Міністерство освіти і науки України   
Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №8 з дисципліни «Операційні системи»  
**Макрозасоби та процедури мови Асемблер**

**Варіант: 15**

Виконав студент групи ТР-12

Каркушевський Владислав

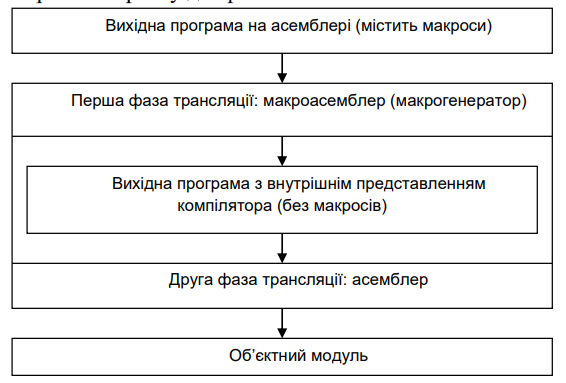
Перевірила д.т.н., проф. Левченко Л. О.

КИЇВ 2022

**Мета роботи :** Вивчення макросів, макрокоманд та макророзширень та їх застосування в асемблерних програмах.

**Теоритична частина**:

Обробка програми на асемблері з використанням макрозасобів неявно здійснюється транслятором у дві фази:



**Макрозасоби Асемблера IBM PC мають три складові:**

1. **Макровизначення** (макрос) – набір команд, який містить опис якоїсь дії або алгоритму. Макрос повинен знаходитися на початку програми, до визначення сегментів. Синтаксис макровизначення: ім’я\_макроса MACRO [список\_формальних\_аргументів] ;; тіло макровизначення Еndm

2. **Макрокоманда** – коротке посилання на макровизначення (виклик макроса): ім’я\_макроса MACRO [список\_формальних\_аргументів]

3. **Макророзширення** (макропідстановка, макровставка) – вставка замість макрокоманди макроса з заміною формальних параметрів на фактичні (якщо вони є).

Макровизначення може простим та вкладеним, тобто містити у собі інше макровизначення. Рівень вкладання макровизначень може бути будь-яким, з одного макроса можна викликати інші макроси.

**Існує три варіанти де повинні розташовуватися макровизначення:**

1. **На початку тексту програми** до сегмента коду та даних. Цей варіант використовується тоді, коли визначені користувачем макрокоманди є актуальними в межах однієї програми.

2. **В окремому файлі**. Такий варіант підходить при роботі з декількома програмами однієї проблемної області. Для того, щоб зробити доступними макровизначення у конкретній програмі, слід записати директиву include ім’я\_файлу. Наприклад: .model small include show.inc ;в це місце буде вставлено текст файлу show.inc ...

3. **В макробібліотеці**. Універсальні макрокоманди, які часто використовуються в програмах користувача, (наприклад, фрагменти програмної затримки, призупинення програми до натискання клавіші, перетворення двійкових чисел у символьну форму) доцільно записати в макробібліотеку. Макробібліотека являє собою файл з текстами макровизначень, які записуються у цей файл, як у текст програми. Файл макробібліотеки може мати будь-яке ім’я і розширення, наприклад, MYMACRO.MAC. В програмі залишаються тільки макровиклики. Включати макрокоманди з цієї бібліотеки в програму також можна за допомогою директиви include (наприклад, include mymacro.mac ). Після цього у програмі можна використовувати будь-які макрокоманди з цієї макробібліотеки.

Для організації **циклу** у макровизначенні мітку треба оголосити за допомогою оператора local.

**LOCAL** мітка... перераховує мітки, які застосовуватимуться усередині макроозначення, щоб не виникало помилки «мітка вже визначена» при використанні макросу більше одного разу або якщо та ж мітка присутня в основному тексті програми (в WASM директива LOCAL дозволяє використовувати макрос з мітками кілька разів, але не дозволяє застосовувати мітку з тим же ім'ям в програмі). Операнд для LOCAL мітка або список міток, які використовуватимуться в макросі.

