

**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

Ясногородський

Н.В.

Прийняв:

доц. каф. ПЗ

Крук О.Г.

« _____ » _____ 2022 р.
Σ = _____

Львів – 2022

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: 30.

30	(7 × 6)	1. Обчисліть скалярний добуток 7-го і 6-го рядків. 2. Обчисліть кількість і суму елементів 3-го стовпця, які задовільняють вказаній умові.	-2 2	81	$a_i < b$ 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
M O V ECX,EBX	П о д в і й н е слово	Копіює EBX в ECX
M O V EDX,ESP	П о д в і й н е слово	Копіює ESP в EDX
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 слово	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 слово	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 слово	Копіює BP в сегмент даних за зміщенням DATA7
MOV NUMBER1, SP	Подвійне слово	Копіює SP в сегмент даних за зміщенням NUMBER
MOV DATA1, EAX	Подвійне слово	Копіює EAX в сегмент даних за зміщенням DATA1
MOV EDI, SUM1	Подвійне слово	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

30	(7 × 6)	1. Обчисліть скалярний добуток 7-го і 6-го рядків. 2. Обчисліть кількість і суму елементів 3-го стовпця, які задовільняють вказаній умові.	-2 2	81	$a_i < b$ 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 реєстрі 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 основной программы:

```
.586P
.MODEL FLAT, STDCALL
_DATA SEGMENT
b                DD -22
ci               DD 81
m               DD 7
n               DD 6
matrix           DD 15, -24, 45, 58, -60, 83
                DD 12, -17, 22, 27, -37, 42
                DD 24, -28, -52, 62, -66, 69
                DD 19, -25, 27, -31, 52, 56
                DD 14, -75, 16, -39, 29, -35
                DD 17, -21, 99, -40, 53, 70
                DD 19, -25, 27, -31, 52, 56
matrixTranposed  DD 42 DUP(0)
tmpCol          DD 0
scalarMult      DD 0
condCount       DD 0
condSum         DD 0
_DATA ENDS

_TEXT SEGMENT
START:
    lea ESI, matrix ; ESI - source index
    lea EDI, matrixTranposed ; EDI - destination index
    mov EBP, m ; outer loop

OUTER_LOOP:
    mov ECX, n ; inner loop

INNER_LOOP:
    mov EAX, [ESI]
    mov [EDI], EAX
    add ESI, 4 ; move matrix element pointer
    add EDI, 28 ; 7(m)*4(bytes) move pointer to the next cell in column
    dec ECX
    jnz INNER_LOOP;

    add tmpCol, 4 ; move to the next cell in row
    lea EDI, matrixTranposed
    add EDI, tmpCol
    dec EBP
    jnz OUTER_LOOP

SCALAR_PREPARE:
    lea EBX, matrix
    add EBX, 120 ; 5*6*4 = 120 (move offset to first element in 6th row)
    mov ECX, n

SCALAR:
    mov EAX, [EBX]
    imul EAX, [EBX + 24] ; 6*4 = 24 (matching cell of 6th row from 7th row)
    add EBX, 4
    add scalarMult, EAX
    loop SCALAR

ROW_SUM:
    lea EBX, matrix
    add EBX, 8 ; 2*4 (first element in 3rd column)
    mov ECX, n

CONDITION:
    mov EAX, [EBX]
    cmp EAX, b
    jl TRUE ; ai < b
    cmp EAX, ci
    jge TRUE ; ai >= c
    jmp FALSE

TRUE:
    inc condCount
    add condSum, EAX

FALSE:
    add EBX, 6*4 ; move to the next row
    loop CONDITION

RET
_TEXT ENDS
END START
```

Масив matrix:

Address:	0x009B4010						
0x009B4010	+15	-24	+45	+58	-60	+83	.. .-.: .S.
0x009B4028	+12	-17	+22	+27	-37	+42*.
0x009B4040	+24	-28	-52	+62	-66	+69> .E.
0x009B4058	+19	-25	+27	-31	+52	+564.8.
0x009B4070	+14	-75	+16	-39	+29	-35	..^.....
0x009B4088	+17	-21	+99	-40	+53	+70	..+.c...5.F.
0x009B40A0	+19	-25	+27	-31	+52	+564.8.




Масив matrixTransposed:

Address:	0x009B40B8						
0x009B40B8	+15	+12	+24	+19	+14	+17	+19
0x009B40D4	-24	-17	-28	-25	-75	-21	-25
0x009B40F0	+45	+22	-52	+27	+16	+99	+27
0x009B410C	+58	+27	+62	-31	-39	-40	-31
0x009B4128	-60	-37	-66	+52	+29	+53	+52
0x009B4144	+83	+42	+69	+56	-35	+70	+56

Скалярний добуток 6 та 7 рядків:

$$17*19+(-21)*(-25)+27*99+(-40)*(-31)+53*52+70*56=11437$$

Скалярний добуток, сума потрібних елементів в 5 стовці:

 int(condSum)	47
 condCount	2
 scalarMult	11437

Значення суми 5 стовпця з заданими умовами

a_i має бути на проміжку $(-\infty; -22) \cup [81, +\infty)$, тоді сума:

$$-52 + 99 = 47$$

Кількість елементів - 2.

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 перевірів виконання тесту, а також написав програму для роботи з двовимірними масивам, виконав покроково в режимі відлагодження та перевірів правильність роботи.