прогалини.

3 Написати програму, котра вводить рядок символів і віднаходить кількість цифр у цьому рядку.

4 Написати програму, котра вводить рядок символів і замінює всі цифри на прогалини.

5 Написати програму, котра вводить рядок символів і змінює місцями перший та останній введені символи.

6 Написати програму, котра вводить рядок символів і віднаходить індекс першої крапки у цьому рядку.

7 Написати програму, котра вводить рядок символів і замінює в ньому всі крапки та коми на знак ‘%’.

8 Написати програму, котра вводить рядок символів і віднаходить серед них елемент з найменшим номером у ANSI-таблиці.

9 Написати програму, котра вводить символьний рядок символів і будує інший рядок, який міститиме лише цифри з цього рядка.

10 Написати програму, котра вводить рядок символів і змінює місцями третій та останній введені символи.

11 Написати програму, котра вводить рядок символів і віднаходить індекс першого знаку “+” у цьому рядку.

12 Написати програму, котра вводить рядок символів і замінює в ньому всі цифри на знак ‘#‘.

13 Написати програму, котра вводить дві символьні послідовності й будує нову послідовність, на початку якої стоять елементи першої послідовності, а за ними – елементи другої.

14 Написати програму, котра вводить рядок символів і будує новий масив, який міститиме лише великі латинські літери з першого рядка.

15 Написати програму, котра вводить символьну матрицю розміром 3х4 і визначає, чи містить ця матриця хоча б одну цифру.

16 Написати програму, котра вводить символьну матрицю розміром 3х3 і будує послідовність, котра співпадає з діагоналлю матриці.

17 Написати програму, котра вводить рядок символів і віднаходить кількість знаків ‘–‘ у цьому рядку.

18 Написати програму, котра вводить символьну послідовність і визначає, чи є в ній символи Ж або ж.

19 Написати програму, котра вводить символьну матрицю розміром 3х4 і будує послідовність, котра співпадає з другим рядком матриці.

20 Написати програму, котра вводить символьну матрицю розміром 3х4 і визначає, чи є у введеній матриці символи Ф та ф.

21 Написати програму, котра вводить символьну матрицю розміром 3х4 і будує послідовність, котра співпадає з четвертим стовпчиком матриці.

22 Написати програму, котра вводить символьну матрицю розміром 3х3 і визначає символ із найбільшим ANSI-номером.

23 Написати програму, котра вводить символьну послідовність і упорядковує елементи введеної послідовності за зростанням кодів ANSI.

24 Написати програму, котра вводить символьну послідовність і замінює в рядку великі літери російської абетки на малі.

25 Написати програму, котра вводить символьну матрицю розміром 3х2 і будує послідовність, котра співпадає з першим стовпчиком матриці.

26 Написати програму, котра вводить рядок символів і віднаходить кількість літер російської абетки у цьому рядку.

27 Написати програму, котра вводить рядок символів і змінює місцями другий та передостанній введені символи.

28 Написати програму, котра вводить рядок символів і віднаходить індекс першої цифри у цьому рядку.

29 Написати програму, котра вводить рядок символів і замінює в ньому всі прогалини на знак ‘+’.

30 Написати програму, котра вводить рядок символів і віднаходить кількість символів, які не є літерами у цьому рядку.

Лабораторна робота № 5

Програмування портів у вбудованому в C++ Builder асемблері

Мета роботи: Вивчення асемблерних команд роботи з портами пристроїв ПК та набуття навичок застосовування їх при створюванні проектів в C++ Builder із вбудованим асемблером.

1 Основні теоретичні відомості

1.1 Команди асемблера роботи з портами

На командах IN та OUT будується спілкування процесора з пристроями введення/виведення – клавіатурою, вінчестером, різними контролерами тощо, і використовуються ці команди, першою чергою, в драйверах пристроїв.

- **IN <акумулятор>, <порт>**

Передати дані з порту до регістру-акумулятора (AL, AX чи EAX). Номер порту введення/виведення задається числом в межах від 0 до 255 або адресується регістром DX, який попередньо повинен бути завантажений потрібним значенням, що забезпечить доступ до портів з номерами від 0 до 65 535.

- **OUT <порт>, <акумулятор>**

Передати до порту зміст акумулятора. Номер порту задається так само, як і для команди IN.

