ЛАБОРАТОРНА РОБОТА № 3.

Програмування задач лінійної структури. Обчислення виразів.

Мета: вивчити способи задання констант та змінних в Асемблері та набути навиків використання арифметичних команд над даними різного розміру.

Визначення даних в мові Assembler.

Асемблер допускає два основні методи визначення даних: через вказування довжини даних та за їх значенням. В першому випадку використовуються директиви визначення даних, а у другому безпосереднє значення або директива EQU.

Розглянемо основний формат визначення даних:

[ім'я] Dn вираз

- Ім'я елемента даних не є обов'язковим але якщо в програмі є посилання на деякий елемент, те це робиться за допомогою імені.
- Для визначення елементів даних є наступні директиви:

DB (байт)
DW (слово – 2 байти)
DD (подвійне слово – 4 байти),
DQ (8 байт)
DT (10 байт).

- Вираз може містити константу, наприклад: FLD1 DB 25,
- або знак питання для невизначеного значення, наприклад: FLDB DB ?

Вираз може містити кілька констант, розділених комами й обмеженими тільки довжиною рядка:

FLD3 DB 11, 12, 13, 14, 15, 16, ...

Асемблер визначає ці константи у вигляді послідовності сусідніх байт. Звертання за FLD3 вказує на першу константу 11, за FLD3+1 - на другу 12. (FLD3 можна представити як FLD3+0).

Наприклад команда MOV AL,FLD3+3 завантажує в регістр AL значення 14 (0Eh).

Вираз допускає також повторення константи в наступному форматі:

[ім'я] Dn кількість повторень DUP (вираз) ...

Символьні рядки

Символьний рядок використаються для опису даних, таких як, наприклад, імена людей або заголовки сторінок. Вміст рядка виділяється одинарними лапками, наприклад, 'PC' або подвійними лапками - "PC". Асемблер переводить символьні рядки в об'єктний код у звичайному форматі ASCII.

Символьний рядок визначається тільки директивою DB, в якій вказується більше двох символів у нормальній послідовності зліва направо. Отже, директива DB є єдиним можливим форматом для визначення символьних даних.

Числові константи

Числові константи використаються для арифметичних величин і для адрес пам'яті. Для опису константи лапки не ставляться. Асемблер перетворить всі числові константи в шістнадцятковий формат і записує байти в об'єктному коді в зворотної послідовності - справа наліво.

Десятковий формат. Десятковий формат допускає десяткові цифри від 0 до 9 і позначається останньою буквою D, що можна не вказувати, наприклад, 125 або 125D. Незважаючи на те, що асемблер дозволяє кодування в десятковому форматі, він перетворить ці значення в шістнадцятковий об'єктний код. Наприклад, десяткове число 125 перетвориться в. 7Dh.

Шістнадцятковий формат. Шістнадцятковий формат допускає шістнадцяткові цифри від 0 до F і позначається останньою буквою H. оскільки асемблер думає, що з букви починаються ідентифікатори, те першою цифрою шістнадцяткової константи повинна бути цифра від 0 до 9. Наприклад, 2EH або 0FFFH, які асемблер перетворить відповідно в 2E і FF0F (байти в другому прикладі записуються в об'єктний код у зворотній послідовності).

Двійковий формат. Двійковий формат допускає двійкові цифри 0 і 1 і позначається останньою буквою B. Двійковий формат звичайно використається для більше чіткого подання бітових значень у логічних командах AND, OR, XOR і TEST. Десяткове 12, шістнадцяткове C і двійкове 1100B усі генерують один і той же код:

*Вісімковий формат. В*ісімковий формат допускає вісімкові цифри від 0 до 7 і позначається останньою буквою Q або O, наприклад, 253Q.

Беззнакові й знакові дані.

Числові поля можуть трактуватися як знакові, або як беззнакові. Це контролюється при написанні програми **самостійно**. Для беззнакових величин всі біти є бітами даних Для знакових величин лівий біт є знаковим.

