

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

ІКНІ
Кафедра ПЗ



ЗВІТ

До лабораторної роботи №4

на тему: “Складення та відлагодження циклічної програми мовою асемблера
мікропроцесорів x86 для Windows”

з дисципліни: “Архітектура комп’ютера”

Лектор:
доцент кафедри ПЗ
Крук О.Г.

Виконав:
студент групи ПЗ-24
Губик А. С.

Прийняв:
доцент кафедри ПЗ
Задорожний І. М.

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 навички складання програми з вкладеними циклами; відтранслювати і виконати в режимі відлагодження програму, складену відповідно до свого варіанту; перевірити виконання тесту.

Індивідуальне завдання

Варіант	Розмір матриці	Операції оброблення матриці	b	c	Умова
3	(9 x 6)		-51	82	$b \leq a_i \leq c$

Теоретичні відомості

У програмну модель входять 8-, 16- та 32-бітові регістри (рис. 1). До 8-бітових належать AH, AL, BH, BL, CH, CL, DH, DL. Вони вказуються в командах асемблера як операнди і дозволяють працювати лише з 8-бітовою інформацією. До 16-бітових регістрів належать AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS та GS. Ці регістри дозволяють працювати відповідно тільки з 16-бітовою інформацією. Всі розширені (32-бітові) регістри починаються з букви E: EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP та EFLAGS. Вони, а також 16-бітові регістри FS та GS реалізовані в мікропроцесорах, починаючи з 80386. Регістри загального призначення 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-десятьова 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 (source index – вказівник джерела) адресується як ESI та SI, у рядкових інструкціях адресує операнд-джерело. Спеціалізовані регістри До спеціалізованих регістрів належать регістри EIP, ESP, EFLAGS, а також сегментні регістри – CS, DS, ES, SS, FS та GS. EIP (instruction pointer – вказівник інструкції) адресує наступну інструкцію (яка буде виконуватись після поточної) в області пам'яті, визначеній як сегмент коду. В реальному режимі використовується 16-бітовий регістр IP. 32-бітовий регістр EIP використовується в захищеному режимі процесорів 80386 – Pentium 4. Вказівник інструкції може бути змінений командою переходу або виклику підпрограми. ESP (stack pointer – вказівник стека) адресує область пам'яті, визначену як стек. У реальному режимі використовується 16-бітовий регістр SP. 32-бітовий регістр ESP використовується в захищеному режимі процесорів 80386 – Pentium . (register).

Хід роботи

1. Перша програма

```
.586p
.model flat, stdcall

_DATA SEGMENT
Num1 DD 17, 3, -51, 242, -113

N DD 5
Sum DD 0
_DATA ENDS

_TEXT SEGMENT
START:
lea EBX, Num1
mov ECX, N
mov EAX, 0
M1: add EAX, [EBX]
add EBX, 4
loop M1
mov Sum, EAX
xor eax, eax

RET

_TEXT ENDS
END START
```

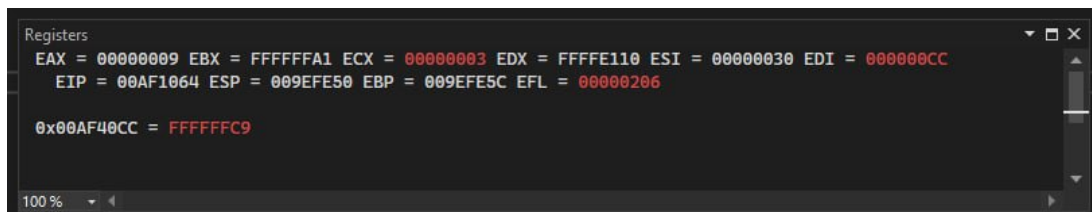


Рис. 1: Транспонування матриці

2. Друга програма

```
.586p
.model flat, stdcall
_data segment
matrix dd -17, 72, 11, 63, 36, 95
dd -36, -99, 67, -82, -70, 39
dd -89, 48, 90, 75, 14, -8
dd 86, 51, 37, 80, 59, 20
dd -68, 44, -1, 84, 25, 45
dd -92, 62, 60, -31, 78, 15
dd -56, -20, -69, -49, -58, 19
```

```

dd -65, -38, -30, 93, 10, 29
dd 9, -85, -95, -55, 57, -97