Слід зауважити, що при безпосередній роботі з портами слід бути надто обережним, саме тому, починаючи з Windows 95/98, пропонується замість прямого звертання до портів використовувати функції WinAPI. Більш того, Windows NT/XP для власного захисту обмежує використовування команд IN та OUT при написанні програм. Для тих розробників програм, які бажають набути доступ до цих команд у своїх програмах, слід використовувати спеціальні драйвери, наприклад, GiveIO, що не є бажано, надто для тривалого терміну, оскільки при цьому "вимикається" механізм захисту Windows, а це може призвести до збою синхронізації драйверів та пристроїв і, в решті решт, краху системи. Всі нижче запропоновані до розгляду приклади було розроблено під Windows 98.

1.2 Годинник реального часу RTC та CMOS-пам'ять

В кожному комп'ютері є мікросхема, яка відповідає за підтримку поточної дати і часу. Для того щоби їхні значення не було втрачено за кожного вимкнення, на мікросхемі розташовано невелику область (від 64 до 128 байт), виконану за технологією CMOS, що дозволяє знизити енергоспоживання до мінімуму. Мікросхема живиться від акумулятора, розташованого на материнській платі, і не вимикається при вимиканні комп'ютера. Для зберігання часу вистачає всього 14 байт такої енергонезалежної пам'яті, а її решта використовується BIOS, щоби зберігати різноманітну інформацію, потрібну задля коректного запускання комп'ютера. Для спілкування з CMOS та регістрами RTC виділяються порти введення/виведення від 70h до 7Fh, але лише призначення портів 70h та 71h є однакове для всіх материнських плат.

Порт 70h для запису: індекс для вибору регістру CMOS.

Порт 71h для зчитування і запису: дані CMOS.

Після запису до порту 70h слід здійснити запис чи зчитування з порту 71h, інакше RTC опиниться у невизначеному стані.

Вміст регістрів CMOS варіюється для різних BIOS, але перші 33h регістри зазвичай виконують одні й ті самі функції. Ось деякі з цих функцій:

00h: RTC: поточна секунда (**00-59h** чи 00 – 3Bh) – формат обирається регістром 0Bh, за замовчуванням – BCD;

02h: RTC: поточна хвилина (**00-59h** чи 00 – 3Bh);

04h: RTC: поточна година**: 00-23h** / 00 – 17h (24-годинний режим), 01-12h / 00 – 1Ch (12-годинний режим до полудня), 81-92h / 81 – 8Ch (12-годинний режим після полудня)

06h: RTC: поточний день тижня (1-7, 1 – неділя);

07h: RTC: поточний день місяця (**01-31h** / 01h–0Fh);

08h: RTC: поточний місяць року (**01-12h** / 01h–0Ch);

09h: RTC: поточний рік (останні дві цифри) (**01-99h** / 00h–63h);

0Bh: регістр стану B:

біт 2: 1/0: формат дати та часу двійковий / BCD;

біт 1: 1/0: 24-годинний / 12-годинний режим;

біт 0: 1/0: автоматичний перехід на літній та зимовий час;

32h: перші дві цифри року (в BCD-формі).

Для того щоби встановити BCD-форму, слід виконати:

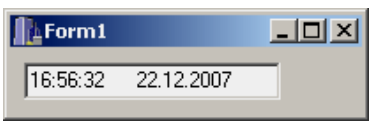
```
mov al, 0Bh          // Керувальний регістр – 0Bh
out 70h, al
in al, 71h
and al, 11111011b    // Обнулити біт 2 (форма чисел – BCD)
out 71h, al          // і записати нове значення назад до порту.
```

Приклад організації асемблерної підпрограми та її виклику для зчитування поточного часу й дати через порти CMOS

