

**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 x86 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.

$\Sigma =$ _____

Тема: складення та відлагодження циклічної програми мовою асемблера мікропроцесорів x86 для Windows.

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

22	(5 × 8)	1. Обчисліть скалярний добуток 5-го і 4-го рядків. 2. Обчисліть кількість і суму елементів 5-го стовпця, які задовільняють вказаній умові.	-4 6	63	$a_i < b$ або $a_i \geq c$
----	---------	--	---------	----	----------------------------

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-десятькова 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.

Інструкції регістрової адресації

Інструкція	Розмірність	Дія
MOV AL, BL	Байт	Копіює BL в AL
MOV CH, CL	Байт	Копіює CL в CH
MOV AX, CX	Слово	Копіює CX в AX
MOV SP, BP	Слово	Копіює BP в SP
MOV DS, AX	Слово	Копіює AX в DS
MOV SI, DI	Слово	Копіює DI в SI
MOV BX, ES	Слово	Копіює ES в BX
MOV ECX, EBX	Подвійне	Копіює EBX в ECX
MOV ESP, EDX	Подвійне	Копіює EDX в ESP
MOV ES, DS	-	Недопустима інструкція - копіювання сегментного реєстра в сегментний реєстр заборонено
MOV BL, DX	-	Інструкція недопустима - операнди мають різну розмірність
MOV CS, AX	-	Недопустима інструкція - сегментний реєстр коду не може бути приймачем

Інструкції прямої адресації

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

Інструкції непрямої адресації

Інструкція	Розмірність	Дія
------------	-------------	-----

[BX]	MOV CX,	Слово	Копіює в CX слово, розташоване в сегменті даних за зміщенням, заданим в BX
DL*	MOV [BP],	Байт	Копіює DL в сегмент стека за зміщенням, заданим в BP
	MOV [DI], BH	Байт	Копіює BH в сегмент даних за зміщенням, заданим в DI
[BX]	MOV [DI],	Байт	Помилка - передача даних між комірками пам'яті підтримується тільки для рядкових інструкцій
	MOV AL,	Подвій	Копіює в AL байт з сегмента даних, зміщення якого задано регістром EDX
[EDX]	MOV ECX,	не слово	Копіює в ECX подвійне слово з сегмента даних, зміщення якого задано в EBX
[EBX]			

Інструкції умовного переходу

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

Індивідуальне завдання

22	(5 × 8)	1. Обчисліть скалярний добуток 5-го і 4-го рядків. 2. Обчисліть кількість і суму елементів 5-го стовпця, які задовільняють вказаній умові.	-4 6	63	$a_i < b$ або $a_i \geq c$
----	---------	--	---------	----	----------------------------

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 M1
    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:

EAX = 00000000

Ітерація 1:

EAX = 00000011

Ітерація 2:

EAX = 00000014

Ітерація 3:

EAX = FFFFFFFE1

Ітерація 4:

EAX = 000000D3

Ітерація 5:

EAX = 00000062

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

$$11_{16}=17_{10}$$

$$14_{16}=20_{10}=17+3$$

$$\text{FFFFFFE1}_{16}=-31_{10}=17+3-51$$

$$\text{D3}_{16}=211_{10}=17+3-51+242$$

$$62_{16}=98_{10}=17+3-51+242-113$$

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

```

Массив *matrix*:

0x000B4020	+1	+18	+8	+4	+15	-92	+4	-13
0x000B4040	+1	-1	+7	+20	-2	-1	-8	-19
0x000B4060	+3	+8	+11	+14	-4	+1	+14	+2
0x000B4080	-9	-5	+20	-16	+4	-12	+4	-1
0x000B40A0	-6	+4	-8	+3	+13	+2	+11	+13
0x000B40C0	+19	-15	+13	+17	-12	-9	+10	-13
0x000B40E0	-1	+3	+17	+10	+9	+2	+16	-18




Массив *transposedMatrix*:

0x000B4100	+1	+1	+3	-9	-6	+19	-1
0x000B411C	+18	-1	+8	-5	+4	-15	+3
0x000B4138	+8	+7	+11	+20	-8	+13	+17
0x000B4154	+4	+20	+14	-16	+3	+17	+10
0x000B4170	+15	-2	-4	+4	+13	-12	+9
0x000B418C	-92	-1	+1	-12	+2	-9	+2
0x000B41A8	+4	-8	+14	+4	+11	+10	+16
0x000B41C4	-13	-19	+2	-1	+13	-13	-18

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:

 int(scalarProduct)	-175	int
 int(condPickedSum)	-92	int
 condPickedElements	1	unsigned long

Значення суми 6 стовпця з заданими умовами

a_i має бути на проміжку $(-\infty; -46) \cup [63, +\infty)$, тоді сума:

-92

Кількість елементів - 1.

Отже, програма працює правильно.

Висновки

Під час виконання цієї лабораторної роботи я ознайомився з основними командами асемблера, відтранслявав і виконав покроково в режимі відлагодження просту циклічну програму, модифікував її відповідно до свого варіанту, відлагодив і перевінив виконання тесту, а також написав програму для роботи з двовимірними масивом, виконав покроково в режимі відлагодження та перевінив правильність роботи.