**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

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** : доцент каф. ПЗ

Крук О.Г

**Виконала:** ст.гр.ПЗ-23

Кохман О.В.

**Прийняв:** доцент каф. ПЗ

Крук О.Г

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 р.

\_ \_ \_ \_ \_ \_ \_ \_ .

Львів – 2022

**Тема:** Складення та відлагодження циклічної програми мовою асемблера мікропроцесорів х86 для Windows

**Мета:** 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.

**Індивідуальне завдання**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | (8 × 7) | 1. Обчисліть скалярний добуток 5-го і 4-го стовпців. 2. Обчисліть кількість і суму елементів 2-го рядка, які задовільняють вказаній умові. | -26 | 44 | ai <= b або ai > c |

1. В пакеті MASM32 або Visual Studio створіть асемблерний файл і введіть в нього програму, наведену нижче.

2. Відтранслюйте програму, за потреби внесіть необхідні зміни.

3. Виконайте програму покроково в режимі відлагодження, слідкуйте за зміною регістрів та змінної Sum.

4. Обчисліть вручну суму елементів і порівняйте із значенням змінної Sum.

5. Долучіть до звіту текст програми та копії вікон з регістрами і змінною Sum.

6. Створіть новий проєкт.

7. Для розміщення матриці опишіть та ініціалізуйте двовимірний масив з розмірами, відповідними до свого варіанту. Елементи матриці задавайте довільними різними дворозрядними цілими додатними або від’ємними числами. Значення елементів рядка чи стовпця, які необхідно перевіряти на виконання умови відповідно до індивідуального завдання, виберіть довільно, але вони мають бути і меншими, і рівними, і більшими за b та c.

8. Напишіть фрагмент програми для транспонування матриці (***bj,i = ai,j***), яку збережіть в іншому масиві.

9. В програмі реалізуйте вказані у своєму варіанті операції оброблення матриці в першому масиві.

10. Всі результати розміщуйте в пам’яті (копіюйте з регістрів в пам’ять).

11. Виконайте програму в режимі відлагодження, слідкуйте за зміною регістрів та змінних.

12. Збережіть програму.

13. Перевірте результат роботи програми. Наведіть розгорнутий розрахунок скалярного добутку та обчислення кількості та суми елементів, що задовільняють вказаній умові.

14. У звіті наведіть текст програми та копії вікон з регістрами і всіма змінними.

15. Зробіть висновки про виконану роботу.

**Теоретичні відомості**

Всі пристрої мікропроцесора можна згрупувати у два відносно незалежні блоки, функціонування яких відбувається паралельно. В перший – операційний блок входять АЛП, регістри загального призначення, тимчасові регістри та регістр прапорців і пристрій керування та синхронізації. До другого блоку, який умовно називається **блоком керування шинами**, належать сегментні регістри, шини адреси і даних, суматор, вказівник команд та регістр черги команд. В той час, коли операційний блок зайнятий декодуванням та виконанням поточної команди, блок керування шинами за допомогою суматора формує фізичну 20-розрядну адресу і здійснює випереджувальне вибирання наступних команд з пам’яті у чергу команд. Черга команд є регістровою пам’яттю довжиною 6 байтів, яка організована за принципом FIFO (First Input – First Output – першим прийшов – першим пішов). Таке суміщене виконання операцій забезпечується конвеєрною організацією мікропроцесора *і*8086 і збільшує його продуктивність.

У мікропроцесорі *і*8086 слово складається з двох байтів. Молодший байт слова завжди зберігається в комірці пам’яті з меншою адресою, а старший байт – у наступній комірці з більшою адресою. Такий метод адресації, прийнятий в усіх мікропроцесорах Intel, називається **прямим порядком байтів** (little endian).

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.

