МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи №5
на тему: «Складення та відлагодження циклічної програми мовою асемблера мікропроцесорів x86 для Windows»
з дисципліни: «Архітектура комп'ютера»

Лектор:
доц. кафедри ПЗ
Крук О. Г.

Виконав:
ст. гр. ПЗ-22
Чаус О. М.

Прийняв:
доц. кафедри ПЗ
Крук О. Г.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Складення та відлагодження циклічної програми мовою асемблера мікропроцесорів x86 для Windows

Мета роботи: ознайомитись на прикладі циклічної програми з основними командами асемблера; розвинути навики складання програми з вкладеними циклами; відтранслювати і виконати в режимі відлагодження програму, складену відповідно до свого варіанту; перевірити виконання тесту.

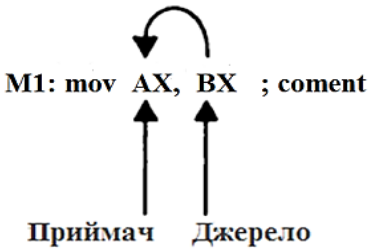
Варіант 27

27	(6 × 8)	1. Обчисліть скалярний добуток 2-го і 4-го стовпців. 2. Обчисліть кількість і суму елементів 5-го рядка, які задовільняють вказаній умові.	-32	78	$a_i <= b$ або $a_i >= c$

Теоретичні відомості

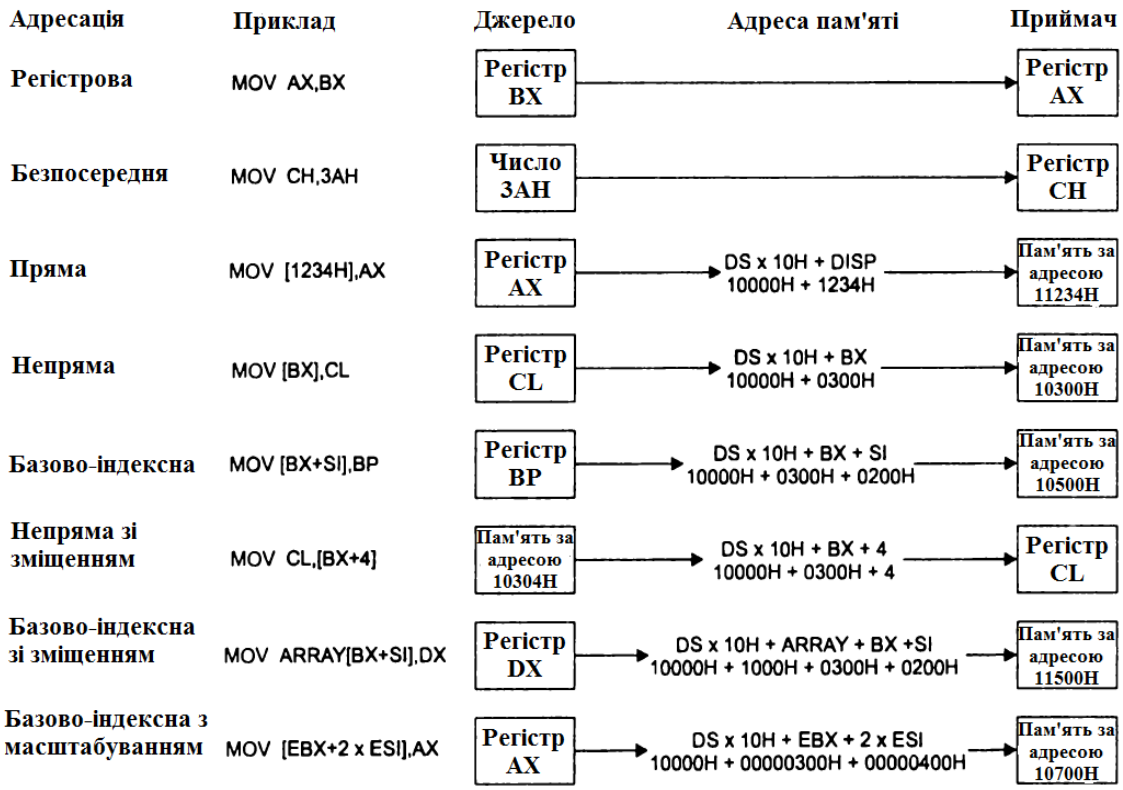
Методи адресації даних.

Методи адресації даних розглянемо з використанням інструкції mov (пересилання), оскільки вона дуже поширена і, до того ж, допускає всі можливі методи адресації Інструкції (оператори або команди) мови асемблера складаються з чотирьох частин або полів. Оператор починається з мітки, яка позначає поточну адресу програми в символічному вигляді. Мітка починається буквою або одним із символів @, $, _ або ? і містить не більше 35 символів. Мітка закінчується символом двокрапки. Наступне поле в операторі асемблера призначене для мнемоніки інструкції; за мнемонікою знаходиться поле операндів. Крайнє поле праворуч призначене для записування коментарів – від крапки з комою до кінця рядка. Наприклад, в операторі M1: mov AX, BX; coment поле M1 – мітка, слово mov – мнемоніка, AX і BX – операнди, а coment – коментар. Зауважимо, що число операндів залежить від інструкції і може змінюватися від нуля до трьох. Операнди вказуються в такому порядку: першим – операнд-приймач, другим – операнд-джерело.



