**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіт до комп’ютерного практикума робіт з дисципліни

Системне програмування

*Виконав:*

студент групи ЗПІ-ЗП 61

Макіян Смбат Артурович

*Перевірив:*

Викладач

доцент, к.т.н.

Лісовиченко Олег Іванович

Київ – 2018

**Звіт з практикума №1**

**Створення програм на асемблері.**

Завдання:

1. Для програми, наведеної вище, створити файл типу .asm. Ця програма не має засобів виводу даних, тому правильність її виконання треба перевірити за допомогою td.exe.
2. Скомпілювати програму, включивши потрібні опції для налагоджувача та створення файла лістингу типу .lst.
3. Ознайомитись зі структурою файла .lst. За вказівкою викладача, для певної команди асемблера розглянути структуру машинної команди і навести її у звіті.
4. Скомпонувати .obj-файл програми. Включити опції для налагодження та створення .map-файлу.
5. Занести до звіту адреси початку та кінця всіх сегментів з .map-файлу.
6. Завантажити до налагоджувача td.exe одержаний .exe-файл програми.
7. У вікні CPU у полі DUMP знайти початкову адресу сегмента даних та записати його до звіту. Знайти масиви SOURCE та DEST. Дані у масиві SOURCE подаються у шістнадцятковій системі.
8. У покроковому режимі за допомогою клавіші F7 виконати програму. Одержані результати у масиві DEST показати викладачеві.

Текст

STSEG SEGMENT PARA STACK "STACK"

DB 64 DUP ("STACK")

STSEG ENDS

DSEG SEGMENT PARA PUBLIC "DATA"

SOURCE DB 10, 20, 30, 40

DEST DB 4 DUP ("?")

DSEG ENDS

CSEG SEGMENT PARA PUBLIC "CODE"

MAIN PROC **FAR**

ASSUME CS: CSEG, DS: DSEG, SS: STSEG

**push** DS

**xor** AX, AX

**push** AX

**mov** AX, DSEG

**mov** DS, AX

**mov** DEST,0

**mov** DEST+1, 0

**mov** DEST+2, 0

**mov** DEST+3, 0

**mov** AL, SOURCE

**mov** DEST+3, AL

**mov** AL, SOURCE+1

**mov** DEST+2, AL

**mov** AL, SOURCE+2

**mov** DEST+1, AL

**mov** AL, SOURCE+3

**mov** DEST, AL

**ret**

MAIN ENDP

CSEG ENDS

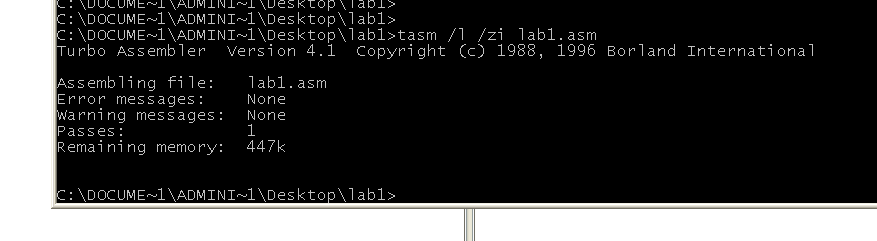
END MAIN

**Виконання:**

ПЗ, що використовуються для виконання даної лабораторної роботи адаптоване для використання в середовищі MS DOS, тому, було встановлено ОС Windows XP Mode 32 bit під VirtualBox.

Для створення .asm файлу використовуються редактор Sublime Text 3

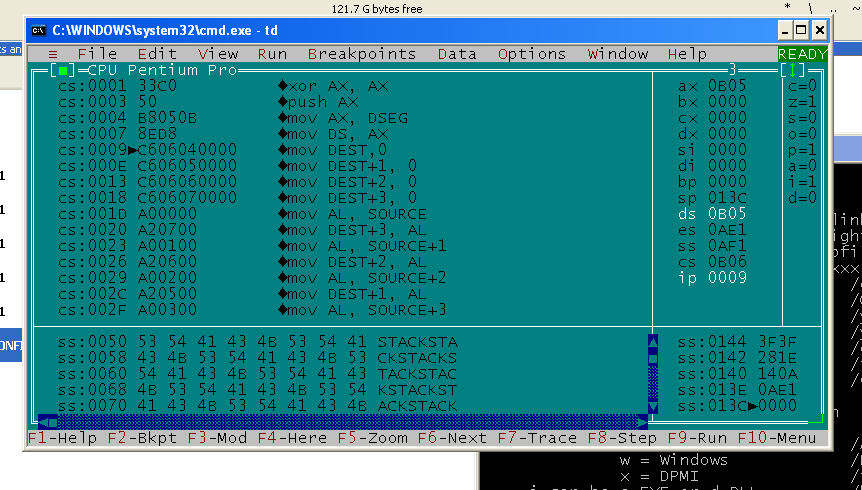
Для перевірки з використанням td.exe необхідно отримати скомпільовану виконувану програму в «.EXE», для цього необхідно створити файл типу «.OBJ» за допомогою TASM.EXE а потім з файлу «.OBJ» отримати «.EXE», за допомогою TLINK.EXE.



