ПРАКТИЧНЕ ЗАНЯТТЯ 5
ПРОГРАМУВАННЯ ЗАДАЧ ОБРОБКИ ДАНИХ МОВОЮ АСЕМБЛЕРА. ТРАНСЛЯЦІЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМ.

5.1 Мета заняття

Познайомитися з функціональними можливостями системи програмування мовою Асемблера, структурою програми, основними командами і директивами мови програмування Асемблера; отримати навички створення програм мовою Асемблера та їх налагодження.

5.2 Методичні вказівки з організації самостійної роботи студентів

5.2.1 Системне програмування

Системне програмування – це процес розробки системних програм (керуючих, обслуговуючих, тощо).

На відміну від прикладного, системне програмне забезпечення повинно уміти працювати напряму з ядром операційної системи (ОС), бути стабільним, забезпечувати швидку реакцію, займати як найменше ресурсів обчислювальної системи. Для забезпечення цих вимог системний програміст повинен чітко представляти структуру операційної системи, її підсистем, засоби взаємодії між підсистемами ОС та взаємодії ОС з апаратним забезпеченням, алгоритми роботи кожної підсистеми. Крім того, системний програміст повинен бути добре знайомий з програмними інтерфейсами, які пропонує операційна система для забезпечення підтримки прикладних і системних програм.

На сьогоднішній день жодна мова програмування не може претендувати на те, щоб на ній можливо було написати "все", окрім мови Асемблера. Таким чином на Асемблері пишуть:

- все, що потребує максимальної швидкості виконання: основні компоненти комп'ютерних ігор, ядра операційних систем реального часу, критичні ділянки програм;
- все, що взаємодіє з зовнішніми пристроями: драйвери, програми, які працюють напряму з портами, звуковими та відео платами;
- все, що використовує повністю можливості процесора: ядра багатозадачних операційних систем, DPMI-сервери та будь-які програми, які переводять процесор у захисний режим;
- все, що повністю використовує можливості операційної системи: віруси ті антивіруси, захист від несанкціонованого доступу, програми, які обходять цей захист, і програми, які захищають від таких програм;
- та багато іншого.

Таким чином, велику кількість того, що пишуть на мовах високого рівня, краще, простіше та скоріше писати мовою Асемблера.

5.2.2 Програмний пакет MASM

Програмний пакет MASM був створений для написання програм мовою Асемблера та поєднує у собі зручність засобів програмування, притаманних мовам високого рівня, і традиційних засобів машинно-орієнтованих мов. За

допомогою пакету MASM розробка програм виконується традиційним для асемблерного програмування засобом – запуском окремих програм трансляції, компонування та відлагодження. Для цього використовуються програми ml.exe, link.exe та cv.exe. Трансляція вихідного файлу відбувається програмою ml.exe. У результаті роботи транслятора створюються файл об’єктного модулю і файл тексту програми, які містять різну інформацію про програму: об’єктний код, повідомлення про синтаксичні помилки, таблицю символі та ін.

Після отримання коректного об’єктного модулю програму необхідно скомпонувати. Для цього застосовується утиліта-компонувальник link.exe, основне призначення якої – дозвіл зовнішніх посилань. Якщо цільова програма складається з кількох окремо трансльованих модулів і вони містять взаємні посилання на змінні або модулі, компонувальник дозволяє їх, формуючи правильні адреси, що переміщуються. Результатом роботи компонувальника є модуль виконання (завантажувальний) з розширенням .exe. Якщо програмний код не містить логічних помилок, програма буде працювати правильно, якщо ні – для їх пошуку використовую спеціальні програмні засоби.

5.2.3 Середовище для відлагодження програмного коду OllyDbg

Для пошуку і усунення логічних помилок призначено спеціальне програмне забезпечення, яке дозволяє швидко позбутися більшості проблем. За його допомогою користувач може здійснювати покрокове виконання програми, контролювати стан всіх регістрів процесора, змінних, список модулів, які використовуються операційною системою, переглядати адресу будь-якого осередку пам'яті та ін. Середовище також дозволяє визначати параметри процедури, цикли, відокремлювати константи, масиви, рядки. Середовище підтримує всі процесори 80x86 та розрізняє велику кількість числових форматів.