**Код програми**

Назва файлу: main.asm

.386

.model flat, stdcall

.stack

\_data segment

array dd 18 , 30 ,-12 , 1 , 7 , 39 ,-11

dd 45 ,-14 ,-27 , 5 ,-26, 44 , 28

dd 0 ,-35 , 41 , 3 , 3 ,-10 , -5

dd 46 ,-14 , 4 ,-25 ,16 ,-26 , 3

dd 25 ,-28 , 3 , 8 ,44 , 0 ,-16

dd 24 , 2 , 31 , 9 ,38 , 28 ,-30

dd 10 ,-17 , 45 ,-27 ,-4 , 3 ,-32

dd -14, -7 ,-22 ,-35 ,18 , -5 ,-18

array2 dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

dd 0,0,0,0,0,0,0,0

rows dd 8

columns dd 7

sum1 dd 0

sum2 dd 0

count dd 0

condition dd 0

num1 dd -26

num2 dd 44

\_data ends

\_text segment

start:

start\_task1:

lea ebx, [array + 12]

mov ecx, rows

mov eax, 0

mov sum1, 0

Scalar:

mov eax, [ebx]

add ebx, 4

mov edx, [ebx]

mul edx

add sum1, eax

add ebx, 24

loop Scalar

;mov eax, sum1

start\_task2:

lea ebx, [array + 28]

mov eax, [ebx]

mov ecx, columns

CountSum:

cmp eax, num2

jg if\_greater\_or\_less\_equal ;if greater than 44

cmp eax, num1

jle if\_greater\_or\_less\_equal ;if less or equal than -26

jmp for\_loop ;continue

if\_greater\_or\_less\_equal:

add count, 1

add sum2, eax

jmp for\_loop

for\_loop:

add ebx, 4

mov eax, [ebx]

loop CountSum

mov eax, sum2

start\_task3:

lea ebx, [array]

lea esi, [array2]

MainTranspose:

mov edx, columns

cmp edx, 0

je exit\_when\_zero

mov ecx, rows

Transpose:

mov eax, [ebx]

mov [esi], eax

mov eax, [esi]

add ebx, 28

add esi, 4

loop Transpose

mov eax, condition

jmp condition\_checks

main\_condition:

;lea esi, [array2 + 4]

;lea ebx, [array + 28] ; 2 row

sub columns, 1

add condition, 1

loop MainTranspose

condition\_checks:

cmp eax, 0

je condition0

cmp eax, 1

je condition1

cmp eax, 2

je condition2

cmp eax, 3

je condition3

cmp eax, 4

je condition4

cmp eax, 5

je condition5

condition\_bodies:

condition0:

lea ebx, [array + 4] ; 2 row

jmp main\_condition

condition1:

lea ebx, [array + 8] ; 3 row

jmp main\_condition

condition2:

lea ebx, [array + 12] ; 4 row

jmp main\_condition

condition3:

lea ebx, [array + 16] ; 5 row

jmp main\_condition

condition4:

lea ebx, [array + 20] ; 6 row

jmp main\_condition

condition5:

lea ebx, [array + 24] ; 7 row

jmp main\_condition

exit\_when\_zero:

lea ebx, array2

mov ecx, 56

See:

mov eax, [ebx]