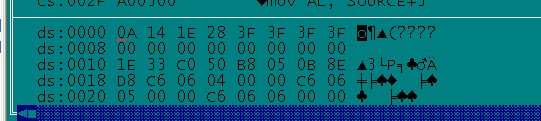
З контекстного меню відкриваємо виконуваний файл «.EXE», в даному випадку LAB1.EXE

Покроково виконавши дебаг програми за допомогою клавіши F7, можна зазначити зміни, що виникають у сегменті стеку SS;, куди записуються початкові значення регістрів DS та AX та зміни у сегменті даних DS: де вказано зміщення (у стеку даних) та значення у шістнадцятковому форматі (DS:0000 = 0A, DS:0001 = 14, DS:0002 = 1E, DS:0003 = 28). Що у свою чергу відповідає 10, 20, 30, 40 у десятковому форматі. Т.к. данні зі стеку можно отримати лиже з кінця, но останнє число (40) буде першим, тобто масив чисел будем у зворотному порядку.

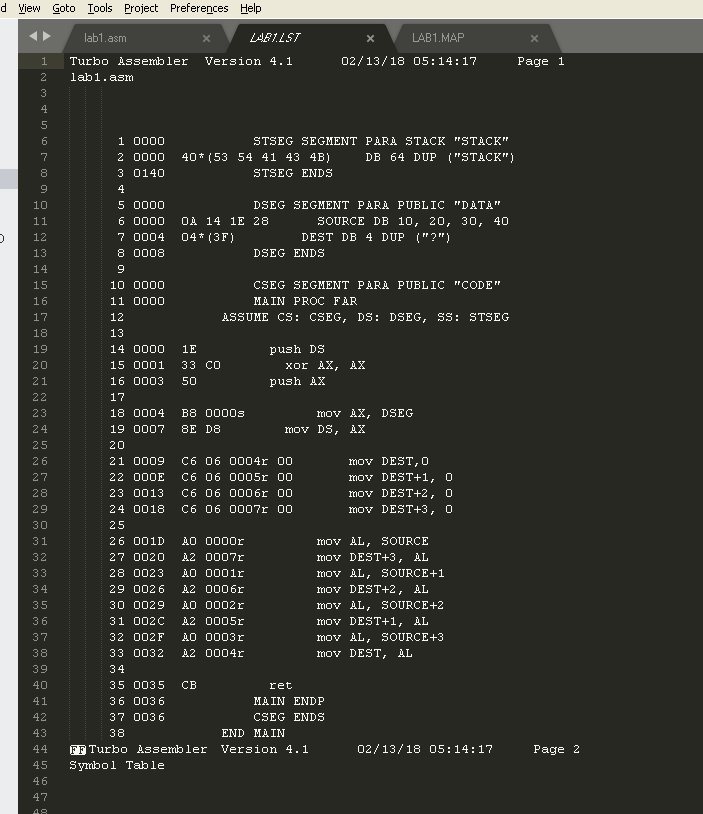
Для створення лістингу типу .lst. програми, необхідно додати параметр /l при створенні файлу «.OBJ».

З контекстного меню відкриваємо виконуваний файл «.EXE», в даному випадку LAB1.EXE

Покроково виконавши дебаг програми за допомогою клавіши F7, можна зазначити зміни, що виникають у сегменті стеку SS;, куди записуються початкові значення регістрів DS та AX та зміни у сегменті даних DS: де вказано зміщення (у стеку даних) та значення у шістнадцятковому форматі (DS:0000 = 0A, DS:0001 = 14, DS:0002 = 1E, DS:0003 = 28). Що у свою чергу відповідає 10, 20, 30, 40 у десятковому форматі. Т.к. данні зі стеку можно отримати лиже з кінця, но останнє число (40) буде першим, тобто масив чисел будем у зворотному порядку.



