**МІНІСТЕРСТВО ОСВІТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи № 5

# **на тему:** *“Складення та відлагодження циклічної програми мовою асемблера мікропроцесорів х86 для Windows”*

**з дисципліни:** *“Архітектура комп’ютера”*

**Лектор:**

доц. каф. ПЗ

Крук О.Г.

**Виконав:**

ст. гр. ПЗ-22

                                                                          Солтисюк Д.А.

**Прийняв:**

доц. каф. ПЗ

Крук О.Г.

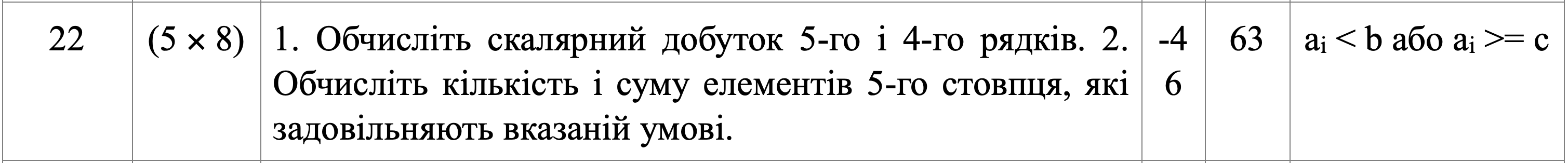
« \_\_\_\_ » \_\_\_\_\_\_\_\_  2022 р.

∑= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Львів – 2022

**Тема:** складення та відлагодження циклічної програми мовою асемблера мікропроцесорів х86 для Window*s.*

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

**Варіант: 22**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

До регістрів загального призначення належать EAX, EBX, ECX, EDX, EBP, EDI та ESI.

EAX (accumulator – акумулятор) адресується як 32-бітовий (EAX), 16-бітовий (AX) або як 8-бітовий регістр (AH та AL). При записуванні в 8- або 16-бітовий регістр решта бітів регістра EAX не змінюється. Регістр-акумулятор EAX/AX/AL використовується як обов’язковий операнд таких інструкцій, як множення, ділення, двійково-десяткова корекція тощо. В мікропроцесорах 80386 – Pentium 4 регістр EAX може використовуватись для непрямої адресації пам’яті.

EBX (base index – вказівник бази) адресується як EBX, BX, BH або BL. В усіх поколіннях мікропроцесорів він використовується як вказівник. У мікропроцесорах 80386 і вище регістр EBX також може використовуватись для непрямої адресації до пам’яті.

ECX (count – лічильник) адресується як ECX, CX, CH або CL, використовується як лічильник в інструкціях циклів, зсуву, циклічного зсуву та рядкових інструкціях з префіксами повторення REP/REPE/REPNE. В мікропроцесорах 80386 – Pentium 4 регістр ECX також може використовуватись для непрямої адресації пам’яті.

EDX (data – дані) адресується як EDX, DX, DH або DL. Його ще називають розширювачем акумулятора, в командах множення і ділення він використовується в парі з EAX/AX. У мікропроцесорах 80386 і вище регістр EDX може використовуватись як вказівник при адресації до пам’яті.

EBP (base pointer – вказівник бази) адресується як EBP, BP і в обох варіантах використовується як вказівник бази.

EDI (destination index – вказівник приймача) адресується як EDI та DI, в рядкових інструкціях використовується як вказівник операнда-приймача.

ESI (sourse index – вказівник джерела) адресується як ESI та SI, у рядкових інструкціях адресує операнд-джерело.

Інструкції регістрової адресації

|  |  |  |
| --- | --- | --- |
| Інструкція | Розмірність | Дія |
| MOV AL, BL  MOV CH, CL  MOV AX, CX  MOV SP, BP  MOV DS, AX  MOV SI, DI  MOV BX, ES  MOV ECX,EBX  MOV ESP, EDX  MOV ES, DS  MOV BL, DX  MOV CS, AX | Байт  Байт  Слово  Слово  Слово  Слово  Слово  Подвійне слово  Подвійне слово  -  -  - | Копіює BL в AL  Копіює CL в СН  Копіює СХ в АХ  Копіює ВР в SP  Копіює АХ в DS  Копіює DI в SI  Копіює ES в ВХ  Копіює ЕВХ в ЕСХ  Копіює EDX в ESP  Недопустима інструкція - копіювання сегментного регістра в сегментний регістр заборонено  Інструкція недопустима - операнди мають різну розмірність  Недопустима інструкція - сегментний регістр коду не може бути приймачем |