Номер функції передається вхідним параметром у підпрограму, а результатом її виконання буде двозначна цифра, перетворена на рядок символів задля зручності подальшого виведення на форму.


```
AnsiString CMOS_time(byte x)
{ byte y1, y2;
 asm
  {  mov al, X            // Надіслати до акумулятора номер функції.
     out 70h, al          // Надіслати al до індексного порту CMOS.
     in al, 71h           // Зчитати дані.
     push ax              // Зберегти вміст акумулятора ax у стеку.
     shr al,4             // Виділити старші 4 біти, якщо BCD-форма add al,'0'.
     mov byte ptr y1,al // Вивантажити першу цифру до змінної y1.
     pop ax               // Відновити вміст акумулятора ax зі стека.
     and al, 0fh          // Виділити молодші 4 біти (якщо BCD-форма add al,30h).
     mov byte ptr y2,al // Вивантажити другу цифру до змінної y2.
  }
 return  IntToStr(y1) + IntToStr(y2);
}
```

```
void __fastcall TForm1::Timer1Timer(TObject *Sender) // Програма для компонента Timer1
{ AnsiString h, min, sec, d, m, y1, y2;
  h = CMOS_time(4);     // функція 4 для зчитування значення поточної системної години
  min = CMOS_time(2); // поточна хвилина
  sec = CMOS_time(0); // поточна секунда
  d = CMOS_time(7);     // поточний день місяця
  m = CMOS_time(8);     // поточний місяць року
  y1 = CMOS_time(50); // перші дві цифри року (функція 32h(50 - десяткове))
  y2 = CMOS_time(9);    // останні дві цифри року
  Edit1->Text = h+':'+min+':'+sec+"       "+d+ ' .' + m + '.' +y1+y2; // 16:56:32   22.12.2007
}
```



1.3 Клавіатура

Контролерові клавіатури відповідають порти з номерами від 60h до 6Fh, хоча для всіх стандартних операцій вистачить портів 60h та 61h.

60h – порт для зчитування даних клавіатури. При зчитуванні з нього можна отримати скан-код останньої натисненої клавіші. Саме в такий спосіб найоптимальніше реалізовувати резидентні програми задля перехоплення переривання IRQ1, тому що за цим кодом можна визначати момент натискання й відпускання якої завгодно клавіші, включаючи такі як Shift, Ctrl, Alt (або навіть pause – скан-код 80h).

60h – порт для запису – регістр керування клавіатурою. Якщо біт 1 в порту 64h дорівнює 0, байт, записаний до порту 60h, інтерпретується як команда. Наприклад, команда **0EDh - змінює стан світлодіодів клавіатури**. Другий байт цієї команди визначає новий стан: біт 0 – стан ScrollLock (1– увімкн, 0 – вимкн), біт 1 – NumLock, біт 2 – CapsLock.

Організація функції та приклади програм задля почергового щосекундного перемикання світлодіодів клавіатури

Асемблерна функція вмикання заданого в регістрі cl стану світлодіодів клавіатури:

```
Change_leds()
{ asm
  { mov  al, 0edh       // Команда клавіатури 0EDh.
    out 60h, al
    mov  al, cl
    out 60h, al          // Новий стан світло діодів.
} }
```

Приклад програми 1 – буде виконувати три повних перемикання світлодіодів по колу:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{  asm
   { push di
     xor   di, di          // Обнулити лічильник перемикань.
     mov ah, 2             // Функція 2 переривання  1Ah
     int    1ah            // зчитує поточний час.
     mov ch, dh            // Запам'ятати  поточну секунду.
     mov cl, 0010b         // cl = стан світлодіодів клавіатури, першим увімкнути NumLock.
 @next:
         call Change_leds      // Виклик функції встановлення світлодіодів відповідно до cl.
         shl cl, 1             // Перейти на наступний світлодіод.
         test cl, 1000b        // Якщо 1 вийшла до біта 3,
         jz @continue
         mov cl, 0001b         // повернути її назад до біта 0.
         inc di
         cmp  di, 3            // Якщо кількість повних перемикань по колу
         je @exit              // дорівнює 3, вийти з програми.
     @continue:
              push cx
              mov ah,2         // Функція 2 переривання  1Ah
              int 1Ah          // зчитує поточний час.
              pop cx
              cmp ch, dh       // Порівняти поточну секунду в dh  з  ch
              mov ch, dh       // і запам'ятати  її .
         je @continue          // Якщо це була та сама секунда, – не перемикати світлодіоди,
         jmp @next             // інакше – перемкнути світлодіоди.
  @exit: pop di
   }
}
```

Приклад програми 2 – буде виконувати задану кількість перемикань світлодіодів:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{ short  kol = StrToInt(Edit1->Text); // Ввести кількість перемикань kol.
  asm
   { push di
     xor di, di            // Обнулити лічильник перемикань.
     mov ah,2              // Функція 2 переривання  1Ah
     int 1ah               // зчитує поточний системний час.
     mov ch, dh            // Запам'ятати  поточну секунду в регістрі ch.
     mov  cl, 0010b        // cl = стан світлодіодів клавіатури, першим увімкнути NumLock.
@next: cmp  di, kol              // Якщо кількість перемикань (секунд)
         je @exit                // дорівнює kol, вийти із програми.
         inc di
         call Change_leds        // Виклик функції встановлення світлодіодів відповідно до cl.
         shl cl, 1               // Перейти на наступний світлодіод.
         test cl, 1000b          // Якщо 1 вийшла до біта 3,
         jz @contin
         mov cl, 0001b           // повернути її назад до біта 0.
     @contin:
              push cx
              mov ah,2           // Функція 2 переривання  1Ah
              int 1Ah            // зчитує поточний час.
              pop cx
              cmp ch, dh         // Порівняти поточну секунду в dh  з  ch
              mov ch, dh         // і запам'ятати  її.
         je @contin              // Якщо це була та сама секунда, – не перемикати світлодіоди,
         jmp @next               // інакше – перемкнути світлодіоди.
  @exit: pop di
  }
}
```