Інструкція MOV, її операнди і напрямок передавання даних

Команда mov AX, BX на рис. 6 пересилає 16-бітовий вміст джерела – регістра BX в приймач – регістр AX. Джерело при цьому не змінюється. Команди пересилання даних не впливають на стан прапорців. На рис. 2.7 наведені всі можливі поєднання методів адресації даних при використанні команди mov.



Методи адресації даних у процесорах 8086 – Pentium 4

Наведені методи можуть застосовуватись для всіх моделей мікропроцесорів Intel, крім індексної адресації з масштабуванням, яка вперше з'явилась у мікропроцесорі 80386.

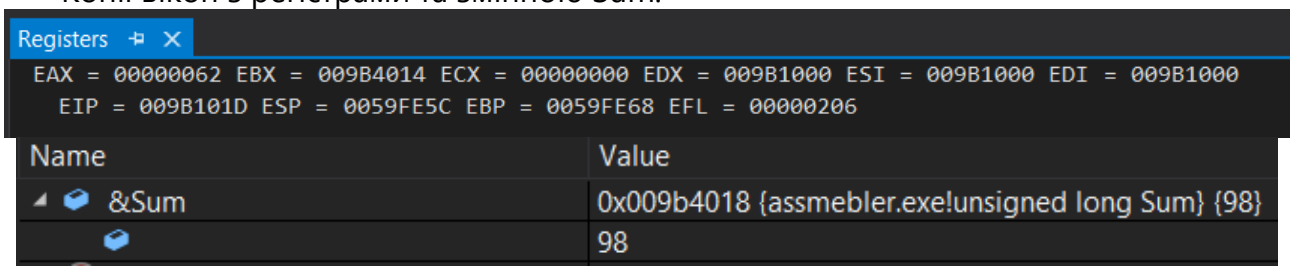
Хід роботи

1. Створив асемблерний файл та ввів в нього наведений файл.
2. Виконав програму і режимі відлагодження.
3. Вручну порахував суму елементів матриці $Sum = 17 + 3 - 51 + 242 - 113 = 98$, та переконався, що програма коректно працює.

Текст програми:

```
.586P
; плоска модель пам'яті
.MODEL FLAT, STDCALL
;-----------------------------------------------
; сегмент даних
_DATA SEGMENT
Num1      DD 17, 3, -51, 242, -113     ;  Оголошення масиву чисел, кожне з яких займає
подвійне слово
N             DD 5              ;  Кількість елементів в масиві Num1
Sum        DD 0                 ;  Сума елементів масиву Num1
_DATA ENDS
; сегмент коду
_TEXT SEGMENT
START:
      lea EBX, Num1             ;   Завантажуємо в BX адресу першого елемента масиву
Num1
      mov ECX, N                ;  Завантажуємо в CX кількість елементів в масиві Num1
      mov EAX, 0                ;  В AX буде сума елементів масиву Num1
M1:     add EAX, [EBX]          ;  Додаємо до AX поточний елемент масиву Num1
      add EBX, 4                ;  Формуємо адресу наступного елемента масиву Num1
      loop M1                   ;  Декрементує CX і якщо CX не дорівнює нулю, то на M1
      mov Sum, EAX        ;  Цикл завершений. Зберігаємо обчислену суму в змінній Sum
      RET                       ;  вихід
_TEXT ENDS
   END START
```

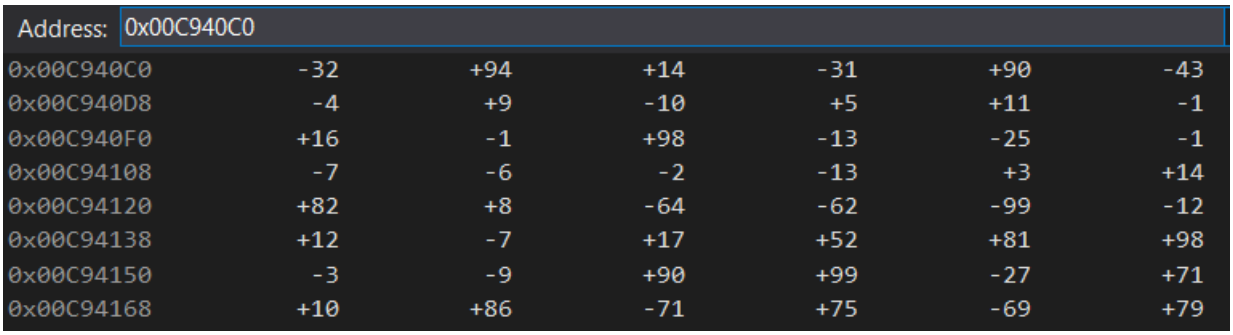
Копії вікон з регістрами та змінною Sum:



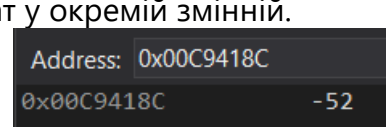
4. Створив новий проект.
5. Ініціалізував двовимірний масив з різними довільними дворозрядними цілими додатними або від'ємними числами

```
orig_array DD  -32,  -4,  16,  -7,  82, 12,  -3,  10
           DD   94,   9,  -1,  -6,   8, -7,  -9,  86
           DD   14, -10,  98,  -2, -64, 17,  90, -71
           DD  -31,   5, -13, -13, -62, 52,  99,  75
           DD   90,  11, -25,   3, -99, 81, -27, -69
           DD  -43,  -1,  -1,  14, -12, 98,  71,  79
```

6. Написав фрагмент коду для транспонування матриці та зберіг її в новому масиві

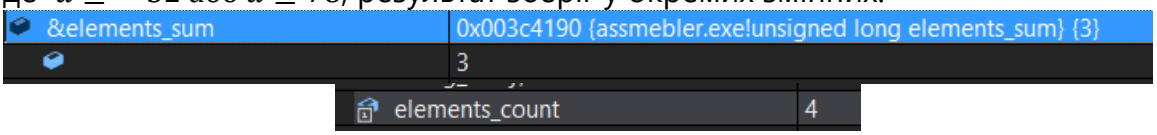


7. Реалізував фрагмент коду для обчислення скалярного добутку 2-го та 4-го стовпців та зберіг результат у окремій змінній.



Для перевірки також порахував результат вручну:
$scalar = (-4 \cdot (-7)) + (9 \cdot (-6)) + (-10 \cdot (-2)) + (5 \cdot (-13)) + (11 \cdot 3) + (-1 \cdot 14)$
$scalar = 28 - 54 + 20 - 65 + 33 - 14 = -52.$

8. Реалізував фрагмент коду для обчислення суми та кількості елементів 5 рядка, де $a \leq -32$ або $a \geq 78$, результат зберіг у окремих змінних.



Для перевірки порахував результат вручну:
$sum = 90 - 99 + 81 - 69 = 3.$

Текст програми:

```
.586P
.MODEL FLAT, STDCALL

_DATA SEGMENT
orig_array DD  -32,  -4,  16, -98,  82, 12,  -3,  10
       DD   94, 149,  -1, -56,   8, -7,  -9,  86
       DD   14, -20,  98,  -5, -64, 17, 190, -71
       DD  -31,  45, -13, -20, -62, 52,  99,  75
       DD   90,  18, -25, 157, -99, 21, -27, -69
       DD  -43, -11,  -1,  79, -12, 98,  71,  79
transposed_array DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
                 DD 0, 0, 0, 0, 0, 0
rows DD 6
columns DD 8
count_columns DD 0
scalar DD 0
elements_sum DD 0
elements_count DD 0
_DATA ENDS

_TEXT SEGMENT
START:
  transpose:
    lea EBP, orig_array
    lea EBX, transposed_array
    mov ECX, rows
    l1:
      mov EDX, [EBP]
      mov [EBX], EDX
      add EBP, 32
      add EBX, 4
    loop l1
    lea EBP, orig_array
    add count_columns, 1
    mov EAX, count_columns
    imul EAX, 4
    add EBP, EAX
    mov ECX, rows
    mov EAX, count_columns
    cmp EAX, columns
    jne l1;
  scalar_multiply_columns:
    lea EBP, orig_array
    add EBP, 4
    mov EAX, EBP
    add EAX, 8
    mov ECX, rows
    mov EBX, 0
    l2:
      mov EDX, [EBP]
      imul EDX, [EAX]
      add EBX, EDX
      add EBP, 32
      add EAX, 32
    loop l2
    mov scalar, EBX
  sum_elements:
```

```
    lea EBP, orig_array
    add EBP, 128
    mov ECX, columns
    mov EBX, 0
    l3:
      mov EAX, -32
      cmp [EBP], EAX
      jle if_true

      mov EAX, 78
      cmp [EBP], EAX
      jge if_true
      jmp next_iter

      if_true:
      inc elements_count
      add EBX, [EBP]

      next_iter:
      add EBP, 4
      loop l3
    mov elements_sum EBP
RET
_TEXT ENDS
END START
```

Висновки: на цій лабораторній роботі я розвинув навики складання програми з вкладеними циклами мовою асемблера для опрацювання двовимірного масиву, відтранслював і виконав в режимі відлагодження програму, складену відповідно до свого варіанту, відлагодив та перевірив виконання тесту.