Інструкції прямої адресації

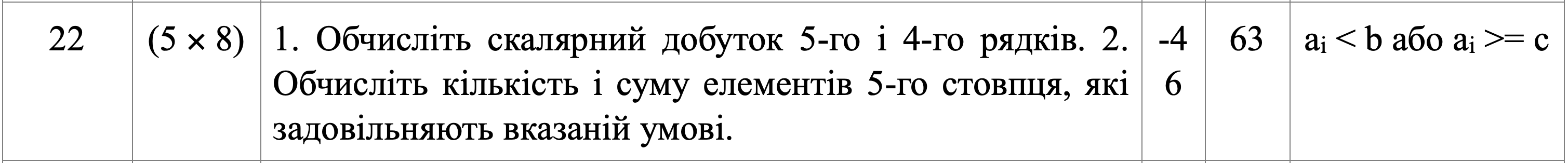
|  |  |  |
| --- | --- | --- |
| Інструкція | Розмірність | Дія |
| MOV AL, NUMBER  MOV AX, COW  MOV EAX, WATER  MOV NEWS, AL  MOV THERE, AX  MOV HOME, EAX  MOV ES:[2000H], AL  MOV CH, DOG  MOV CH, [1000H]  MOV ES, DATA6  MOV DATA7, BP  MOV NUMBER1, SP  MOV DATA1, EAX  MOV EDI, SUM1 | Байт  Слово  Подвійне слово  Байт  Слово  Подвійне слово  Байт  Байт  Байт  Слово  Слово  Слово  Подвійне слово  Подвійне слово | Копіює в AL байт з сегмента даних за зміщенням NUMBER  Копіює в АХ слово з сегмента даних за зміщенням COW  Копіює в EAX подвійне слово з сегмента даних за зміщенням WATER  Копіює AL в сегмент даних за зміщенням NEWS  Копіює АХ в сегмент даних за зміщенням THERE  Копіює EAX в сегмент даних за зміщенням HOME  Копіює AL в додатковий сегмент даних за зміщенням 2000Н  Копіює в СН байт з сегмента даних, розташований за зміщенням DOG  Копіює в СН байт з сегмента даних, розташований за зміщенням 1000Н  Копіює в ES слово з сегмента даних, розташоване за зміщенням DATA6  Копіює ВР в сегмент даних за зміщенням DATA7  Копіює SP в сегмент даних за зміщенням NUMBER  Копіює EAX в сегмент даних за зміщенням DATA1  Копіює в EDI подвійне слово, розташоване в сегменті даних за зміщенням SUM1 |

Інструкції непрямої адресації

|  |  |  |
| --- | --- | --- |
| Інструкція | Розмірність | Дія |
| MOV CX, [BX]  MOV [BP], DL\*  MOV [DI], BH  MOV [DI], [BX]  MOV AL, [EDX]  MOV ECX, [EBX] | Слово  Байт  Байт  Байт  Подвійне слово | Копіює в СХ слово, розташоване в сегменті даних за зміщенням, заданим в ВХ  Копіює DL в сегмент стека за зміщенням, заданим в ВР  Копіює ВН в сегмент даних за зміщенням, заданим в DI  Помилка - передача даних між комірками пам'яті підтримується тільки для рядкових інструкцій  Копіює в AL байт з сегмента даних, зміщення якого задано регістром EDX  Копіює в ЕСХ подвійне слово з сегмента даних, зміщення якого задано в ЕВХ |

Інструкції умовного переходу

|  |  |  |
| --- | --- | --- |
| Команда | Значення прапорців для переходу | Умова переходу |
| ja / jnbe  jae / jnb  jb / jnae  jbe / jna  jc  je / jz  jg / jnle  jge / jnl  jl / jnge  jle / jng  jnc  jnе / jnz  jno  jns  jnp / jpo  jo  jp / jpe  js  jcxz  jecxz | С = 0 і Z = 0  С = 0  С = 1  С = 1 або Z = 1  С = 1  Z = 1  Z = 0 і S = 0  S = 0  S <> 0  Z = 1 або S <> 0  С = 0  Z = 0  О = 0  S = 0  P = 0  О = 1  P = 1  S = 1  CX = 0  ECX = 0 | Беззнакове більше (above)  Беззнакове більше або рівне (above or equal)  Беззнакове менше (below)  Беззнакове менше або рівне (below or equal) Встановлений прапорець переносу  Рівне / Нуль (equal / zero)  Знакове більше (greater than)  Знакове більше або рівне (greater than or equal)  Знакове менше (less than)  Знакове менше або рівне (less than or equal) Немає переносу  Не рівне / Не нуль (not equal / not zero)  Немає переповнення  Немає знака (no sign)  Немає паритету (no parity)  Встановлений прапорець переповнення Встановлений прапорець паритету (parity)  Встановлений прапорець знака (sign)  Вміст регістра СХ дорівнює нулю  Вміст регістра ЕСХ дорівнює нулю |