**Псевдооператори EQU та „=” (дорівнює)**

До простих засобів макроасемблера відноситься псевдооператори EQU та „=”. Ці псевдооператори призначені для присвоєння деякому виразу символічного імені або ідентифікатора. При трансляції макроасемблер підставить замість нього відповідний вираз (це можуть бути константи, імена міток, символічні імена та рядки в апострофах).

**Синтаксис псевдооператора equ:**

ім’я\_ідентифікатора equ рядок або числовий\_вираз

**Синтаксис псевдооператора “=”:**

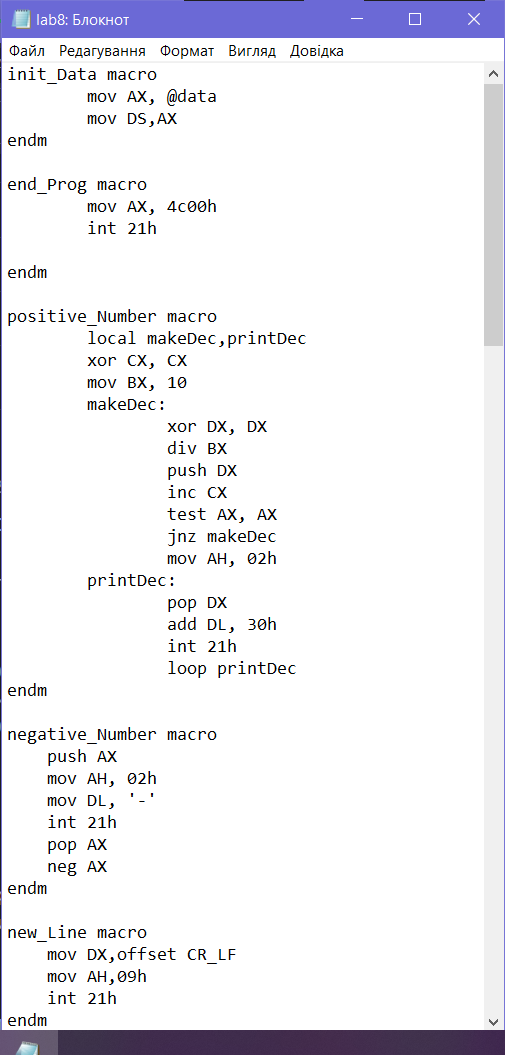
ім’я\_ідентифікатора = числовий\_вираз

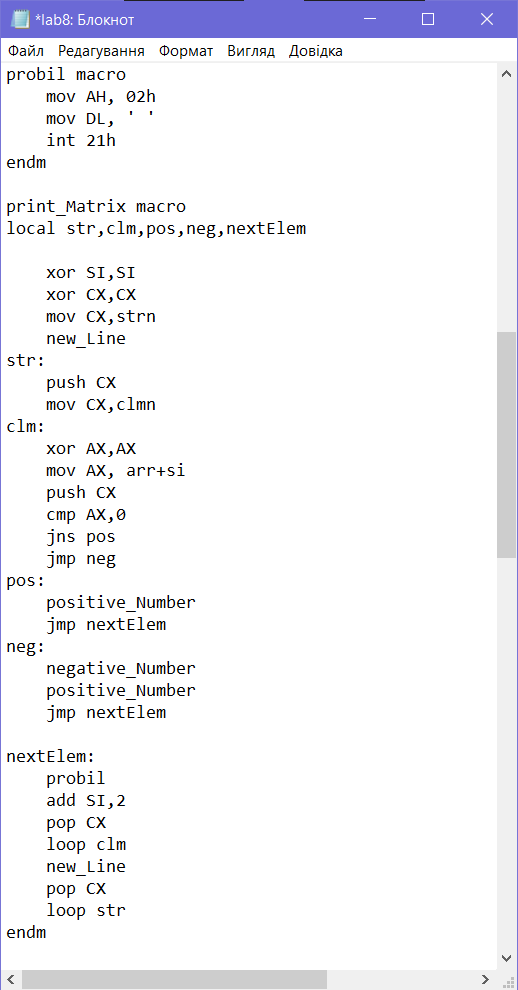
Псевдооператор “=” може використовуватися тільки з числовими виразами, тоді як псевдооператори EQU може працювати і з текстовими виразами. Псевдооператор equ зручно використовувати для налаштування програми на конкретні умови виконання, заміни складних в позначенні об’єктів, які багатократно використовуються в програмі, більш простими іменами.