1.4 Системний таймер. Динамік

Системний таймер викликає переривання IRQ0 приблизно 18,2 рази за секунду. Програмований інтервальний таймер – надто складна система, яка складається з трьох частин – трьох каналів таймера, кожен з яких можна запрограмувати для роботи в одному з шести режимів. Й, більш того, на багатьох сучасних материнських платах розташовуються два таких таймери, тобто кількість каналів дорівнює шести. Задля власних потреб програми можуть використовувати канал 2 (якщо їм не потрібен динамік) і канал 4 (якщо є другий таймер). За необхідності можна перепрограмувати й канал 0, але потім його слід буде повернути до початкового стану, щоби BIOS та DOS могли продовжувати роботу.

В просторі портів введення/виведення для таймера виділено область від 40h до 5fh:

порт 40h – канал 0 (генерує IRQ0);

порт 41h – канал 1 (підтримує оновлювання пам'яті);

порт 42h – канал 2 (керує динаміком);

порт 43h – керувальний регістр першого таймера;

порт 44h - 47h – другий таймер комп'ютерів з шиною MicroChannel;

порт 48h – 4Bh – другий таймер комп'ютерів з шиною EISA.

Все керування таймером здійснюється шляхом виведення одного байта до порту 43h (для першого таймера). Розглянемо призначення бітів у цьому байті:

біти 7-6: якщо не 11 – це номер каналу, який буде програмуватися ($00_2=0_{10}$, $01=1$, $\mathbf{10}_2=\mathbf{2}_{10}$);

біти 5-4:

00 – зафіксувати поточне значення лічильника для зчитування (біти 3-0 не використовуються);

01 – зчитування/запис лише молодшого байта;

10 – зчитування/запис лише старшого байта;

11 – зчитування/запис спочатку молодшого, а потім і старшого байта;

біти 3-1: режим роботи каналу

000 – переривання RQ0 за досягнення нуля;

001 – очікуваний мультивібратор;

010 – генератор імпульсів;

011 – генератор прямокутних імпульсів;

біт 0: формат лічильника 0 – двійкове 16-бітове число (0-0FFFFh), 1 – двійково-десяткове число (0-9999)

Канал 2 системного таймера керує динаміком ПК – він генерує прямокутні імпульси з частотою 1 193 180 / (початкове значення лічильника). При програмуванні динаміка початкове значення лічильника таймера називається дільником частоти: мається на увазі, що динамік працює з частотою (1 193 180/дільник) герців. Після програмування каналу другого таймера слід увімкнути динамік шляхом установлення бітів 0 та 1 порту 61h в 1. Біт 0 фактично дозволяє роботу даного каналу таймера, а біт 1 – вмикає динамік.