**Індивідуальне завдання**

**Хід роботи**

Приклад циклічної 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 М1

mov Sum, EAX ; Цикл завершений. Зберігаємо обчислену суму в змінній Sum

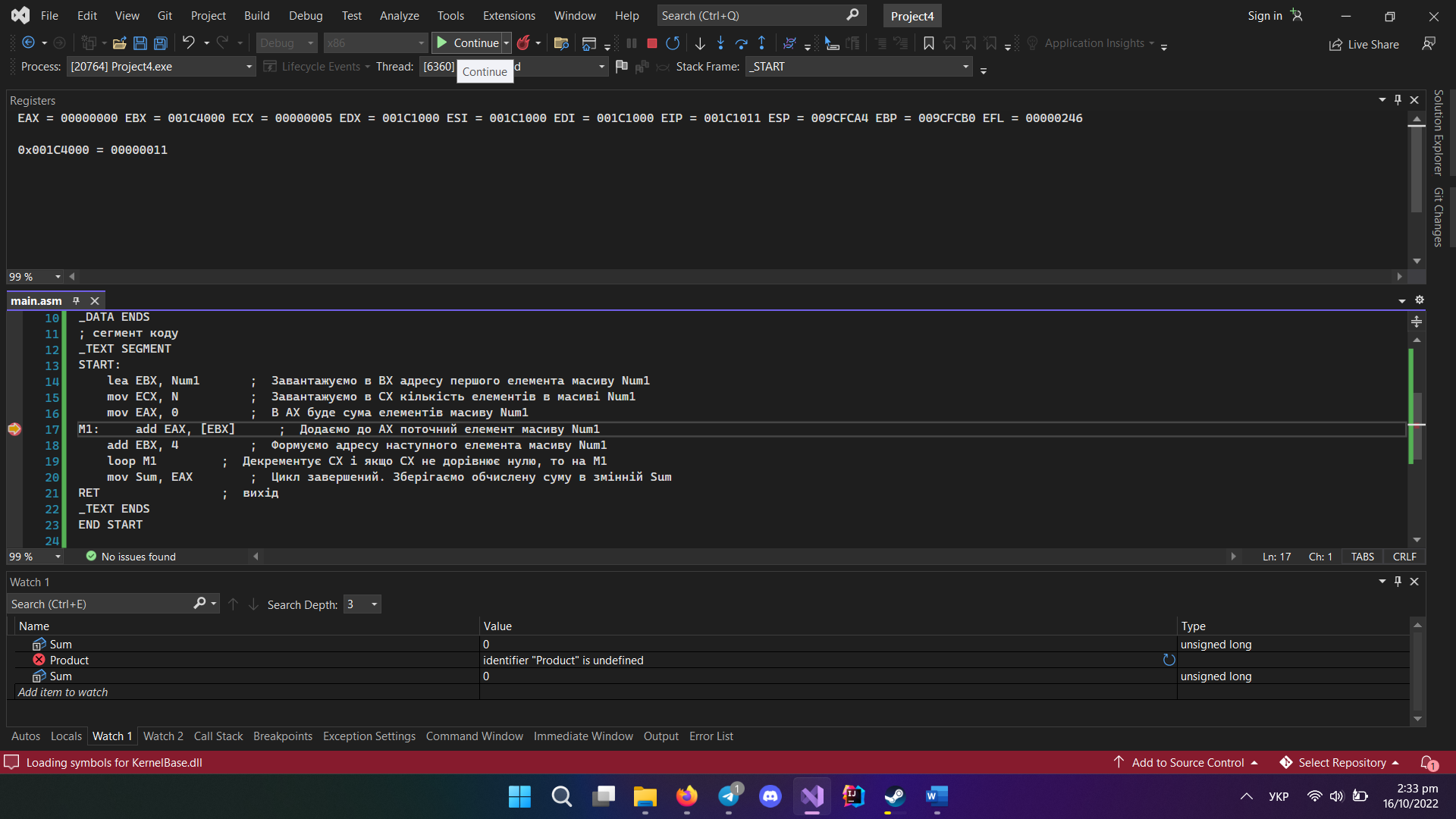
RET ; вихід

\_TEXT ENDS

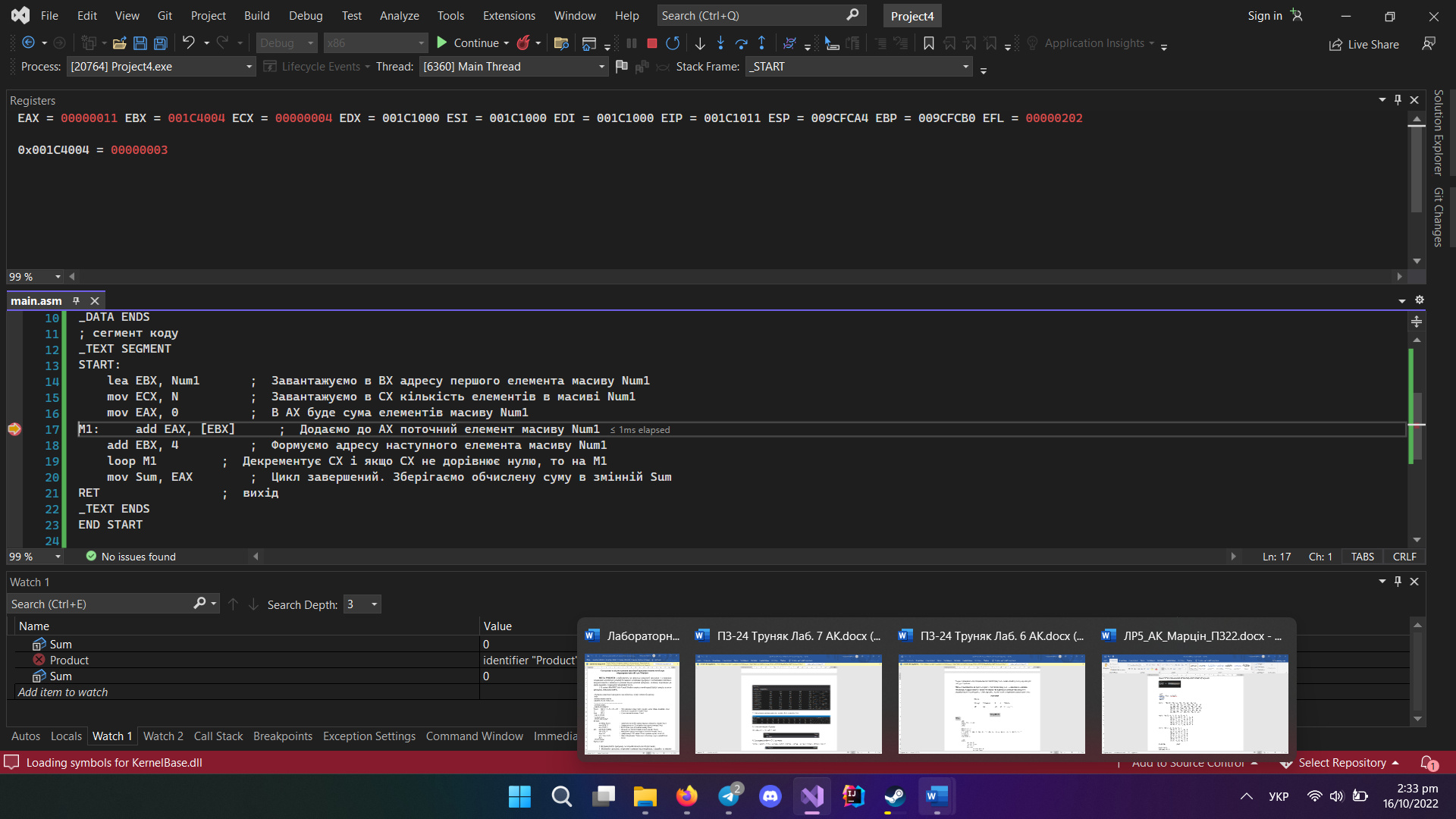
END START

**Значення регістру EAX під час виконання**

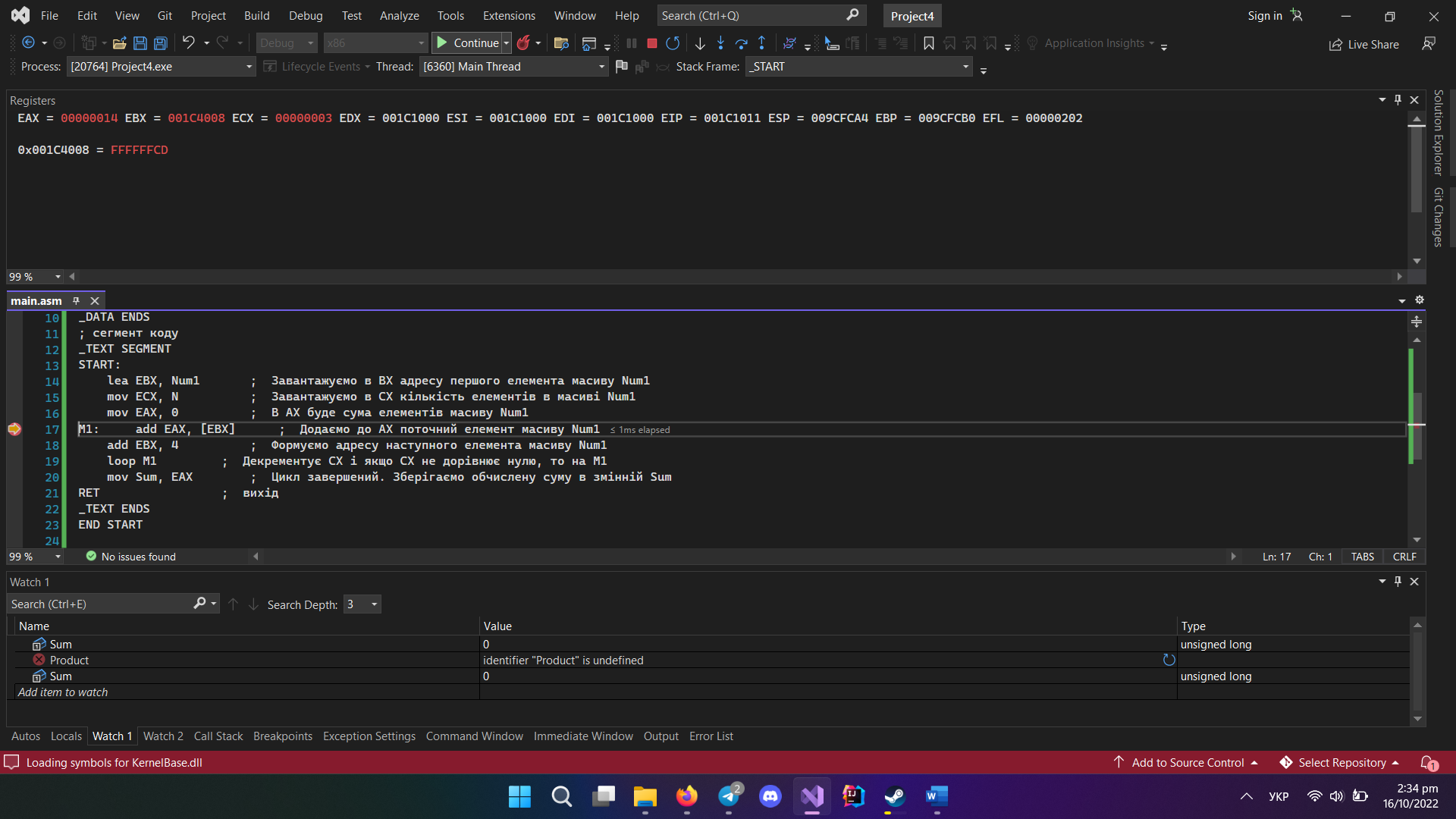
Перед ітераціями:



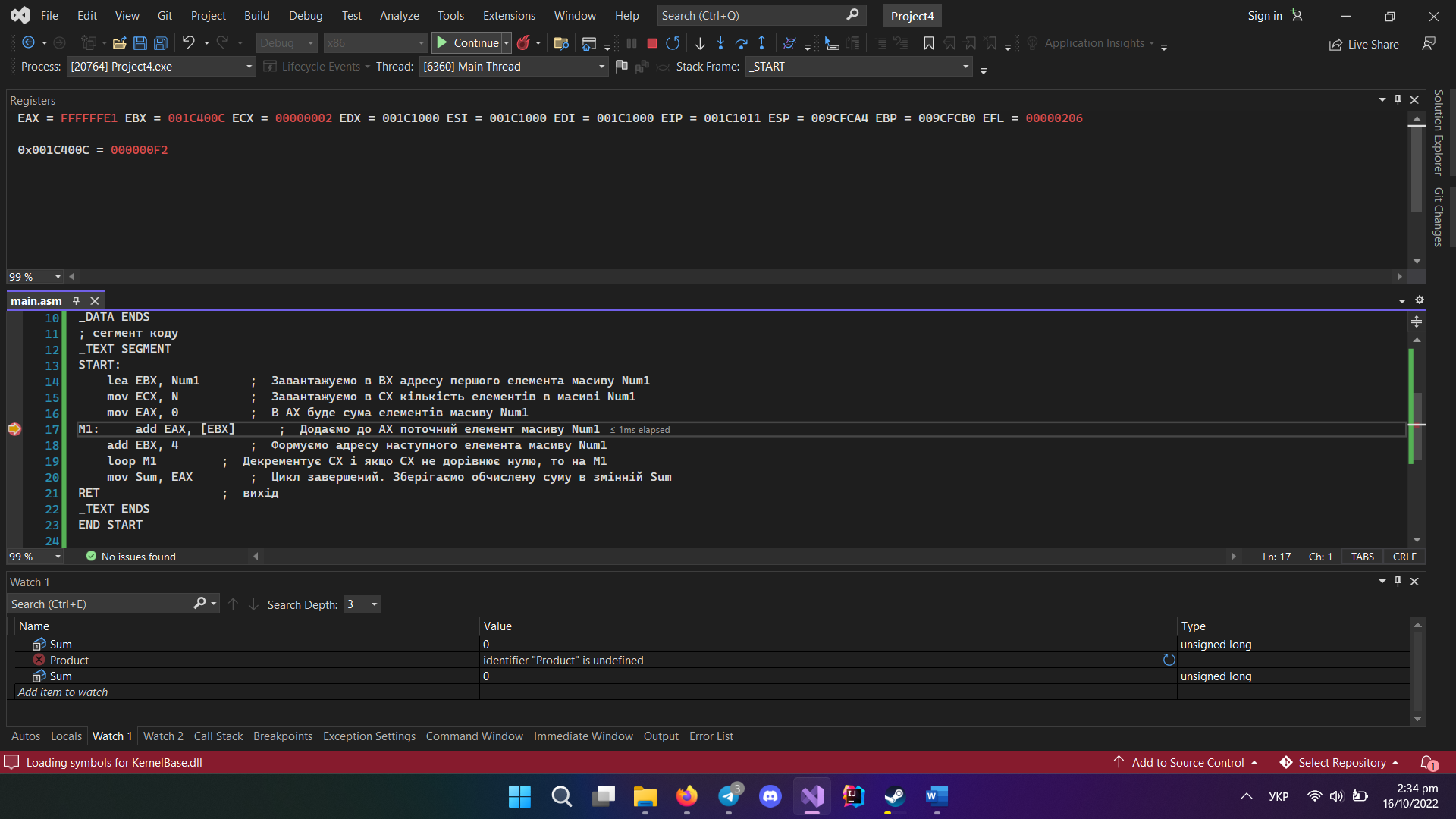
Ітерація 1:



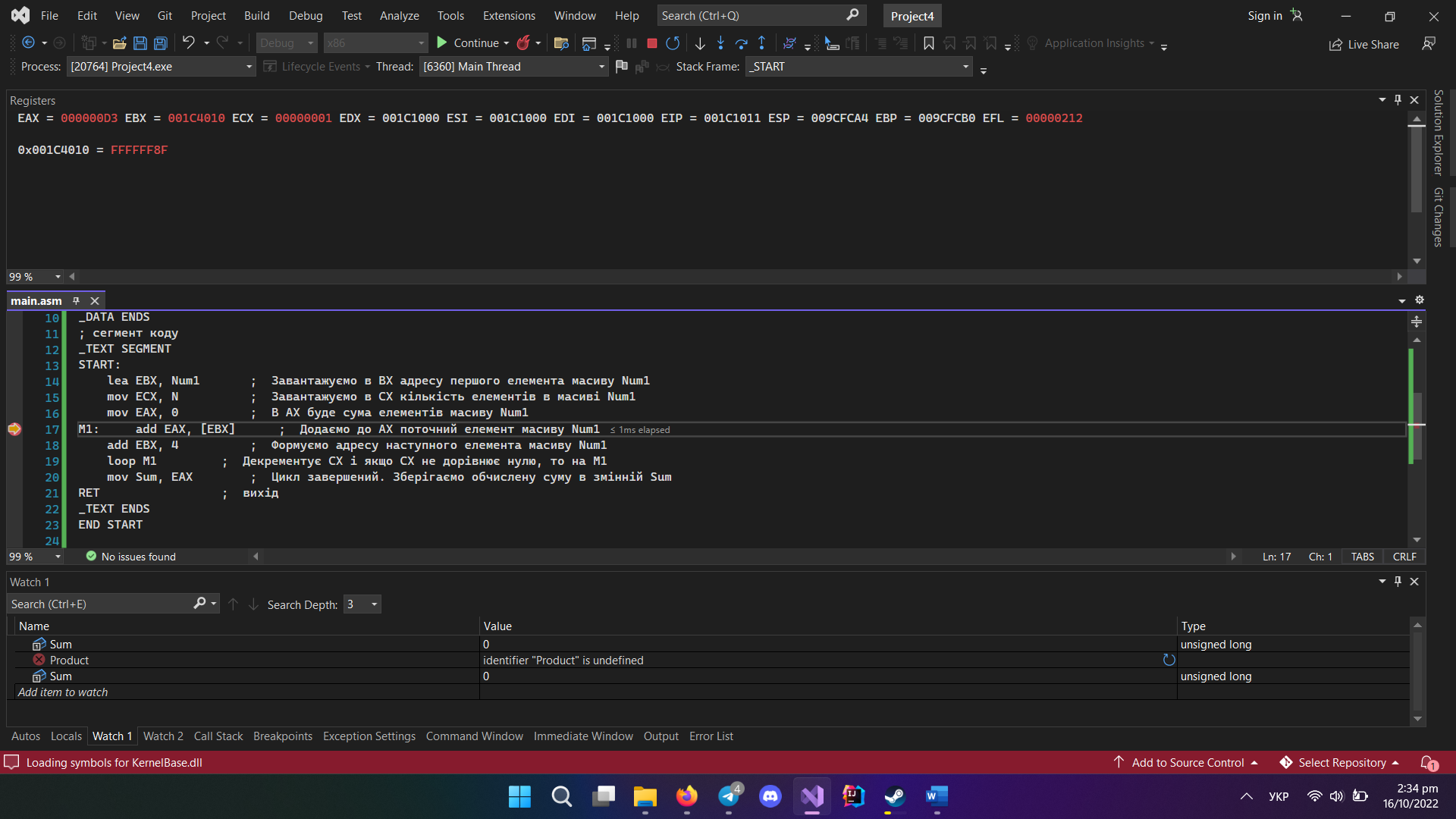
Ітерація 2:



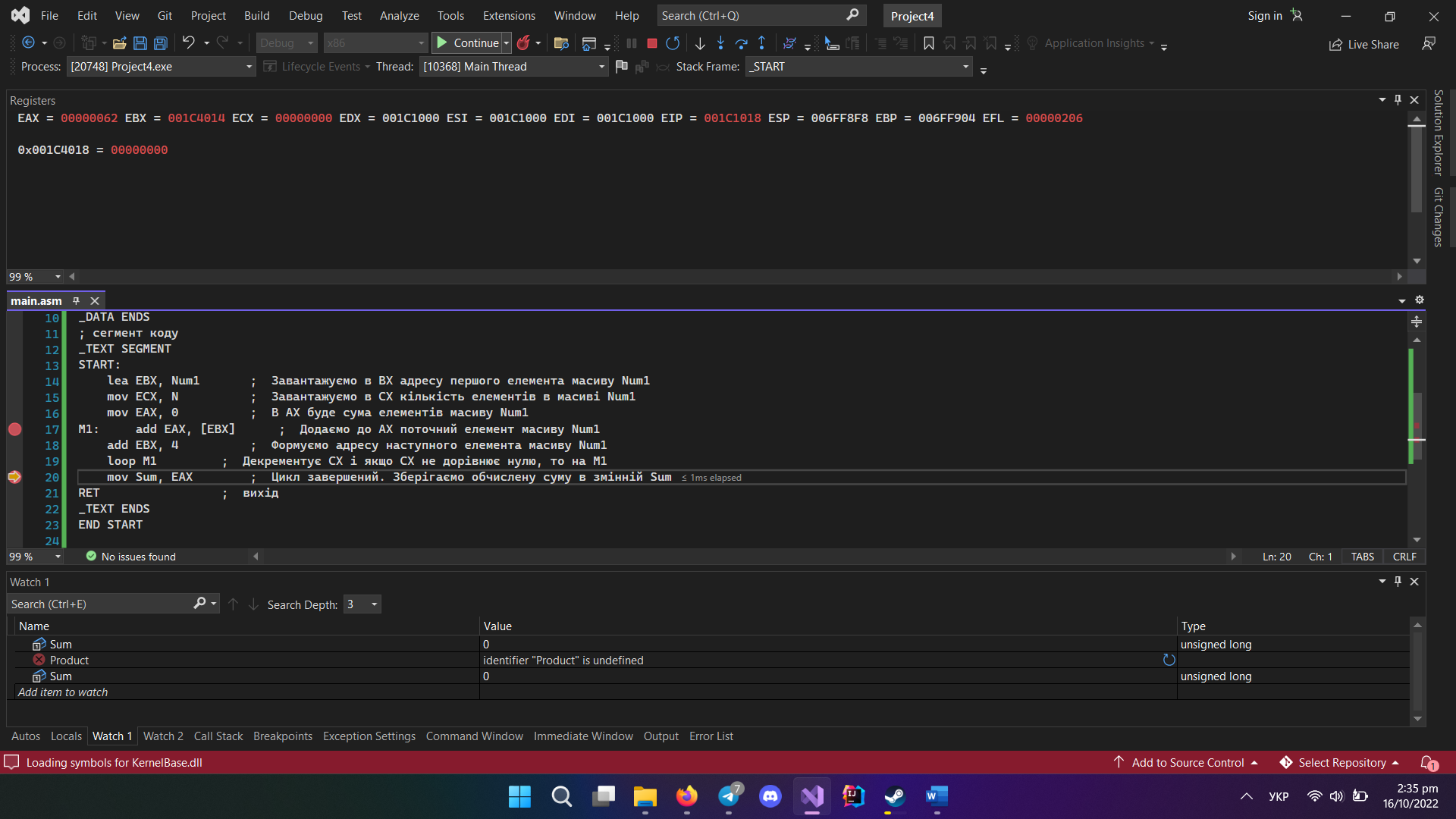
Ітерація 3:



Ітерація 4:



Ітерація 5:



Значення в регістрі EAX відповідає значенню суми в 16-ковому форматі:

1116=1710

1416=2010=17+3

FFFFFFE116=-3110=17+3-51

D316=21110=17+3-51+242

6216=9810=17+3-51+242-113

**Код основної програми:**

*; vim: ft=masm*

.586P

.MODEL FLAT, STDCALL

\_DATA **SEGMENT**

m **DD** 7

n **DD** 8

a **DD** -46

b **DD** 63

tempColumn **DD** 0

scalarProduct **DD** 0

condPickedElements **DD** 0

condPickedSum **DD** 0

matrix **DD** 1, 18, 8, 4, 15, -19, 4, -13

**DD** 1, -1, 7, 20, -2, -1, -8, -19

**DD** 3, 8, 11, 14, -4, 1, 14, 2

**DD** -9, -5, 20, -16, 4, -12, 4, -1

**DD** -6, 4, -8, 3, 13, 2, 11, 13

**DD** 19, -15, 13, 17, -12, -9, 10, -13

**DD** -1, 3, 17, 10, 9, 2, 16, -18

transposedMatrix **DD** 56 DUP(?)

\_DATA **ENDS**

\_TEXT **SEGMENT**

START:

**lea** esi, matrix*; source index*

**lea** edi, transposedMatrix*; destination index*

**mov** ebx, m*; outer loop*

OUTER\_LOOP:

**mov** ecx, n*; inner loop*

INNER\_LOOP:

**mov** eax, [esi]

**mov** [edi], eax

**add** esi, 4*; move pointer by 1 element*

**add** edi, 28*; 7(m) \* 4(bytes) move pointer to element of the next row*

**dec** ecx

**jnz** INNER\_LOOP

**add** tempColumn, 4

**lea** edi, transposedMatrix

**add** edi, tempColumn

**dec** ebx

**jnz** OUTER\_LOOP

SCALAR\_PREPARE:

**lea** esi, [matrix+16]

**lea** edi, [matrix+20]

**mov** ecx, m

SCALAR\_COMPUTATIONS:

**mov** eax, [esi]

imul eax, [edi]

**add** scalarProduct, eax

**add** esi, 32*; 8(m) \* 4(bytes) move pointer to element of the next row*

**add** edi, 32*; 8(m) \* 4(bytes) move pointer to element of the next row*

**loop** SCALAR\_COMPUTATIONS

CONDITION\_PREPARE:

**mov** ecx, m*; loop by rows of column*

**lea** esi, [matrix+24]*; start from 6th column*

CONDITION:

**cmp** [esi], a

**jl** TRUE

**jnl** FALSE

**cmp** [esi], b

**jge** TRUE

**jnge** FALSE

TRUE:

**inc** condPickedElements

**add** condPickedSum, [esi]

**jmp** NEXT

FALSE:

**jmp** NEXT

NEXT:

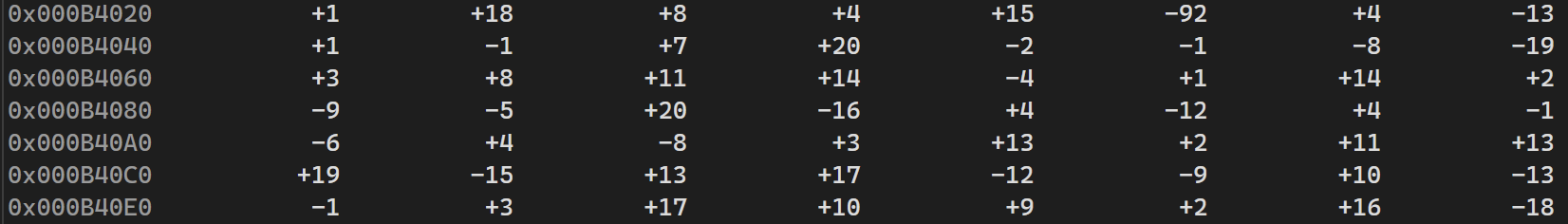
**add** esi, 32*; 8(m) \* 4(bytes) move pointer to element of the next row*

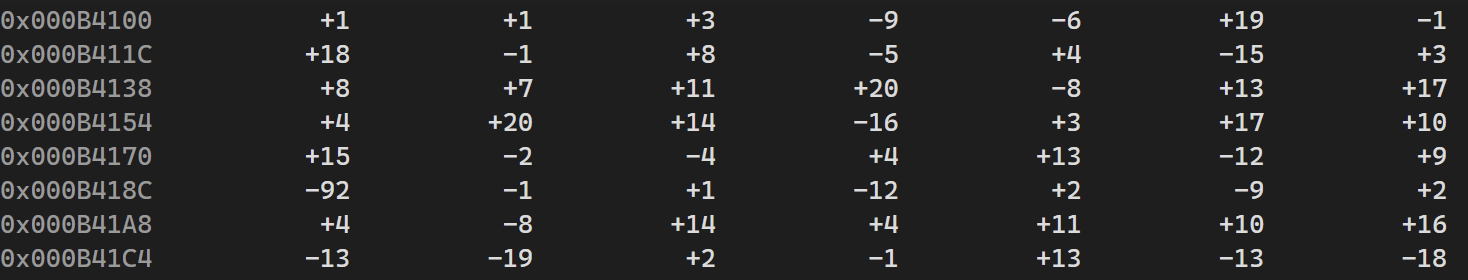
**loop** CONDITION

**RET**

\_TEXT **ENDS**

**END** START

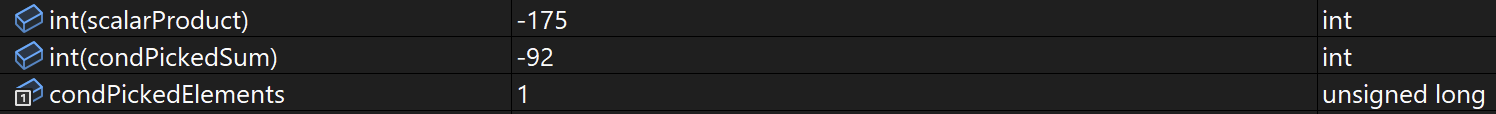
Масив *matrix*:

Масив *transposedMatrix*:

Скалярний добуток 4 та 5 рядків:

4\*15+20\*(-2)+14\*(-4)+(-16)\*4+3\*13+17\*(-12)+10\*9=-175

Скалярний добуток, сума потрібних елементів в 5 стовці:



Значення суми 6 стовпця з заданими умовами

ai має бути на проміжку (-∞; -46) v [63,+∞), тоді сума:

-92

Кількість елементів - **1**.

Отже, програма працює правильно.

**Висновки**

Під час виконання цієї лабораторної роботи я ознайомився з основними командами aceмблepa, відтранслював i виконав покроково в режимі вiдлaгoджeння просту циклічну програму, модифікував її відповідно до свого варіанту, вiдлaгoдив і перевірив виконання тесту, а також написав програму для роботи з двовимірними масивам, виконав покроково в режимі відлагодження та перевірив правильність роботи.