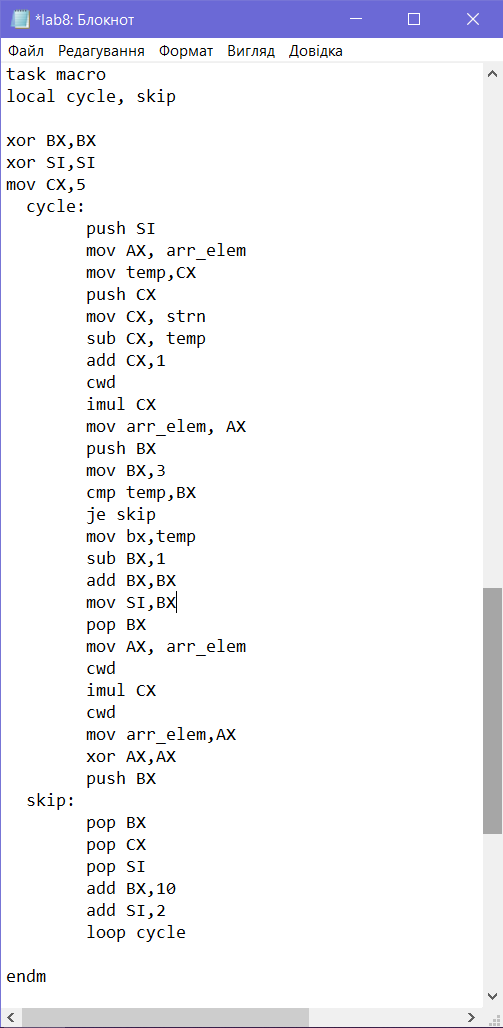
**Завдання**

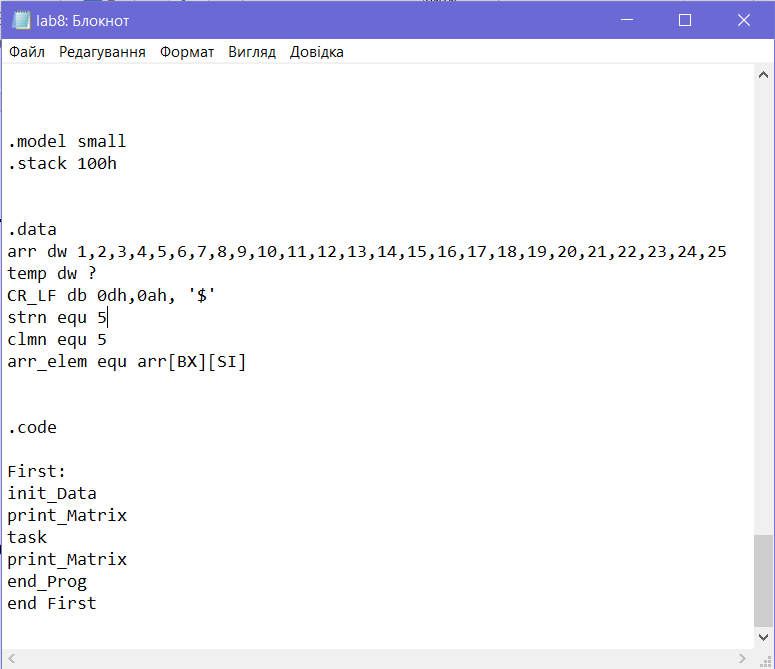
15. В сегменті даних створити квадратну матрицю 5\*5. Діагональні елементи помножити на номер рядка, в якому він знаходиться.