Приклад організації асемблерної програми керування динаміком ПК

```
void __fastcall TForm1::Button1Click(TObject *Sender) // увімкнути динамік
{ asm
  { mov al,10110110b // Канал 2, режим 3.
    out 43h,al
    mov al,0dh       // Молодший байт дільника частоти 11D0h(4560)  – частота 261 Гц –
                     //  нота MI середньої октави.
    out 42h,al
    mov al,11h       // Старший байт дільника частоти.
    out 42h,al
    in al,61h        // Зчитати поточний стан порту 61h в al
    or al,00000011b  // і встановити 0 та 1 біти в 1, –
    out 61h,al       // тепер динамік увімкнено.
  }
}
```

```
void __fastcall TForm1::Button2Click(TObject *Sender) // вимкнути динамік
{ asm
  {  in al, 61h          // Зчитати поточний стан порту 61h в al
    and al,11111100b  // і обнулити молодші два біти, –
    out 61h, al          // тепер динамік вимкнено.
   }
}
```

```
// функція  увімкнення динаміка на kol секунд із заданою частотою звучання
// (k1 – молодший байт дільника частоти, k2 – старший байт дільника частоти)
dinamic(unsigned char kol, unsigned char k1, unsigned char k2)
{ asm
  {  mov al, 10110110b       // Канал 2, режим 3.
    out 43h, al
    mov al, k1          // Молодший байт дільника частоти
    out 42h, al
    mov al, k2          // Старший байт дільника частоти.
    out 42h, al
    in al, 61h          // Зчитати поточний стан порту 61h в al
    or al, 00000011b    // і встановити 0 та 1 біти в 1, –
    out 61h, al         // тепер динамік увімкнено.
    xor cx, cx
    mov ah, 2
    int 1ah             // Зчитати поточний системний час і
```

```
    mov ch, dh          // запам'ятати  поточну секунду в регістрі ch.
@nex:
    push cx
    mov ah, 2
    int 1ah
    pop cx
    sub dh, ch          // Обчислити кількість секунд звучання.
    cmp dh, kol         // Якщо час не вийшов,
    jb @nex             // продовжити звучання,
     in al, 61h         // інакше – вимкнути динамік.
    and al, 11111100b
    out 61h, al
   }
}
```

```
// Приклад виклику функції увімкнення динаміка на kol секунд із заданою частотою
// звучання:(k1 – молодший байт дільника частоти, k2 – старший байт дільника частоти)
void __fastcall TForm1::Button3Click(TObject *Sender)
{ unsigned char kol = StrToInt(Edit1->Text),
   k1 = StrToInt(Edit2->Text),
   // старший байт дільника частоти k2 змінюється від 1 до 20 із кроком 5
   for(k2 = 1; k2 <= 20; k2 += 5)
      dinamic(kol, k1, k2);
   }
```

2 Контрольні запитання

1 Запишіть команди асемблера для передачі до порту з номером 70h значення 255.
2 Як записати команду асемблера для передачі даних з порту 70h?

3 Лабораторне завдання

Розробити програмний проект в C++ Builder для роботи з тим чи іншим портом комп'ютера (за вказівкою викладача) засобами вбудованого асемблера. Рівень складності виконуваної роботи визначає викладач.

Список рекомендованої літератури

1 **Магда Ю.С.** Ассемблер. Разработка и оптимизация Windows-приложений. – БХВ-Петербург, 2003. – 544 с.

2 **Джордейн Р.** Справочник программиста персональных компьютеров типа IBM PC, XT, AT. – М.: Финансы и статистика, 1992. – 544с.

3 **Скэнлон Л.** Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблер. – Москва: Радио и связь, 1989. – 336с.

4 **Абель П.** Язык ассемблера для IBM PC и программирования. Пер. с англ. – М.: Высшая школа, 1992. – 447с.

5 **Юров В., Хорошенко С.** Assembler: учебный курс. – Спб: Питер Ком, 1999. – 672 с.

6 **Сван Т.** Освоение Turbo Assembler / Пер. с англ. – К.-М.-Спб.: Диалектика, 1996.-544 с.

7 **Майко Г.В.** Assembler для IBM PC. – М.: Бизнес-информ, 1999. – 212 с.

ЗМІСТ

Редактор *І.В. Ращупкіна*
Комп'ютерне макетування *Ж.А. Гардиман*

Здано в набір 16.10.2007 Підписано до друку 27.11.2007
Формат 60/88/16 Зам. № 33
Тираж 60 прим. Обсяг 3,6 друк. арк.
Віддруковано на видавничому устаткуванні фірми RISO
у друкарні редакційно-видавничого центру ОНАЗ ім. О.С. Попова
ОНАЗ, 2007