Директива визначення байта (DB)

Один байт виражається двома шістнадцятковими цифрами. Для беззнакових величин за допомогою цієї директиви можна представити числа від 0 до 255 (FFh). Для знакових даних найбільше додатне число в одному байті це 7F, всі "більші" числа від 80 до FF представляють від'ємні значення. У десятковому виразі ці межі визначаються числами +127 і -128.

Директива визначення слова (DW)

Директива DW визначає елементи, які мають довжину в одне слово (два байти). . Два байти представляються чотирма шістнадцятковими цифрами. Для беззнакових величин за допомогою цієї директиви можна представити числа від 0 до 65535 (FFh). Для знакових даних найбільше додатне шістнадцяткове число у двох байтах це 7FFF; всі "більші" числа від 8000 до FFFF представляють від'ємні значення. У десятковому форматі ці межі визначаються числами +32767 і -32768.

Символьний вираз в DW обмежений двома символами, які асемблер представляє в об'єктному коді так, що, наприклад, 'PC' стає 'CP'. Для визначення символьних рядків директива DW має обмежене застосування.

Директиви визначення DD, DQ та **DT** є аналогічними до DW з відповідними довжинами даних.

Безпосередні операнди

Команда MOV AX,0123H пересилає безпосередню шістнадцяткову константу 0123 у регістр AX. Трибайтний об'єктний код для цієї команди є B82301, де B8 означає "переслати безпосереднє значення в регістр AX", а наступні два байти містять саме значення. Багато команд мають два операнди: перший може бути регістр або адреса пам'яті, а другий - безпосередня константа.

Використання безпосереднього операнду більш ефективне ніж визначення числової константи в сегменті даних і організація посилання на неї в операнді команди MOV

Довжина безпосередньої константи залежить від довжини першого операнду. Наприклад, якщо безпосередній операнд є двобайтовим, а регістр AL має тільки один байт: MOV AL,0123H, то асемблер повідомить про помилку, однак, якщо безпосередній операнд коротший, ніж операнд, що одержує, наприклад: ADD AX,25H (немає помилки) то асемблер розширює безпосередній операнд до двох байт, 0025 і записує об'єктний код у вигляді 2500.

Формати безпосередніх даних Безпосередня константа може бути шістнадцятковою, наприклад, 0123h; десятковою, наприклад, 291 (яку асемблер перетворить в 0123h); або двійковою, наприклад, 100100011B (яка перетвориться в 0123h).

Список команд, які допускають використання безпосередніх операндів:

Команди пересилання й порівняння:	MOV, CMP.
Арифметичні команди:	ADC, ADD, SBB, SUB.
Команди зсувів:	RCL, RCR, ROL, ROR, SHL, SAR, SHR.
Логічні команди:	AND, OR, TEST, XOR.

Директива EQU

Директива EQU не визначає елемент даних, але визначає значення, що може бути використане для підстановки в інших командах. Припустимо, що в сегменті даних закодовано наступну директиву EQU:

TIMES EQU 10

Ім'я, у цьому випадку TIMES, може бути представлено будь-яким припустимим в асемблері ім'ям. Тепер, у якій би команді або директиві не використалося слово TIMES асемблер підставить значення 10. Наприклад, асемблер перетворить директиву

FIELDA DB TIMES DUP (?)

в директиву

FIELDA DB 10 DUP (?)

Ім’я, пов’язане з деяким значенням за допомогою директиви EQU, може використатися в командах, наприклад:

```
COUNTR EQU 05
       .....
MOV CX,COUNTR
```

Асемблер заміняє ім’я COUNTR у команді MOV на значення 05, створюючи операнд з безпосереднім значенням, так, ніби було закодовано MOV CX,05 ;Асемблер підставляє 05.

Перевага директиви EQU полягає в тому, що багато команд можуть використати значення, визначене іменем COUNTR. Якщо це значення повинне бути змінене, то зміні підлягає лише одна директива EQU. Природно, що використання директиви EQU доцільне лише там, де підстановка має сенс для асемблера. У директиві EQU можна використати символічні імена:

```
1. TP    EQU    TOTALPAY
2. MPY   EQU    MUL
```

Перший приклад припускає, що в сегменті дані програми визначене ім’я TOTALPAY. Для будь-якої команди, що містить операнд TP, асемблер замінить його на адресу TOTALPAY. Другий приклад показує можливість використання в програмі слова MPY замість звичайного мнемокоду MUL.