**Код програми:**

****







**Розглянемо детальніше код програми, та пояснимо основні дії:**

**init\_Data macro ;** Макрос завантаження сегменту данних

**mov AX, @data** ; ініціалізація регістру ds

**mov DS,AX** ; за адресою сегмента даних

**endm** ;кінець макросу

**end\_Prog macro** ; макрос завершення програми

**mov AX, 4c00h** ;коректно завершуємо нашу програму, виходимо

**int 21h** ; виклик DOS - функція переривання

**endm** ;кінець макросу

**positive\_Number macro** ; Макрос для виведення позитивного числа

**local makeDec,printDec** ; локальні мітки

**xor CX, CX** ; обнуляємо регістр сх

**mov BX, 10** ;завантажуємо основу 10-ї системи числення

**makeDec:** ;локальна мітка

**xor DX, DX** ; обнуляємо регістр dх

**div BX**  ; виконуємо ділення на 10

**push DX** ; результат у зворотньому порядку утворює число у

;новій системі числення

**inc CX** ;збільшуємо на одиницю

**test AX, AX** ;перевіряємо чи дорівнює нулю

**jnz makeDec** ; ділимо поки ціла частина не буде дорівнювати нулю

**mov AH, 02h;** виводимо остачу

**printDec:** ;локальна мітка

**pop DX** ; дістаємо зі стеку збережені числа

**add DL, 30h** ; переводимо їх в ASCII-коди

**int 21h** ; виклик DOS - функція переривання

**loop printDec** ; повторюємо цикл

**endm** ;кінець макросу

**negative\_Number macro** Макрос для виведення негативного числа

**push AX** ; зберігаємо в стек

**mov AH, 02h**

**mov DL, '-'** ;виведення «-»

**int 21h** ; виклик DOS - функція переривання

**pop AX** ; дістаємо зі стеку збережені числа

**neg AX**  ; перетворюємо від’ємне число у позитивне

**endm** ;кінець макросу

**new\_Line macro** ; макрос для перенесення рядка

**mov DX,offset CR\_LF**

**mov AH,09h**

**int 21h** ; виклик DOS - функція переривання

**endm** ;кінець макросу

**probil macro** ;Макрос для виведення пробілу

**mov AH, 02h**

**mov DL, ' '**

**int 21h** ; виклик DOS - функція переривання

**endm** ;кінець макросу

**print\_Matrix macro**; макрос для виведення матриці

**local str,clm,pos,neg,nextElem** ; локальні мітки

**xor SI,SI** ; обнуляємо регістр si

**xor CX,CX** ; обнуляємо регістр сх

**mov CX,strn ;** в CX передаємо strn

**new\_Line** ; переходимо на новий рядок

**str:** ;локальна мітка

**push CX** ; зберігаємо в стек

**mov CX,clmn** в CX передаємо clmn

**clm:** ;локальна мітка

**xor AX,AX** ; обнуляємо регістр aх

**mov AX, arr+si** ; matrix+si → аx

**push CX** ; зберігаємо в стек

**cmp AX,0** ; перевіряємо чи AX дорівнює нулю

**jns pos** ; якщо знак плюс - переходимо на локальну мітку pos

**jmp neg** ; якщо ні - переходимо на локальну мітку neg

**pos:** ;локальна мітка

**positive\_Number** ;виводимо додатнє число

**jmp nextElem** ;переходимо на nextElem

**neg:** ;локальна мітка

**negative\_Number** ;виводимо від’ємне число

**positive\_Number**

**jmp nextElem** ;переходимо