Переглянемо вміст файлу LAB1.LST:

З малюнку видно, що «.LST» файл містить інформацію про версію Турбо Ассемблера, дату, час створення, позначку сторінки, назву вихідного файлу.

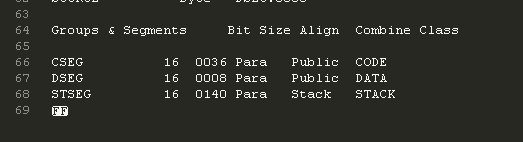
Далі міститься таблиця стовпців «вкладення», «номер\_строки», «зміщення», «машинний код», «вихідний\_код».

«вкладення» в данному випадку пусте, бо немає вкладених файлів, тощо.

«зміщення» вказує на зміщення поточного сегменту відповідно від початкової строки генерованої Турбо Ассемблером.

«машинний код» вказує фактичну послідовність шістнадцяткових значень розміром в байт або слово, які отримані із відповідної строки «вихідний\_код».

На другій сторінці містяться данні про групи та сегменти з інформацією про кількість біт, розмірність, вирівнювання, атрибут та клас Мінімальна кількість сегментів - 3.



«.MAP» містить інформацію про адреси початку та кінця всіх сегментів а також про розмір сегментів, що співпадає з даними із «.LST» файлу:

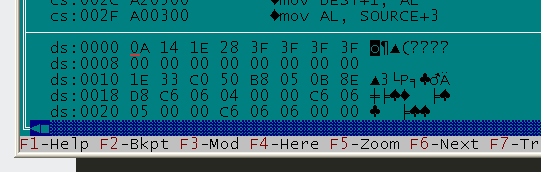


Початкова адреса сегмента даних DS:0000:

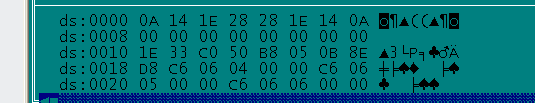
Вміст масивів SOURCE та DEST можна переглянути за допомогою сегмента даних:

Виходячи з малюнка, можна зробити висновок, що вхідний масив SOURCE був перенесений до масиву DEST у оберненому вигляді.

Було



Стало



**Висновок:**

Програма вирішує поставлене завдання. Теоретичні розрахунки відповідають отриманим. Програма працює правильно.

**Звіт з комп’ютерного практикуму №2**

**Засоби обміну даними.**

Завдання:

1. Скласти процедуру введення і перетворення цілого числа.

2. Скласти і реалізувати програму введення та виведення цілого числа зі знаком та виведення рядка символів.

3. Введення та виведення цілого числа з запрошенням до користувача.

Виконання:

Введення та оброблення числа.

Реалізуємо процедуру введення цілого числа. Для цього використаємо функцію 0Ah переривання DOS 21h, створимо окрему процедуру *User\_input proc:*



Відповідно довідки, функція реалізує ввід строки в буфер. Необхідно створити буфер *Buff\_num* (в сегменті даних) для зберігання числа:

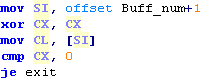


Буфер містить інформацію про максимальну кількість елементів (5 - 1 = 4), заповнений нулями та у кінці містить символ *'$'* для виведення строки у подальшому за допомогою функції 09h переривання DOS 21h.

Ми створили буфер з максимальною ємністю 4 цифри (або знаки).

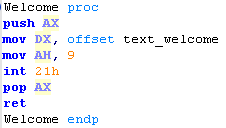
Для початку виконаємо перевірку, чи було занесено щось до буферу, для цього проаналізуємо інформацію про вміст буферу. Функція 0Ah створює буфер наступного вигляду: , де перший елемент (05) вказує на максимальну кількість символів плюс один, другий елемент - на кількість символів які містяться у буфері (02). Елементи самого буфера знаходяться після зміщення +2 та далі (+3, +4).

Тобто ми маємо перевірити чи є другий елемент буферу відмінним від нуля:



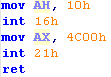
*Offset -* передає вказівник на другий елемент буфера. Якщо є відповідність  викликається функція виходу із помилкою ().