Арифметичні операції в Асемблері

Для виконання даної роботи слід розглянути наступні команди арифметики: ADD ADC INC SUB SBB DEC CMP NEG MUL DIV

Команди **ADD і SUB** виконують додавання й віднімання байтів або слів, що містять двійкові дані. Віднімання виконується у комп’ютері по методу додавання із двійковим доповненням: для другого операнду встановлюються зворотні значення бітів і додається 1, а потім відбувається додавання з першим операндом. У всьому, крім першого кроку, операції додавання й віднімання ідентичні. Можливі п’ять ситуацій:

ADD / SUB регістр-регістр;
ADD / SUB пам'ять-регістр;
ADD / SUB регістр-пам'ять;
ADD / SUB регістр-безпосереднє значення;
ADD / SUB пам’ять-безпосереднє значення.

Переповнення. При виконанні арифметичних операцій можливе переповнення. Один байт містить знаковий біт і сім біт даних, тобто значення від -128 до +127. Результат арифметичної операції може легко перевищити розрядність однобайтового регістра. Наприклад, результат додавання в регістрі AL, що перевищує його розрядність, автоматично не переходить у регістр AH. Припустимо, що регістр AL містить 60h, тоді результат команди ADD AL,20h генерує в AL суму -.80h. Але операція також встановлює прапорець переповнення і прапорець знаку у стан "негативно". Причина в тому, що.80h або двійкове 1000 0000 є від’ємним числом. Таким чином, в результаті, замість +128, ми одержимо -128. Оскільки регістр AL занадто малий для такої операції то варто скористатися регістром AX. Для розширення AL до AX можна скористатися командою CBW (Convert Byte to Word - перетворити байт у слово).

```
CBW
ADD AX,20H
```

Але повне слово має також обмеження. Для розширення AX до EAX можна скористатися командою CWDE.

Для беззнакових величин всі біти є бітами даних і замість обмеження +32767 регістр може містити числа до +65535. Для знакових величин лівий біт є знаковим. Команди ADD і SUB не роблять різниці між знаковими і беззнаковими величинами, вони просто додають і віднімають біти.

Наприклад, при додаванні двох двійкових чисел, одне з яких містить одиничний лівий біт можливе такі випадки. Для беззнакового числа біти представляють додатне число 249, для знакового – від’ємна число -7:

	Беззнакове	Знакове
11111001	249	-7
00000010	2	+2
11111011	251	-5

Двійкове представлення результату додавання однакове для беззнакового й знакового числа. Однак, біти представляють +251 для беззнакового числа й -5 для знакового. Таким чином, числовий вміст поля може інтерпретуватися по різному. Стан "перенос" виникає в тому випадку, коли є перенос у знаковий розряд. Стан "переповнення" виникає у тому випадку, коли перенос у знаковий розряд не створює переносу з розрядної сітки або перенос із розрядної сітки відбувається без переносу в знаковий розряд. При виникненні переносу при додаванні беззнакових чисел, результат одержується неправильний:

	Беззнакове	Знакове	CF	OF
11111100	252	-4		
00000101	5	+5		
00000001	1	1	1	0

(неправильно)

При виникненні переповнення при додаванні знакових чисел, результат виходить неправильний:

	Беззнакове	Знакове	CF	OF
01111001	121	+121		
00001011	11	+11		
10000100	132	-124	0	1

(неправильно)

При операціях додавання й віднімання може одночасно виникнути й переповнення, і перенос:

	Без знакове	Знакове	CF	OF
11110110	246	-10		
10001001	137	-119		
01111111	127	+127	1	1

(неправильно) (неправильно)

Таким чином, перенос, що може виникати при додаванні найстарших слів, слід трактувати для беззнакових даних як значущий біт і враховувати його в загальній сумі, а для знакових, вважати ознакою від'ємного числа.