```

```

colCount equ 6
rowCount equ 9

```

```

transposedMatrix dd 6*9 dup(?) ; Transposed matrix

```

```

result DWORD ? ; Result of dot product
sum9throw DWORD ? ; Sum of the 9th row elements

```

```

_data ends

```

```

_text segment

```

```

start:

```

```

;mov ecx, 9 ; Number of rows
;mov ebx, 6 ; Number of columns

```

```

; Transpose the matrix

```

```

mov esi, 0 ; Row index

```

```

transpose_loop:

```

```

mov edi, 0 ; Column index

```

```

transpose_column_loop:

```

```

; Copy elements [matrix + esi*4*colCount + edi*4] to [transposedMatrix + edi*4 + esi*colCount]

```

```

imul ebx, esi, 4*colCount

```

```

imul edx, edi, 4*rowCount

```

```

mov eax, [matrix + ebx + edi*4]

```

```

mov [transposedMatrix + edx + esi*4], eax

```

```

inc edi

```

```

cmp edi, colCount

```

```

jnl transpose_column_loop

```

```

inc esi

```

```

cmp esi, rowCount

```

```

jnl transpose_loop

```

```

; Calculate the dot product of the first and third columns

```

```

mov edx, 0 ; Initialize dot product result to 0

```

```

mov edi, 0 ; Column index

```

```

mov esi, 0

```

```

dot_product_loop:

```

```

imul esi, edi, colCount

```

```

mov eax, [matrix + esi*4] ; First column

```

```

mov ebx, [matrix + 2*4 + esi*4] ; Third column

```

```

imul eax, ebx

```

```

add edx, eax

```

```

inc edi

```

```

cmp edi, rowCount

```

```

jnl dot_product_loop

```

```

; Calculate the sum of the 9th row elements that are >= -51 and <= 82

```

```

mov edi, 8*6*4 ; Start at the end of the 9th row
mov eax, 0      ; Initialize sum to 0
mov ecx, 6
sum_9th_row_loop:
cmp [matrix + edi], -51
jl skip_element
cmp [matrix + edi], 82
jg skip_element
add eax, [matrix + edi]
skip_element:
add edi, 4
loop sum_9th_row_loop

; Store the results
mov [result], edx ; Dot product result
mov [sum9throw], eax ; Sum of the 9th row elemen

ret
_text ends
end start

```

Memory 1									
Address: 0x00AF4000									
0x00AF4000	-17	+72	+11	+63	+36	+95	п'яятий.....?...\$.....		
0x00AF4018	-36	-99	+67	-82	-70	+39	б'яятийC...0...яяяяяя'...		
0x00AF4030	-89	+48	+90	+75	+14	-8	\$яяя0...Z...K...яяяяяя...		
0x00AF4048	+86	+51	+37	+80	+59	+20	V...3...%...P...яяяяяя...		
0x00AF4060	-68	+44	-1	+84	+25	+45	jяяя,...яяяяT...яяяяяя...		
0x00AF4078	-92	+62	+60	-31	+78	+15	м'яяя>...<...б'яяяN...яяяя...		
0x00AF4090	-56	-20	-69	-49	-58	+19	Ияяям'яяя»яяяП'яяяИяяя...		
0x00AF40A8	-65	-38	-30	+93	+10	+29	Iяяяb'яяяяяяяяя]...яяяяяя...		
0x00AF40C0	+9	-85	-95	-55	+57	-97	...«яяяУяяяяИяяя9...ц'яяяя...		

Memory 2									
Address: 0x00AF40D8									
0x00AF40D8	-17	-36	-89	+86	-68	-92	-56	-65	+9 п'яятий
0x00AF40FC	+72	-99	+48	+51	+44	+62	-20	-38	-85 Н...
0x00AF4120	+11	+67	+90	+37	-1	+60	-69	-30	-95 ...
0x00AF4144	+63	-82	+75	+80	+84	-31	-49	+93	-55 ?...
0x00AF4168	+36	-70	+14	+59	+25	+78	-58	+10	+57 \$...
0x00AF418C	+95	+39	-8	+20	+45	+15	+19	+29	-97 ...