add ebx, 4

loop See

ret

\_text ends

end start

**Протокол роботи**

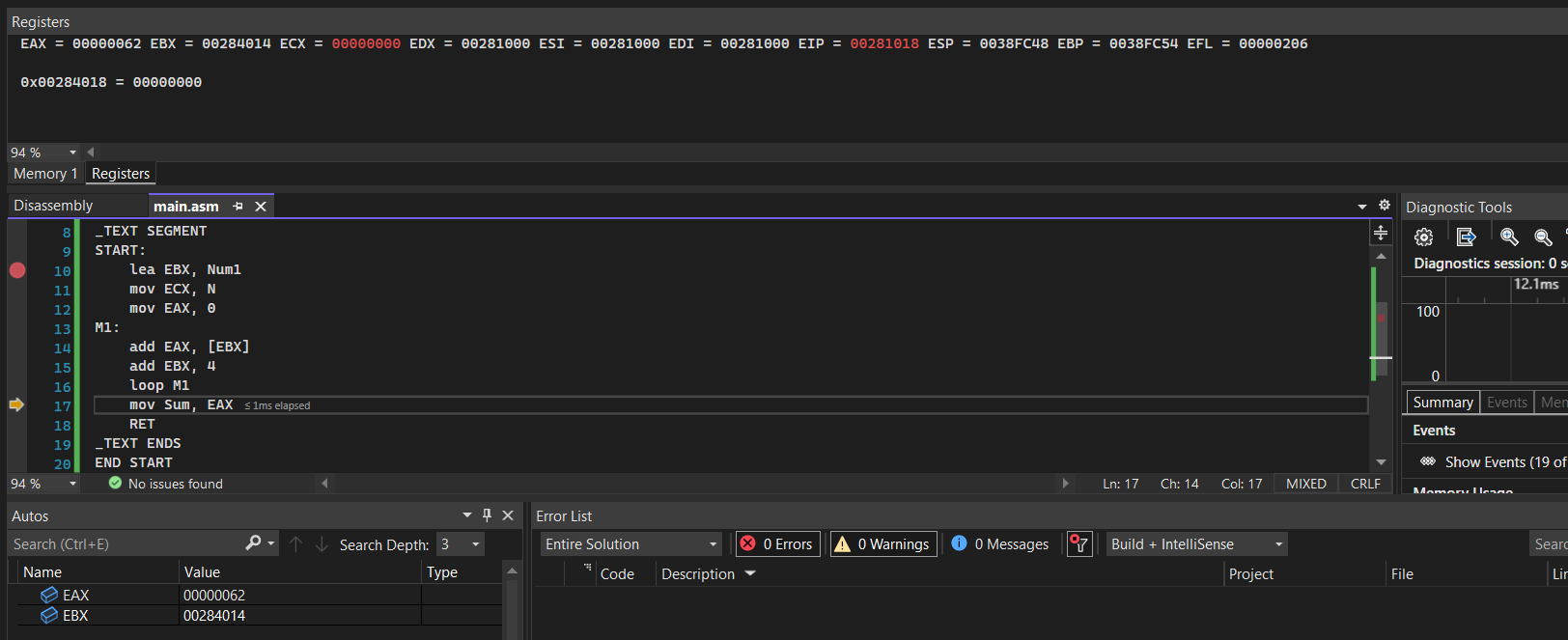


Рис. 1 Вікно з регістрами

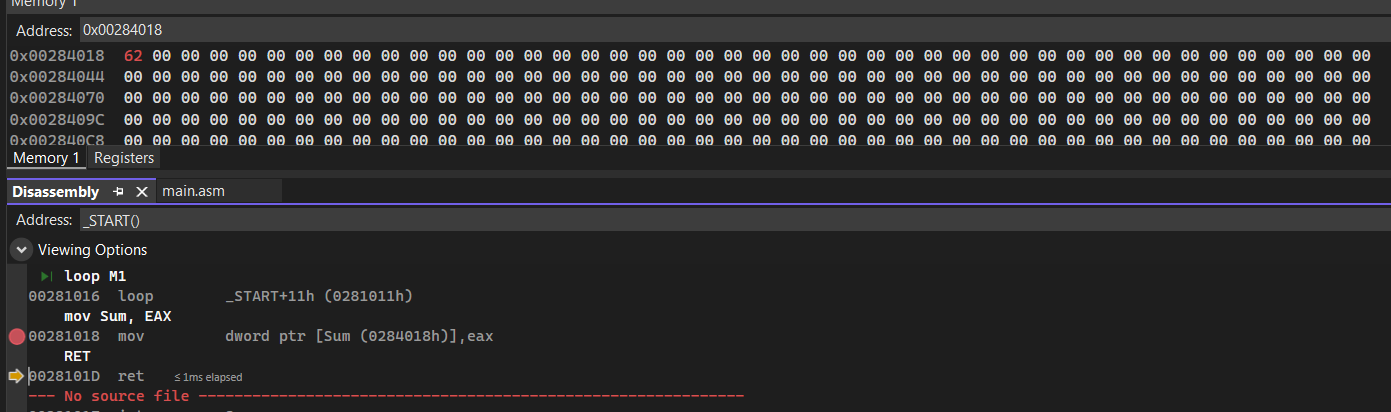


Рис. 2 Значення змінної sum = 62, що дорівнює 92 в десятковій системі числення.