Команда **ADC** виконує додавання з врахуванням переносу. Тобто якщо прапорець CF має значення 1, то ця команда додає 1 до першого операнду, а вже потім додає другий операнд.

Команда **SBB** виконує віднімання з запозиченням. Тобто якщо прапорець CF має значення 1, то ця команда спочатку віднімає 1 від першого операнду, а вже потім віднімає від нього другий операнд.

Команда **INC/DEC** додає / віднімає одиницю до байту або слова в регістрі або пам'яті.

Команда **CMP** виконує операцію порівняння двох полів даних. Фактично, ця команда виконує віднімання другого операнду від першого, але значення операндів на змінює. Операнди повинні бути однакової довжини: байт або слово. Команда може мати наступний формат:

CMP регістр-регістр;
CMP пам'ять-регістр;
CMP безпосереднє значення -регістр;
CMP регістр-пам'ять;
CMP безпосереднє значення –пам'ять

Команда **NEG** змінює війкове значення з додатного на від'ємне і з від'ємного на додатне. Фактично, вона віднімає операнд від нуля і додає одиницю.

Множення

Операція множення для беззнакових даних виконується командою **MUL**, а для знакових - **IMUL** (Integer MULtiplication - множення цілих чисел). У єдиному операнді команд MUL і IMUL вказується множник. Відповідальність за контроль над форматом чисел, що обробляються і за вибір відповідної команди множення лежить на самому програмісті. Існують такі варіанти операції множення:

"Байт на байт". Множене перебуває в регістрі AL, а множник у байті пам'яті або в однобайтовом регістрі. Після множення добуток перебуває в регістрі AX. Операція ігнорує і витирає будь-які дані, які були в регістрі AH.

AX=AL*регістр$_8$/пам'ять$_8$

"Слово на слово". Множене перебуває в регістрі AX, а множник - у слові пам'яті або у двобайтному регістрі. Після множення добуток знаходиться в подвійному слові, для якого потрібно два регістри: старша (ліва) частина добутку - в регістрі DX, а молодша (права) частина - в регістрі AX. Операція ігнорує й стирає будь-які дані, які перебували в регістрі DX.

DX:AX=AX* регістр$_{16}$/пам'ять$_{16}$

"Подвійне слово на подвійне слово". Множене перебуває в регістрі EAX, а множник - у слові пам'яті або у чотирьохбайтному регістрі. Після множення добуток знаходиться в восьмибайтному слові, для якого потрібно два регістри: старша (ліва) частина добутку - в регістрі EDX, а молодша (права) частина - в регістрі EAX. Операція ігнорує й стирає будь-які дані, які перебували в регістрі EDX.

EDX:EAX=EAX* регістр$_{32}$/пам'ять$_{32}$

Розглянемо наступну команду:

MUL MULTR

Якщо поле MULTR визначене як байт (DB), то операція припускає множення вмісту AL на значення байта з поля MULTR. Якщо поле MULTR визначене як слово (DW), то операція припускає множення вмісту AX на значення слова з поля MULTR. Якщо множник перебуває в регістрі, то довжина регістра визначає тип операції, як це показано нижче:

MUL CL ; Байт-множник: множене в AL, добуток в AX.

MUL BX ; Слово-множник: множене в AX, добуток DX:AX.

Команда IMUL призначена для множення чисел зі знаком. Формат її використання аналогічний MUL.

Якщо множник і множене мають однаковий знак, то IMUL та MUL генерують однаковий результат, а якщо множники мають різні знаки, то команда MUL виробляє доданій результат, а команда IMUL – від'ємний.

Для підвищення ефективності операції множення, при множенні на степінь числа 2 більш доцільним є зсув вліво на необхідну кількість бітів. Зсув більше ніж на один розряд потребує параметру у регістрі CL. Наприклад:

Множення на 2	SHL al,1
Множення на 8:	MOV CL,3
	SHL AX,CL

Ділення.