Рис. 2: Транспонування матриці



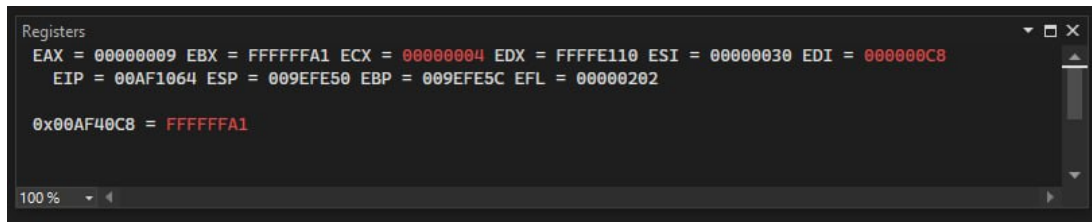
Name	Value
 result	0xffffe110
 sum9throw	0x00000042
Add item to watch	

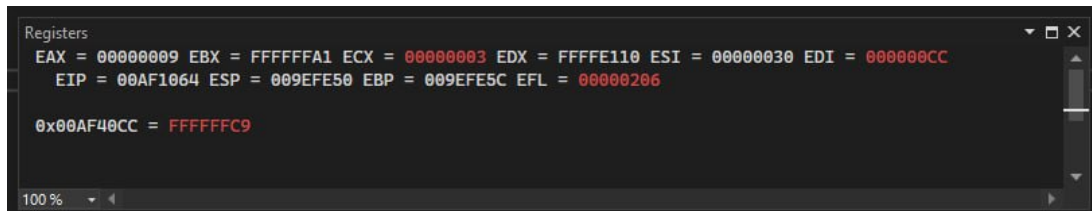
Рис. 3: Змінні



```
Registers
EAX = 00000009 EBX = FFFFFFFA1 ECX = 00000004 EDX = FFFE110 ESI = 00000030 EDI = 000000C8
EIP = 00AF1064 ESP = 009EFE50 EBP = 009EFE5C EFL = 00000202

0x00AF40C8 = FFFFFFFA1
```

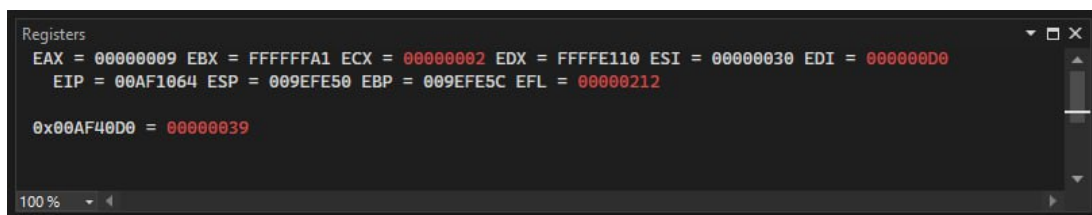
Рис. 4: Перша ітерація



```
Registers
EAX = 00000009 EBX = FFFFFFFA1 ECX = 00000003 EDX = FFFE110 ESI = 00000030 EDI = 000000CC
EIP = 00AF1064 ESP = 009EFE50 EBP = 009EFE5C EFL = 00000206

0x00AF40CC = FFFFFFFC9
```

Рис. 5: Друга ітерація



```
Registers
EAX = 00000009 EBX = FFFFFFFA1 ECX = 00000002 EDX = FFFE110 ESI = 00000030 EDI = 000000D0
EIP = 00AF1064 ESP = 009EFE50 EBP = 009EFE5C EFL = 00000212

0x00AF40D0 = 00000039
```

Рис. 6: Третя ітерація

Висновок

Я розібрався з базовим функціоналом асемблеру і як використовувати Visual Studio для його відлагодження.