Для зручності користування створено декілька типових функцій виведення інформації на екран: *Massage, Welcome, Error\_Exit,* які використовують функцію 09h для виведення повідомлення на екран. Функція передає у регістр *DX* вказівник на повідомлення (о*ffset)*, та реалізує резервування та відновлення з стеку вмісту регістру *AH:*

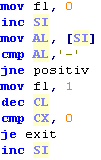




Додатково функція *Error\_Exit* виконує закриття програми після натискання клавіші*:*



Реалізуємо функцію перевірки на від’ємне число:

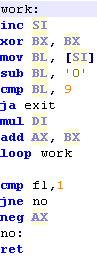


Обнуляємо флаг fl, зміщуємо вказівник на наступний елемент буферу, який містить першу цифру з введеного числа (змішення +2). Порівнюємо чи є цей елемент знаком «-». Якщо ми отримали позитивний результат порівняння (- присутній), ZH буде містити одиницю, тож, якщо флаг ZH = 0, продовжуємо виконання з функції *«positive»* для невід’ємного числа. У випадку, якщо першим є знак «-», заносимо 1 до змінної флага fl. Перевіряємо чи є інші символи окрім мінуса (), якщо інші відсутні - викликаємо вихід з помилкою. Переходимо до другого розряду числа (). Якщо число є позитивним проводимо подальшу обробку:



Обнуляємо регістри, поміщаємо 10 в *DI* для переведення у 10-кратність. Повертаємо адресу першого числа *(SI).*

Починаємо обробляти числа:

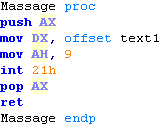


Беремо вказівник на перше число. Передаємо у регістр *BL* саме число, обробляємо його до вигляду десяткового (віднімаємо символ «0», що відповідає 30h). Порівнюємо число у регістрі *BL*, якщо там буде символ а не число, тобто елемент вище числа 9, спрацює перехід *ja* та програма завершиться з помилкою. Якщо це число, помножимо його на 10 та складаємо до *AX* і так до повного звільненні буферу (до досягнення останньої цифри числа записаного у кількості елементів буферу), в результаті ми отримаємо оброблене число в *AX*.

Перевіряємо присутність флага на від’ємне число, якщо флаг присутній, викликаємо команду *neg*, що змінює знак числа по протилежний, тобто вихідне число буде від’ємне.

Процедура повідомлення.

Ми закінчили введення, перевірку та обробку числа, повідомимо користувачу, що результат успішний, створимо процедуру *Massage* *proc:*



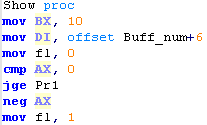
Для виведення інформації використаємо функцію 09h переривання DOS 21h. До регістру DX поміщаємо вказівник на необхідний текст повідомлення, та викликаємо перевивання.



Додатково у стек зберігаємо вміст регістру AX та відновлюймо його зі стеку після виведення необхідного повідомлення.

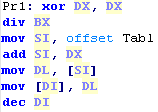
Процедура виведення числа:

Розробимо процедуру виведення числа *Show proc:*



Записуємо систему числення на яку будемо ділити наше число до BX () та переходимо до кінця буферу (). Обнуляємо флаг від’ємного числа та перевіряємо чи є записане число від’ємним. Якщо число від’ємне, змінюємо знак числа на протилежний () та записуємо флаг *fl.*

Якщо число є додатнім, переходимо до кроку *Pr1:*





При діленні у кроці  інструкція буде виконувати ділення як для змінної типу «слово». В такому випадку буде виконуватися ділення пари регістрів *АХ* та *DX*. Для початку обнуляємо регістр *DX,*  буде виконуватись ділення виключно числа у регістрі *АХ.*

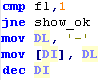
При діленні регістр *АХ* буде розділено на *ВХ* а залишок від ділення заноситься до регістру *DX.* Тобточисло у регістра *АХ* буде зменшувати свій порядок а залишок (одна цифра) має зберігатися окремо для подальшого виведення числа.

Далі передаємо адресу початку таблиці з символами, додаємо отримане зміщення, що відповідає залишку від ділення (який міститься в *DX)* таотримаємо, як результат сам символ номера з таблиці  у регістр *SI*. Передаємо символ до регістру DL () та записуємо його на «комірку» яку обробляли у регістрі DI (), переходимо до наступного символу у DI ()

Після оброблення символу ми маємо перевірити чи не закінчилось число, якщо результат негативний - перейти в початок кроку *Pr1* та виконати обробку другого, з кінця числа, номеру.