Операція ділення для беззнакових даних виконується командою **DIV**, а для знакових - **IDIV.** Відповідальність за вибір відповідної команди лежить на програмісті. Існують дві основні операції ділення:

"Слово на байт". Ділене перебуває в регістрі AX, а дільник - у байті пам'яті або а однобайтовому регістрі. Після ділення остача виходить у регістрі AH, а частка - в AL. Оскільки однобайтова частка дуже мала (максимально +255 (.FFh) для беззнакового ділення і +127 (7Fh) для знакового), то дана операція має обмежене використання.

AX / регістр$_8$ = AL(частка) AH(остача)

AX / пам'ять$_8$ = AL(частка) AH(остача)

"Подвійне слова на слово". Ділене перебуває в регістровій парі DX:AX, а дільник - у слові пам'яті або в регістрі. Після ділення остача виходить у регістрі DX, а частка в регістрі AX. Частка в одному слові допускає максимальне значення +32767 (FFFFh) для беззнакового ділення і +16383 (7FFFh) для знакового.

DX(старша частина) **AX**(молодша частина) **/ регістр$_{16}$ = AX(частка) DX(остача)**

DX(старша частина) **AX**(молодша частина) **/ пам'ять$_{16}$ = AX(частка) DX(остача)**

"Восьмибайтне слова на подвійне слово". Ділене перебуває в регістровій парі EDX:EAX, а дільник - у подвійному слові пам'яті або в регістрі. Після ділення остача виходить у регістрі EDX, а частка в регістрі EAX.

EDX(старша частина) **EAX**(молодша частина) **/ регістр32 = EAX(частка) EDX(остача)**
EDX(старша частина) **EAX**(молодша частина) **/ пам'ять32 = EAX(частка) EDX(остача)**

У єдиному операнді команд DIV і IDIV вказується дільник. Розглянемо наступну команду:

DIV DIVISOR

Якщо поле DIVISOR визначене як байт (DB), то операція припускає ділення слова на байт. Якщо поле DIVISOR визначене як слово (DW), те операція припускає ділення подвійного слова на слово. При діленні, наприклад, 13 на 3, виходить результат 4 1/3. Частка є 4, а остача - 1.

Команда IDIV (Integer DIVide) виконує ділення знакових чисел. Таким чином, якщо ділене й дільник мають однаковий знаковий біт, то команди DIV і IDIV генерують однаковий результат . Але, якщо ділене й дільник мають різні знакові біти, то команда DIV генерує додатну частку, а команда IDIV – від'ємну частку.

Зауважимо, що для підвищення продуктивності, при діленні на степінь числа 2 (2, 4, і т. д.) варто використовувати зсув вправо на відповідну кількість війкових розрядів.

Переповнення й переривання

Використовуючи команди DIV і особливо IDIV, дуже легко викликати переповнення Переривання до непередбачених результатів. В операціях ділення передбачається,, що дільник значно менший, ніж ділене. Ділення на нуль завжди викликає переривання. Але ділення на 1 генерує частку, що дорівнює діленому, що може також легко викликати переривання. Рекомендується використати наступне правило:

якщо дільник - байт, то його значення повинне бути більше, ніж лівий байт (AH) діленого; якщо дільник - слово, то його значення повинне бути більше, ніж ліве слово (DX) діленого.

Проілюструємо дане правило для дільника, рівного 1:

Операція ділення:	Ділене	Дільник	Частка
Слово на байт:	0123	01	(1)23
Подвійне слово на слово:	4026 0001	0001	(1)4026

В обох випадках частка перевищує можливий розмір. Для того щоб уникнути подібних ситуацій, корисно вставляти перед командами DIV і IDIV відповідну перевірку (CMP). Для команди IDIV дана логіка повинна враховувати той факт, що або ділене, або дільник можуть бути від'ємними, а порівнюються абсолютні значення, тому необхідно використати команду NEG для тимчасового перетворення від'ємного значення в додатне.