Графічний інтерфейс середовища (рис. 1) складається з чотирьох основних вікон: вікно дезасемблера (ліве верхнє), вікно даних (ліве нижнє), вікно регістрів (праве верхнє), вікно стеку (праве нижнє). Повний перелік вікон, які можна використовувати під час роботи середовища, знаходиться у меню **View.**

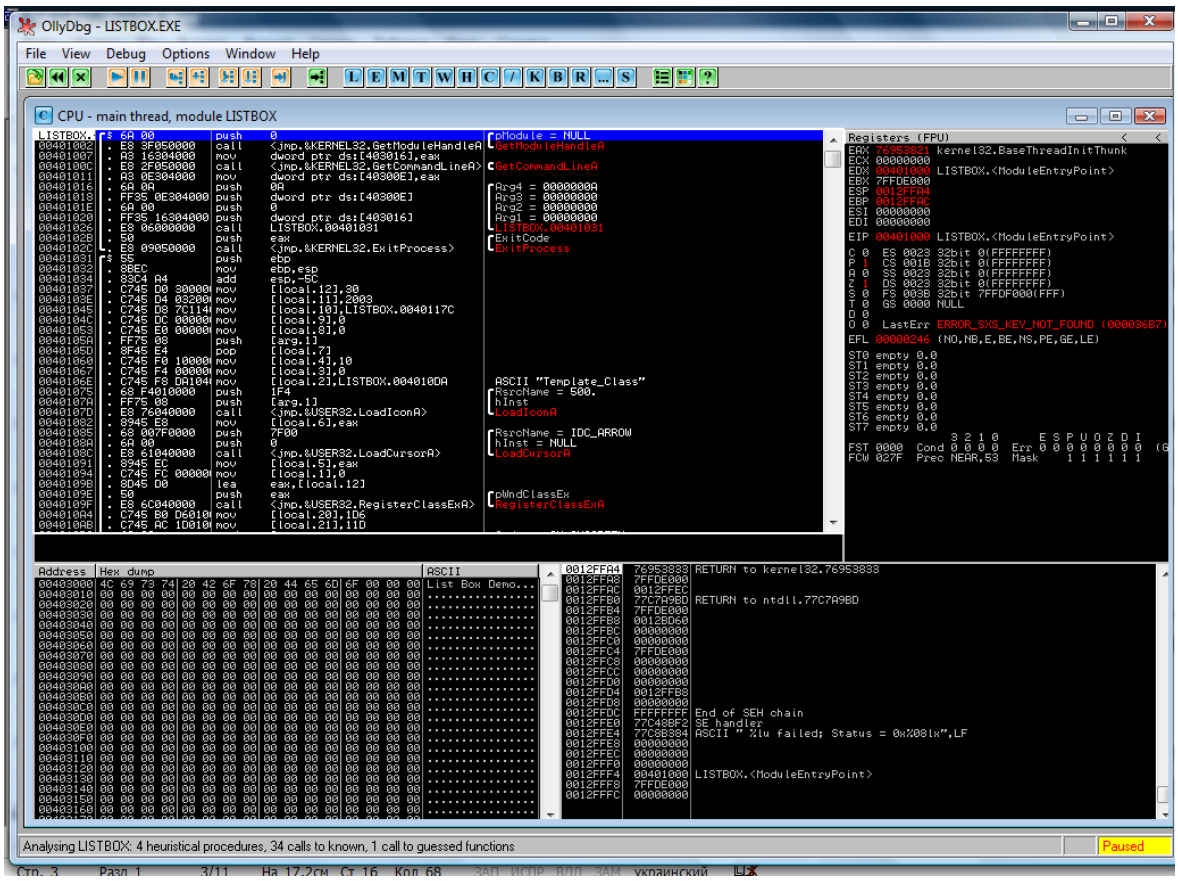

Рис. 1 Графічний інтерфейс середовища для відлагодження програмного коду OllyDbg1.10.

Вікно дезасемблера складається з чотирьох стовпчиків:

– **Address** – стовпчик відображує віртуальну адресу команди, яку вона отримує при завантаженні модуля до пам'яті. Подвійне натискання лівої клавіші миші на цей стовпчик переводить усі адреси у зсуви відносно поточної адреси ($, $-2, $+4, т. ін.)

– **Hexdump** – стовпчик відображує коди команд і значення операндів. Крім того, стовпчик містить різні значки, що допомагають розібратися з логікою програми: вказують на команду, до якої є переходи (>), команду, що здійснює перехід (^ - догори, ˅ - донизу) та ін. У цьому стовпчику також позначені цикли, які розпізнала програма. Подвійне натискання лівої клавіші миші на цей стовпчик призводить до встановлення точки зупинки (адреса буде позначена червоним кольором).