на nextElem

**nextElem:** ;локальна мітка

**probil** ; виводимо пробіл

**add SI,2** ; до SI додаємо 2

**pop CX** ; дістаємо зі стеку збережені числа

**loop clm** ; повторюємо цикл

**new\_Line** ; переходимо на новий рядок

**pop CX**  ; дістаємо зі стеку збережені числа

**loop str** ; повторюємо цикл

**endm** ;кінець макросу

**task macro** ; макрос із завданням згідно варіанту

**local cycle, skip** ; локальні мітки

**xor BX,BX** ; обнуляємо регістр bх

**xor SI,SI** ; обнуляємо регістр si

**mov CX,5 ; cx 🡪 5**

**cycle:** ;локальна мітка

**push SI** ; зберігаємо в стек

**mov AX, arr\_elem** ; arr\_elem 🡪 ax

**mov temp,CX** ; cx 🡪 temp

**push CX** ; зберігаємо в стек

**mov CX, strn** ; strn 🡪 cx

**sub CX, temp ;** cx - temp 🡪 cx

**add CX,1 ;** cx + 1 🡪 cx

**cwd** ;команда перетворення слова у подвійне слово

**imul CX ; ax\*cx 🡪 ax** тобто множимо на номер рядка

**mov arr\_elem, AX** ; ax 🡪 arr\_elem

**push BX** ; зберігаємо в стек

**mov BX,3 ; 3 🡪 bx**

**cmp temp,BX** ; порівнюємо temp і bx

**je skip** ; якщо дорівнює то skip

**mov bx,temp ;** temp 🡪 bx

**sub BX,1 ;** bx-1 🡪 bx

**add BX,BX** ; bx + bx 🡪 bx

**mov SI,BX** ; bx 🡪 si

**pop BX** ; дістаємо зі стеку збережені числа

**mov AX, arr\_elem ;** arr\_elem 🡪 ax

**cwd** ;команда перетворення слова у подвійне слово

**imul CX ; ax\*cx 🡪 ax** тобто множимо на номер рядка

**cwd** ;команда перетворення слова у подвійне слово

**mov arr\_elem,AX** ; ax 🡪 arr\_elem

**xor AX,AX** ; обнуляємо регістр aх

**push BX** ; зберігаємо в стек

**skip:** ;локальна мітка

**pop BX** ; дістаємо зі стеку збережені числа

**pop CX** ; дістаємо зі стеку збережені числа

**pop SI** ; дістаємо зі стеку збережені числа

**add BX,10 ; bx + 10 🡪 bx**

**add SI,2 ; si+2 🡪 si**

**loop cycle** ; повторюємо цикл

**endm** ;кінець макросу

**.model small** ; модель пам’яті для ехе-файлів

**.stack 100h** ;виділяємо PSP 256 байт

**.data** ; початок сегменту даних

**arr dw 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25** ; числа ;матриці

**temp dw ?** ; створюємо додаткову змінну

**CR\_LF db 0dh,0ah, '$'**

**strn equ 5 ; кількість рядків = 5**

**clmn equ 5 ; кількість стовпців = 5**

**arr\_elem equ arr[BX][SI] ; arr\_elem дорівнює arr[BX][SI]**

**.code** ; початок сегменту коду

**First:** ; точка входу

**init\_Data** ; завнатажуємо сегмент даних

**print\_Matrix** ; виводимо матрицю

**task** ; виконуємо завдання згіднно варіанту

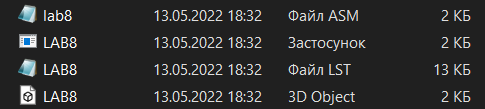
**print\_Matrix** ; виводимо матрицю

**end\_Prog** ; закінчуємо програму

**end First** ; кінець точки входу

Запустимо **транслятор** та створими лістинг(\*.lst) та об’єктний файл(\*.obj) за допомогою команди: **tasm /la /zi lab7.asm**

Запустимо **компоновщик** та створимо файл (\*.exe) за допомогою команди: **tlink /x /v lab7.obj**