Ділення відніманням. Якщо частка занадто велика, то ділення можна виконати з допомогою циклічного віднімання. Метод полягає в тому, що дільник віднімається з діленого й у цьому ж циклі частка збільшується на 1. Віднімання триває, поки ділене залишається більше ніж дільник. При такому методі, дуже велика частка й малий дільник можуть викликати тисячі циклів.

```
SUB CX,CX          ;Очищення частки
C20:  CMP AX,BX    ;Якщо ділене < дільника,
JB C30             ; то вийти
SUB AX,BX          ;Віднімання дільника від діленого
INC CX             ;Інкремент частки
JMP C20            ;Повторити цикл
C30:  RET          ;Частка в CX, остача в AX
```

Наприкінці підпрограми регістр CX буде містити частку, а AX – остачу від ділення.

Якщо частка отримується в регістровій парі DX:AX, те необхідно зробити два доповнення:

1. У мітці C20 порівнювати AX і BX тільки при нульовому DX.
2. Після команди SUB вставити команду SBB DX,00.

Контрольні запитання:

1. Як визначаються в Асемблері дані довжиною в 2 байти? Як вони записуються у пам’ять?
2. Чим відрізняються директива EQU від DW ?
3. Які арифметичні команди є в мові Асемблер?
4. Пояснити виникнення переповнення і переносу при виконанні арифметичних команд.
5. Які формати операцій множення і ділення ви знаєте?
6. Що робить команда CMP ?

Література:

1.Р.Джордейн.Справочник програмиста персональных компъютеров типа IBM PC XT и AT. -М."Финансы и статистика",1992,стор.13-31.

2.Л.О.Березко,В.В.Троценко. Особливості програмування в турбо-асемблері. -Київ,НМК ВО,1992.

3.Л.Дао. Программирование микропроцессора 8088.Пер.с англ.-М."Мир",1988.

4.П.Абель.Язык ассемблера для IBM PC и программирования. Пер. з англ.-М.,"Высшая школа",1992.

ЗАВДАННЯ

1. Створити *.exe програму, яка реалізовує обчислення, заданого варіантом виразу і зберігає результат в пам'яті. Вхідні операнди A, B, C, D, E, F вважати знаковими і довжиною в байтах, згідно з індексу; K – константа, довжина якої визначається значенням(згідно варіанту). Для її опису слід використати директиву EQU.
2. За допомогою Debug, відслідкувати правильність виконання програми (продемонструвати результати проміжних та кінцевих обчислень).
3. Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї.
4. Дати відповідь на контрольні запитання.

ВАРІАНТИ ЗАВДАННЯ:

A, B, C, D, E, F - знакові операнди, довжиною в байтах, згідно з індексу, значення K подано у 16-му форматі.

№	Вираз	K
1	$X=A_2+B_2*C_1-D_2/E_1+K$	1254021
2	$X=A_4/B_2+C_3-D_1*E_1-K$	202
3	$X=K-B_2+C_2/D_1-E_1*F_2$	37788663
4	$X=A_1*(B_2+C_1)-D_4/E_2+K$	45694
5	$X=A_2*B_2-A_2*C_1-D_4/E_2+K$	505
6	$X=K+B_2/C_1-D_2*F_2-E_1$	6DD02316
7	$X=A_4/B_2-C_1*(D_1+E_2-K)$	717
8	$X=A_4-B_2+K-D_2/E_1+F_1*B_2$	88
9	$X=A_1*B_2-A_1*C_2+D_2/(E_1+K)$	29
10	$X=A_4-B_4/C_2+K+E_2*F_1$	2310
11	$X=(A_4-B_3-K)*D_1+E_4/F_2$	311
12	$X=K+B_4/C_2-D_2*F_2-E_1$	7055E0AC
13	$X=A_2/B_1+C_1*(D_1+E_2-K)$	2513
14	$X=A_4-B_1-K-D_2/E_1+F_1*B_1$	614
15	$X=A_3+(B_1*C_2)-D_4/E_2+K$	4569600F
16	$X=A_4/B_2+C_2-D_1*E_1+K$	616
17	$X=A_4-K+C_4/D_2-E_1*F_1$	1017
18	$X=A_1*(B_2-C_1)+D_2/E_1+K$	56987018
19	$X=A_2*B_2+A_2*C_1-D_2/E_1+K$	4019
20	$X=K+B_4/C_2-D_1*F_1-E_2$	18932020
21	$X=A_4/B_2+C_1*(D_2-E_1+K)$	21
22	$X=K-B_1-C_1-D_2/E_1+F_2*B_1$	45781022
23	$X=A_2*B_2-C_1+D_4/E_2+K$	7AA02023
24	$X=K-B_2/C_1+D_3+E_2*F_2$	74569024
25	$X=(K-B_2-C_1)*D_1+E_4/F_2$	2B05025
26	$X=A_2+K+C_2/D_1-E_1*F_1$	6C26
27	$X=A_2*B_1+C_4/(K-E_1*F_1)$	A77627
28	$X=K+B_4/C_2+D_3-E_2/F_1$	3FF28
29	$X=K-B_1*C_1+D_2-F_2/E_1$	12A0C029
30	$X=K+B_3-D_2/C_1+E_1*F_2$	25630