Якщочисло закінчилось переходимо до наступного кроку, перевіряємо флаг від’ємного числа, та у випадку якщо флаг присутній, додаємо символ «-» на початок числа.



На даному кроці оброблення числа завершено, саме число знаходиться у *DI*, викликаємо функцію виведення на екран нашого числа:

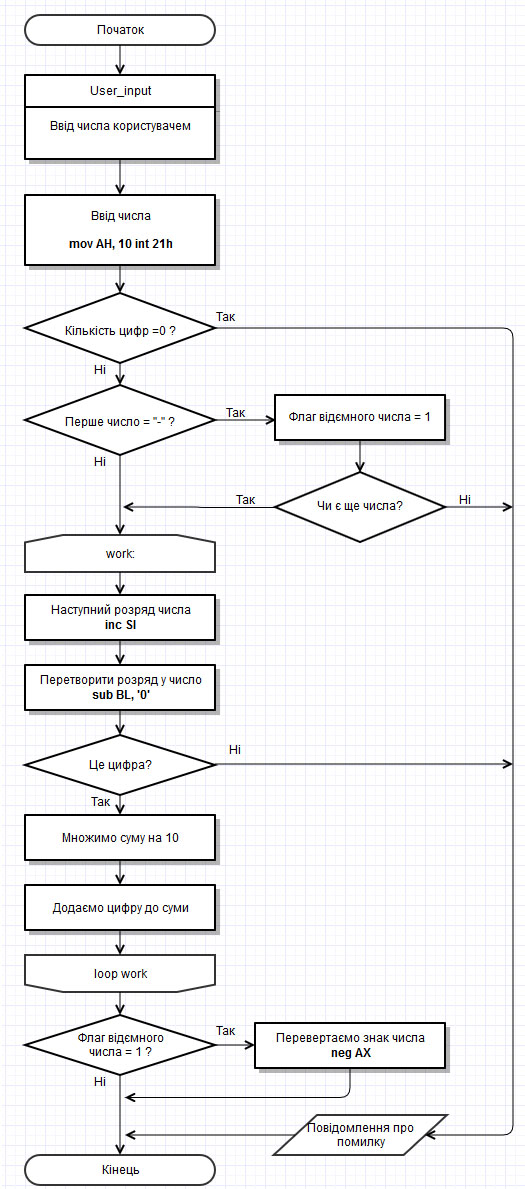


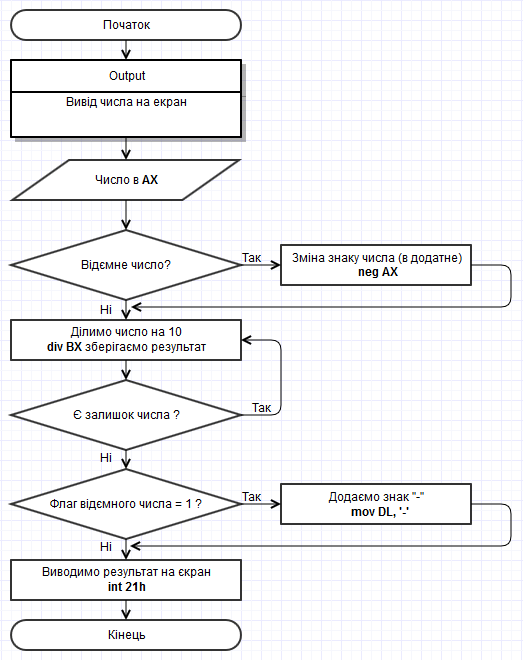
Ми закінчили вивід нашого числа на екран, виконуємо вихід із програми після натиску будь-якої клавіші:



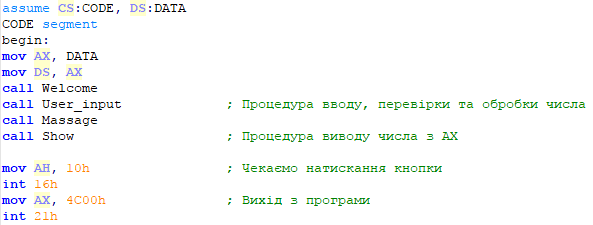
Виконання програми завершено.