– **Disassembly** – стовпчик містить асемблерну позначку команди. Подвійне натискання лівої клавіші миші на цей стовпчик призводить до появи вікна редагування асемблерної команди (далі після виправлення у роботі програми буде приймати участь виправлена команда). Також виправлений текст програми можна записати до модулю, що виконується.

– **Comment** – стовпчик містить додаткову інформацію про код (API-функції, бібліотечні функції та ін.). Подвійне натискання лівої клавіші миші на цей стовпчик дозволяє додавати коментарі до кожного рядка коду.

Вікно даних має три стовпчики: стовпчик адреси (**Address**), стовпчик шістнадцятирічних значень осередку (**Hexdump**), стовпчик текстової інтерпретації змісту осередків (**ASCII, Unicode** та ін.). Змінювати можна сенс другого та третього стовпчика (наприклад: інтерпретувати уміст осередків за кодуванням Unicode).

Вікно регістрів може містити три можливих набори: стандартні регістри та регістри співпроцесора, стандартні регістри та регістри MMX, стандартні регістри та регістри за технологією 3DNow. Подвійне натискання лівої клавіші миші на цей стовпчик дозволяє редагувати уміст відсутнього регістру.

Вікно стеку надає зміст стеку. Перший стовпчик (**Address**) містить адресу осередку у стеку, другий (**Value**) – зміст осередку, третій (**Comment**) – можливі коментарі до змісту.

При натисканні правої клавіші миші на будь-яке вікно можна отримати контекстне меню, індивідуальне для кожного вікна.

Уміст вікон **не є** незалежним. Натиснувши правою клавішею миші на один з регістрів завжди можливо перекласти його уміст як адресу в області даних або в області стеку.

Засоби виконання програми:

- покрокове виконання в обхід процедури (**stepover**). При натисканні клавіші **<F8>** виконується поточна команда (у інших вікнах при цьому відображується зміна умісту регістрів, секцій даних, секцій стеку). Якщо наступною командою буде виклик процедури (**CALL**), то автоматично будуть виконуватися усі команди процедури (як одна інструкція);
- покрокове виконання з заходом до процедури (**stepinto**), клавіша **<F7>**. Якщо наступною командою буде виклик процедури (**CALL**), далі буде покрокове виконання інструкції процедури;
- використання анімації (**animation**) – автоматизація покрокового виконання команд (**stepover)** та **(stepinto**) у автоматичному режимі одна за другою з невеликою затримкою (комбінації **<Ctrl>+<F8>** та **<Ctrl>+<F7>).** При виконанні вікна середовища оновлюються. Виконання можна призупинити (клавіша **<Esc>**), також виконання призупиняється у точках зупинки, та у випадку, якщо програма генерує виключення;
- використання трасування (**trace**). Від анімації відрізняється тим, що при виконанні вікна середовища неп оновлюються. Комбінації клавіш **<Ctrl>+<F12>**та **<Ctrl>+<F11>**. Після кожної команди інформація про її виконання додається у спеціальний трасувальний буфер (перегляд – **View/Runtrace**). Також можна визначити умови. За якими буде відбуватися зупинка трасування – точка зупинки (комбінація **<Ctrl>+<T>**). При цьому можна вказати: діапазон адрес, у якому буде зупинка, умовні вислови (наприклад EAX>100000), при виконанні яких трасування буде зупинено, номер команди або набір команд, за якими відбудеться зупинка. Якщо процес трасування привів до системного коду, можна вийти з нього (**executetillusercode**) за допомогою комбінації **<Alt>+<F9>**. Можна виконувати код, доки не зустрінеться повернення з процедури

(**executetillreturn**), тобто буде виконаний весь код поточної процедури та повернення з неї (**<Ctrl>+<F9>**).

5.2.4 Загальна структура програми

Для набору коду програми мовою Асемблера у програмному пакеті MASM32 є власне середовище для набору, трансляції та компонування програмного коду, яке запускається \masm32\QEDITOR.exe. Загальна структура програми має наступний вигляд:

```
   .486
.model flat, stdcall
option casemap :none   ; з урахуванням регістру

; підключення зовнішніх бібліотек та їх опис
include \masm32\include\windows.inc
include \masm32\include\masm32.inc
include \masm32\include\kernel32.inc
include \masm32\macros\macros.asm
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\kernel32.lib

.data
; Ініціалізовані дані
.data?
; Неініціалізовані дані
.const
; Константи

.code

start:

; Код програми

end start
```

.486 – це директива мови Асемблера, що вказує використовувати набір операцій для процесора 80486. Можна використовувати .386 або .586. Також існує два ідентичних вибори для кожного варіанту CPU. .386/.386p, .486/.486p. Але "p"-версії необхідні тільки у випадку, коли програма використовує привілейовані інструкції, тобто зарезервовані процесором/операційною системою для захищеного режиму. Такі інструкції можуть бути використані тільки у захищеному коді, наприклад, vdx-драйверами.