Приклад виконання.

Завдання: Написати програму, яка обчислює арифметичний вираз і результат записує в пам'ять.

Заданий вираз:

$$X=K-B_1+C_4/D_2-E_2*F_1 \quad K=56982h$$

Для обчислення заданого виразу слід виконати такі дії:

1. **K-B** (від трьох (чотирьох) байт відняти один байт – чотири байти);
2. **C/D** (чотири байти поділити на два байти, результат – два байти);
3. **E*F** (два байти помножити на один (два) байт, результат – чотири байти);
4. проміжний (1)+проміжний (2) (чотири байти додати два (чотири) байти, результат – чотири байти);
5. проміжний (4)- проміжний (3) (чотири байти відняти чотири байти, результат – чотири байти).

Таким чином, в при описі даних слід відвести місце не лише під вхідні дані та константу, а й під проміжні та остаточні результати, відповідних розмірностей.

Отже, секція опису даних може бути описана таким чином:

```
.data
K       EQU   56982h            ;354690
B       db    56h          ;86
Cc      dd    342376h      ;3416950
D       dw    6745h        ;26437
E       dw    2561h        ;9569
F       db    89h          ;-119
Temp1   dd    0h
Temp2   dw    0h
Temp3   dd    0h
X       dd    ?
```

Всі невідомі значення заповнюємо нулями, щоб уникнути непередбачуваних результатів.

Далі, переходимо до безпосереднього виконання арифметичних обчислень, тобто до опису секції коду.

Обчислення виконуємо за допомогою операцій DIV, MUL, ADD, SUB згідно з їх форматами та розмірністю даних.

Повний текст програми, що виконує дані обчислення приведений нижче. Остаточний результат обчислення виразу буде знаходитися в пам'яті за адресою **X.**

```
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib

.data

K       EQU   56982h        ;354690
B       db    56h          ;86
Cc      dd    342376h      ;3416950
D       dw    6745h        ;26437
E       dw    2561h        ;9569
F       db    89h          ;-119
Temp1   dd    0h
Temp2   dw    0h
```

```
Temp3   dd     0h
X dd    ?

Message db 'X = K - B + C / D - E * F =            ', 13, 10
NumberOfCharsToWrite dd $-Message
format db '%d', 0
hConsoleOutput dd 0
NumberOfCharsWritten dd 0

.code
start:
;X = K - B + C / D - E * F
;Temp1 = K - B
mov ebx, K
mov al, B
cbw
cwde
sub ebx, eax
mov Temp1, ebx
;Temp2 = C / D
mov eax, Cc
shr eax, 16
mov dx, ax
mov eax, Cc
idiv D
mov Temp2, ax
;Temp3 = E * F
mov al, F
cbw
imul E
mov bx, ax
mov ax, dx
shl eax, 16
mov ax, bx
mov Temp3, eax
;X = Temp1 + Temp2 - Temp3
mov ax, Temp2
cwde
mov ebx, Temp1
add eax, ebx
mov ebx, Temp3
sub eax, ebx
mov X, eax

push X
push offset format
push offset [Message+28]
call wsprintfA

push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push NumberOfCharsToWrite
push offset Message
push hConsoleOutput
call WriteConsoleA
push 0
call ExitProcess
end start
```