****

Запустимо нашу програму ввівши команду **lab7.exe**

****

**1 \* 1 = 1 5 \* 1 = 5**

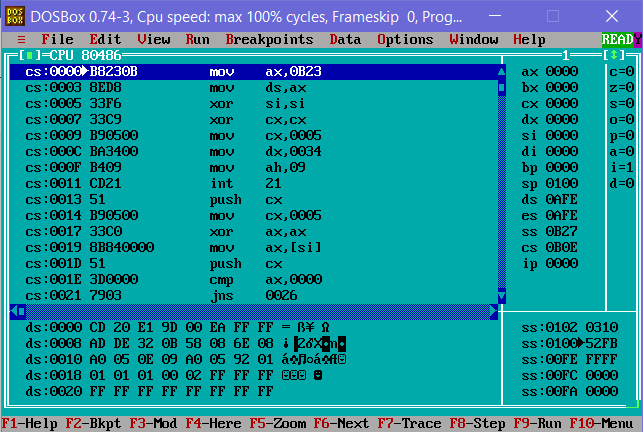
**7 \* 2 = 14 9 \* 2 = 18**

**13 \* 3 = 39**

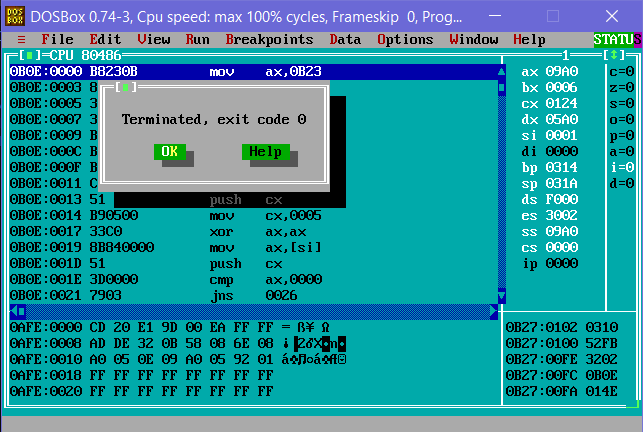
**17 \* 4 = 68 19 \* 4 = 76**

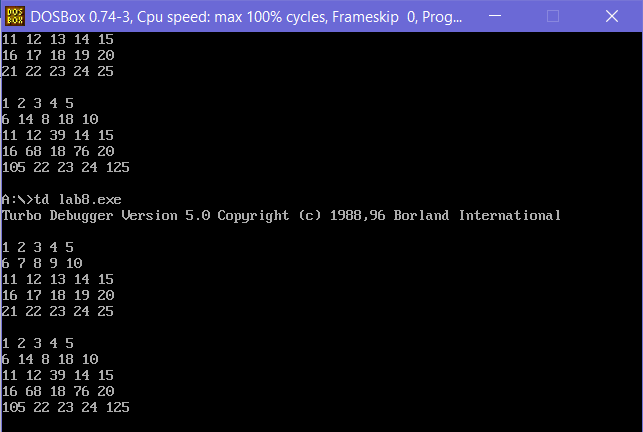
**21 \* 5 = 105 25 \* 5 = 125**

Запустимо **налагоджувач**

****

За допомогою клавіші **F8**(або **F7**) покроково виконуємо нашу програму і виводимо отримане словосполучення в консольному режимі

****

****

**Висновок:**

На цій лабораторній роботі я ознайомився з макрозасобами мови асемблер та отримав практичні навички роботи з ними. Було підготовлено та реалізовано програму відповідно до індивідуального завдання. Роботу програми продемоностровано під керуванням налагоджувача. Було створено розташовання макровизначення на початку тексту програми до сегмента коду та даних. Створено макроси для завантаження сегменту данних, завершення програми, для виведення позитивного і негативного числа, для перенесення рядка, для виведення пробілу, для виведення матриці та макрос із множенням діагональних елементів на номер рядка в якому вони знаходяться.