.model – це директива, яка визначає модель пам'яті та вказує на середовище, у якому буде "жити" програма. Тип моделі "flat" (пласка) вказує

на Win32. У 16-розрядному програмуванні сегменти були необхідні. Ця проблема була вирішена у 32-розрядному програмуванні (Windows 95 та вище). Сегменти все ще існують, але вони вже не 64kb (як у 16-розрядному), а 4 Гб. Windows може навіть "зависнути", якщо намагатися змінити один з сегментних регістрів. Саме це називається пласкою (flat) моделлю пам'яті. Існують тільки зсуви і вони тепер 32-розрядні (діапазон від 1 до 4,294,967,295). Кожний осередок у пам'яті вказується зсувом.

stdcall – визначає порядок передачі параметрів (з ліва направо або навпаки), а також, хто урівнює стек після виклику функції.

Для Win16 існує два типи передачі параметрів: C та PASCAL. При C-передачі параметри передаються з права наліво (самій перший параметр розташовується у стеку першим). Стек урівнює той, хто здійснює виклик. Наприклад, під час виклику функції з ім'ям foo(intfirst_param, intsecond_param, intthird_param), при використанні C-передачі параметрів, код буде мати вигляд:

```
push [third_param]  ; Помістити до стеку третій параметр
push [second_param] ; Наступний - другий
push [first_param]  ; Останній - перший
callfoo
addsp, 12         ; Виклик урівнює стек
```

PASCAL-передача параметрів відбувається з ліва направо і стек урівнює той, кого викликають.

Такий порядок передачі даних використовується у Win16 для зменшення кількості програмного коду. C-передача потрібна у випадках, коли не відомо скільки параметрів буде передано функції (наприклад, wsprintf() функція не може знати, скільки параметрів буде відправлено до стеку, тому не може його зрівняти). STDCALL – гібрид C та PASCAL, дані передаються з права наліво, але урівнює стек той, кого викликають. Платформа Win32 використовує тількиstdcall, за виняткомwsprintf(), тоді потрібно використовувати C-передачу параметрів.

option – директива, потрібна для налаштування компіляції. Опція casemap вказує на чуттєвість до регістру символів, ":none" – встановлення чуттєвості (наприклад, ExitProcess та exitprocess – це різні імена). Це потрібно, щоб уникнути конфліктів між файлами, що були додані від різних авторів.

Одна з головних переваг MASM це макрос invoke, який дозволяє викликати API-функції звичайним чином з перевіркою кількості та типу параметрів.

Програмний пакет MASM припускає об'явлення доданих файлів, макросів та ін. Для цього використовується директива include, після якої вказується ім'я потрібного файлу (наприклад, include \masm32\include\ windows.inc) Файл windows.inc містить у собі визначення констант і структур, які потрібні для програмування у Win32, але не містить прототипів

функцій, тому необхідно підключити деякі інші файли з директорії \masm32\ include.

Прототип кожної з бібліотек існує у однойменних файлах директорії include. У прикладі викликається функція, експортована з kernel32.dll, для цього потрібно підключити прототипи функції з kernel32.dll. Це файл - kernel32.inc, який складається з прототипів функцій з відповідної dll. Якщо не підключити kernel32.inc, функцію ExitProcess можливо буде викликати тільки за допомогою команди call, але не за допомогою invoke (тоді потрібно було б розташувати у вихідному коді прототип самого invoke). Таким чином, файли підключення позбавляють користувача необхідності самостійно створювати прототипи.

На відміну від include директива includelib вказує шлях до статичних бібліотек і є лише засобом вказати, які бібліотеки, використані програмою необхідно прилінкувати.

У Win32 не має сегментів, тому адресний простір можна поділити на логічні секції. Початок одної секції відзначає кінець попередньої. Існує дві групи секцій: даних і коду.

Секції даних:

.data – секція, яка містить ініціалізовані дані програми і додається до файлу виконання.