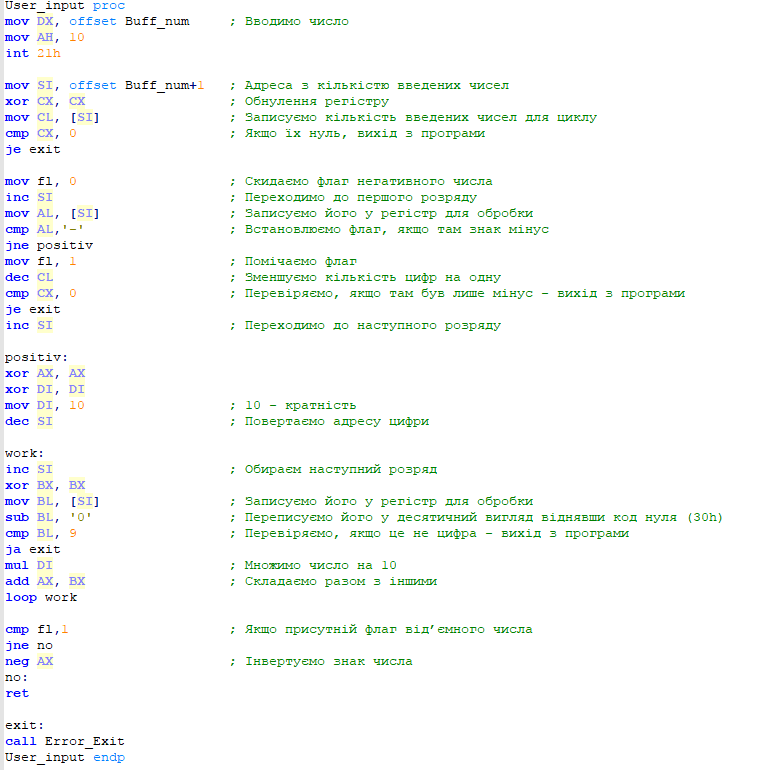
Алгоритм роботи:

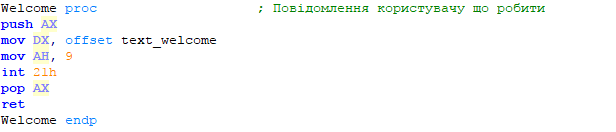
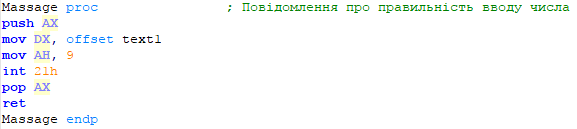


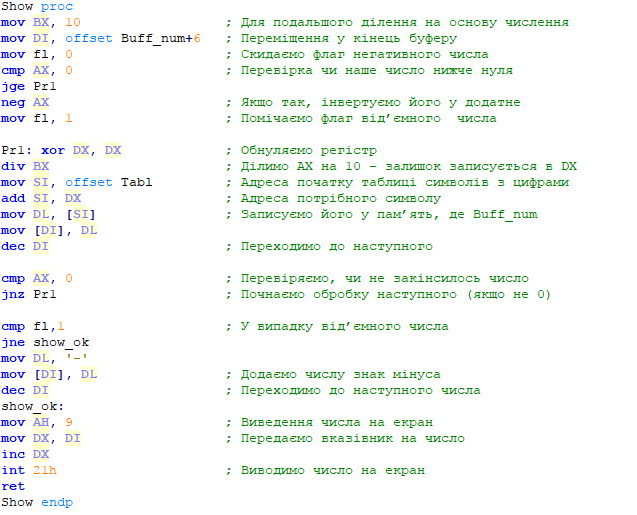


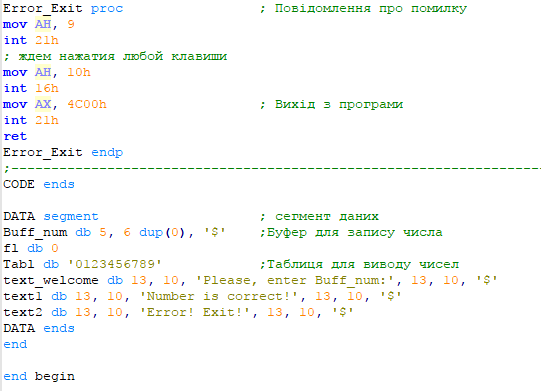
Лістинг коду:









Висновок:

Програма виконує поставлені задачі. Перевірка даних виконується. Програма працює правильно.