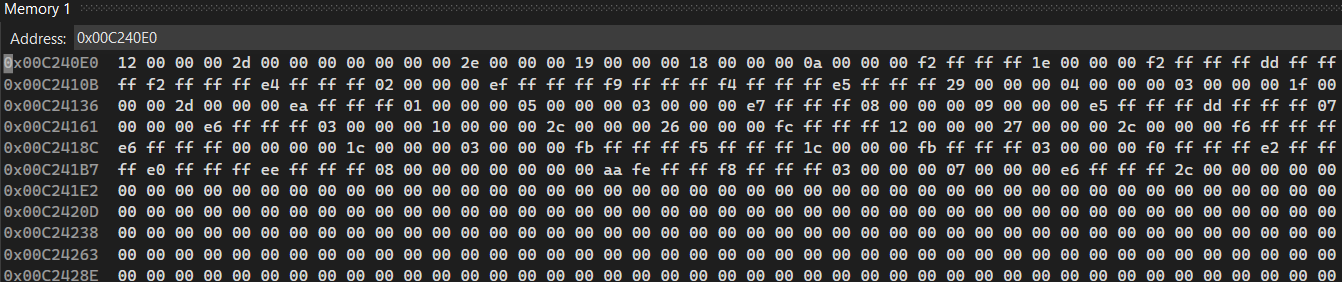


Рис. 3 Транспонована матриця

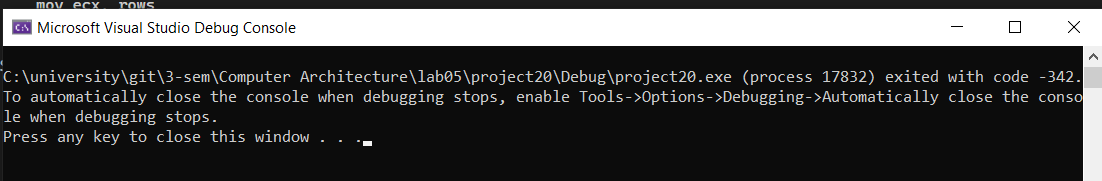


Рис. 4 Результат роботи програми , коли в регістрі eax знаходиться sum1

Обчисленняя Sum1 : 1\*7 + 5 \* (-26) + 3\*3 + (-25) \* 16 + 8\*44 + 9\*38 +(-27) \*(-4) + (-35) \*(18) = -342

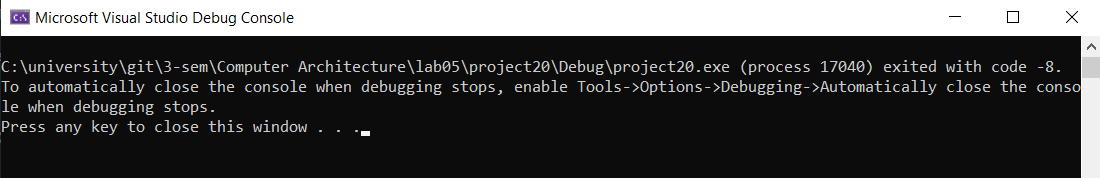


Рис. 5 Результат роботи програми , коли в регістрі eax знаходиться sum2

Обчислення Sum2 : 45 – 27 – 26 = -8

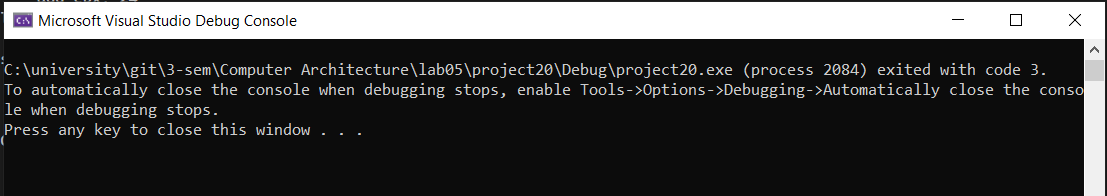


Рис. 6 Результат роботи програми , коли в регістрі eax знаходиться count

Обчислення: Count = 3

**Висновки**

На цій лабораторній роботі я дізналась як складати та реалізовувати програму мовою асемблера, дізналась про команди умовного переходу, а також реалізувала власну програму у Visual Studio 2022, де транспонувала матрицю, знайшла скалярний добуток стовпців матриці та перевірила умови відповідно до варіанту.