.data? – секція, яка містить неініціалізовані дані програми (інколи буває потрібно тільки "попередньо" виділити деяку кількість пам'яті). Перевага неініціалізованих даних полягає у тому, що вони не займають місце у файлі виконання (наприклад, при виділенні 10 000 у секції .dataexe-файл не збільшиться на 10 kb, це лише вказівка для компілятора на те, скільки місця потрібно, коли програма завантажиться у пам'ять).

.const – секція, яка містить об'яву констант.

Секція коду:

.code – секція, яка містить код.

При написанні коду необхідно чітко розрізняти, коли відбувається операція з адресами, а коли зі значенням операнду. Крім того, необхідно звертати увагу на відповідність розмірів операндів у команді (якщо до регістру процесора потрібно занести адресу змінної, то необхідно використовувати 32-розрядний регістр, якщо використовується мітка, що вказує на 16 розрядну змінну, то регістр 16-розрядний). Якщо команда процесора потребує для свого виконання більше одного операнда, то у якості одного з них обов'язково треба зазначити регістр процесора:

movebx,offset var1

Для позначення змінних використовуються мітки, що описують адресу розташування даних. Для позначення розміру даних можуть бути використані директиви DB (BYTE), DW (WORD), DD (DWORD), DQ (QWORD), DT (TBYTE). Самою останньою командою у програмі перед директивою end повинна бути команда ret. Ця команда здійснює передачу управління операційної системи.

Символом (;) позначають коментарі.

Після того, як код програми буде набраний, його необхідно зберегти у файлі з розширенням .asm (файл зберігається у теці masm32) та виконати інтерпретацію. У MASM32 інтерпретація коду відбувається через меню Debug (рис. 2). Потрібно обрати "Dbg Assemble & Link" у випадку роботи з консольним додатком або "Dbg ConsoleAssemble & Link" у випадку роботи з віконним додатком. Під час виконання цих команд відбувається створення об’єктного файлу .obj і файлу виконання (з розширенням .exe).

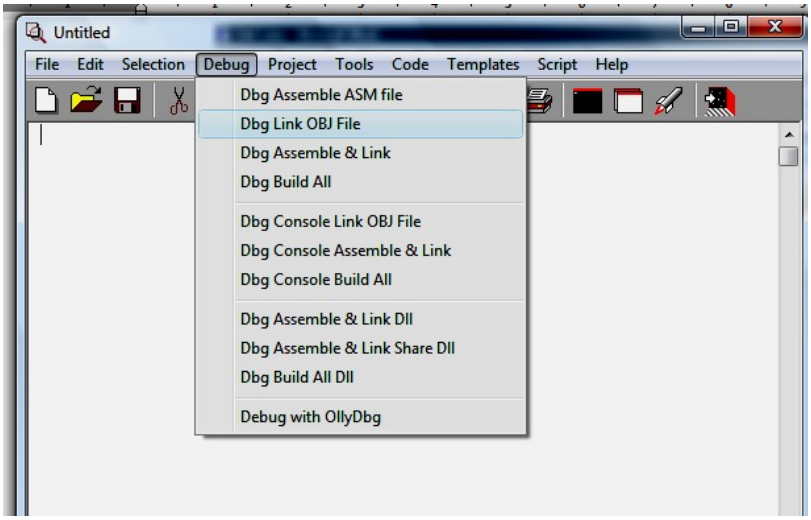

Рис. 2 Графічний інтерфейс середовища QEDITOR.

Інтерпретацію, також, можна робити за допомогою утиліт командного рядка (файловий менеджер FAR, процесор командного рядка cmd.exe або command.com). Інтерпретація за допомогою командного рядка відбувається у два етапи:

1. Створення об’єктного файлу:

c:\masm32\bin\ml /c /coff /Zd /Zi\шлях\ім'я_програми.asm

/c – асемблювання без лінкування;

/coff – генерувати об’єктний файл у форматі coff

/Zd – створювати інформацію про номери рядків для налагодження;

/Zi - створювати інформацію про символьні імена для налагодження.

У результаті буде створений файл за тим же ім'ям і розширенням .obj.

2. Створення файлу виконання:

\link /SUBSYSTEM:CONSOLE /DEBUG ім'я_обєктного_файлу

SUBSYSTEM:{...} – підсистема. Обирає операційну систему, у якій буде виконуватися програма:

NATIVE|WINDOWS|CONSOLE|WINDOWSCE|POSIX

DEBUG – розташовує інформацію для налагодження у файл виконання.

5.3 Контрольні завдання.

1. Запустити програму QEDITOR, набрати програмний код, зберегти його у файлі з розширенням .asm.

2. Отримати файл виконання. За необхідності відлагодити програму.

3. Використовуючи OllyDbg відкрити файл з розширенням .exe, виконати його та переглянути вміст регістрів, які були задіяні для написання програмного коду.

4. Перевірити результат, виконавши розрахунки вручну і порівнявши їх з отриманим у програмі. Скласти звіт.

За варіантом створити програму, яка:

Варіант 1

Написати програму, яка обчислює значення функції $f(x)=(x-a)\cdot(x-b)+d$, якщо $a=2, b=-3, d=5$, у точці $x=7$.

Варіант 2

Написати програму, яка обчислює значення функції $f(x)=(x+3a)\cdot(2x-b)+d$, якщо $a=4, b=-2, d=7$, у точці $x=3$.

Варіант 3

Написати програму, яка обчислює значення функції $f(x)=(x-b)\cdot(3x+ab)+d$, якщо $a=-2, b=3, d=5$, у точці $x=7$.

Варіант 4

Написати програму, яка обчислює значення функції $f(x)=(4x-b)\cdot(3y+a)+d$, якщо $a=12, b=3, d=7$, $x=7, y=10$ у точці $x=-4$.

Варіант 5

Написати програму, яка обчислює значення функції $f(x)=(2x^2+a)\cdot(x+b)+d$, якщо $a=-2, b=-7, d=-9$, у точці $x=7$.

Варіант 6

Написати програму, яка обчислює значення функції $f(x)=ax^2+by^2+d$, якщо $a=8, b=5, d=-6$, у точці $x=-7, y=8$.

Варіант 7

Написати програму, яка обчислює значення функції $f(x)=(x-a)\cdot(3y^2+bd)+d$, якщо $a=-4, b=-2, d=9$, у точці $x=7, y=10$.

Варіант 8

Написати програму, яка обчислює значення функції $f(x)=ax^2y+aby+d$, якщо $a=-3, b=5, d=-5$, у точці $x=6, y=7$.

Варіант 9

Написати програму, яка обчислює значення функції $f(x)=abx+by^2+ad$, якщо $a=8, b=-9, d=-6$, у точці $x=17, y=-5$.

Варіант 10

Написати програму, яка обчислює значення функції $f(x)=(ax-b)+by^2+d$, якщо $a=-12, b=4, d=-13$, у точці $x=17, y=-5$.

Варіант 11

Написати програму, яка обчислює значення функції $f(x)=ax^2+(by-x)^2+d$, якщо $a=2, b=-3, d=5$, у точці $x=2, y=-6$.

Варіант 12

Написати програму, яка обчислює значення функції $f(x)=(3ax-d)+by^2+ad$, якщо $a=-2, b=3, d=-5$, у точці $x=-3, y=10$.

Варіант 13

Написати програму, яка обчислює значення функції $f(x)=(ax+2y)^2+bd$, якщо $a=-4, b=-3, d=12$, у точці $x=-10, y=5$.

Варіант 14

Написати програму, яка обчислює значення функції $f(x)=x^3+aby+d$, якщо $a=8, b=12, d=-15$, у точці $x=7, y=-5$.

Варіант 15

Написати програму, яка обчислює значення функції $f(x)=(2ax^2+4)by+d$, якщо $a=-7, b=-13, d=-5$, у точці $x=-7, y=5$.

Варіант 16

Написати програму, яка обчислює значення функції $f(x)=(a-bc)+dy$, якщо $a=15, b=2, c=3, d=8$, у точці $x=-7, y=5$.

Варіант 17

Написати програму, яка обчислює значення функції $f(x)=(a^2-b)-(c+d)y$, якщо $a=8, b=11, c=3, d=8$, у точці $x=-7, y=5$.

Варіант 18

Написати програму, яка обчислює значення функції $f(x)=abc+d-y^2$, якщо $a=5, b=6, c=4, d=22$, у точці $x=3, y=7$.

Варіант 19

Написати програму, яка обчислює значення функції $f(x)=ab(c+d)-(y+8)$, якщо $a=5, b=6, c=4, d=22$, у точці $x=3, y=7$.

Варіант 20

Написати програму, яка обчислює значення функції $f(x)=9a-bc+d-y^2$, якщо $a=5, b=-6, c=4, d=-22$, у точці $x=3, y=7$.