**Лабораторна робота №4**

**Команди передачі керування та цикли**

**Ціль роботи:** навчитися розробляти програми на мові Асемблер з використанням команд передачі керування та організації циклів при обробці інформації.

**Короткі теоретичні відомості**

Команди передачі керування дозволяють змінити послідовність проходження виконання команд на основі аналізу деяких умов або даних, а також при необхідності перейти в іншу точку програми. Вони змінюють вміст регістрів CS та IP, у результаті чого мікропроцесор вибирає для виконання команду в іншій ділянці програми.

Адреса переходу представляється у вигляді мітки або адреси в області пам'яті. Існує декілька різновидносте команди, які визначають дальність переходу й способи завдання цільової адреси.

Якщо необхідний адрес переходу знаходиться в поточному сегменті коду, то такий перехід називається **внутрісегментним** або *близьким*, і для зміни адреси необхідно змінюється тільки вміст регістра IP. Як модифікатор для уточнення переходу застосовується оператор перевизначення типу ptr, який застосовується для уточнення типу мітки або змінної:

**<тип > ptr <вираз>,**

де значення <тип> може приймати одне з наступних значень: byte, word, dword, qword, tbyte, near, far.

Прямий внутрісегментний перехід представляється трибайтною командою, в якій поле адреси містить два байти, й перехід може здійснюватися в межах сегмента, тобто в межах 64 кбайт.

Можна виділити 4 основні групи команд передачі керування:

- безумовної передачі керування;

- виклику процедури й повернення із процедури;

- умовного переходу;

- керування циклами

**Команди безумовної передачі керування**

Для зміни послідовності виконання програми служить команда JMP, яка дозволяє перейти на задану адресу без збереження інформації про точку повернення.

Після виконання заданого фрагмента програми повернення у вихідну точку не відбувається.

Команда JMP за замовчуванням формує прямий внутрісегментний перехід і може мати такий вигляд:

**JMP адреса\_переходу**

**JMP near ptr адреса\_переходу**

Непрямий перехід на мітку всередині поточного сегмента коду задається за допомогою модифікатора word ptr:

**JMP word ptr адреса\_переходу**

При цьому в регістр ІР записується 16-розрядна адреса, яка знаходиться по вказаній в адресі переходу команді. Перехід програми до іншого сегменту пам'яті називається міжсегментним або далеким. Адреса далекого переходу складається з 16-бітного селектора й 16-бітного зміщення, які завантажуються, відповідно, у регістри CS та IP:

**JMP far ptr адреса\_переходу**

Непрямий перехід на мітку в іншому сегменті коду здійснюється за допомогою модифікатора dword ptr. Модифікуються регістри CS та IP значенням тільки з пам'яті. Перше слово цієї адреси представляє зсув і завантажується в ip; друге слово завантажується в cs:

JMP dword ptr адреса\_переходу

Команди умовної передачі керування

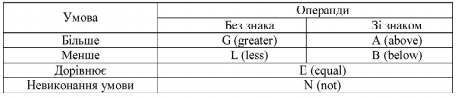
Команди умовної передачі керування дозволяють здійснити перехід на виконання іншого фрагменту програми при виконанні певних умов. Умови переходу можуть визначатися за вмістом прапорів, регістра CX; за результатами виконання команди порівняння CMP.

Усі команди умовного переходу мають однакову структуру:

**J<cc> мітка\_переходу,**

Мітка\_переходу може перебувати тільки в межах поточного сегмента коду.

Умови переходів задаються за допомогою позначень



**Організація виклику процедур**

Одним з важливих методів зміни послідовності виконання програми є виклик процедури **CALL** й повернення із процедури **RET**. Особливість організації програми з процедурою полягає в тому, що механізм виклику процедури дозволяє запам'ятати інформацію про стан програми в точці виклику і потім повернутися в цю точку після завершення її роботи. Команда виклику процедури має наступну будову:

**call [модифікатор] ім'я\_процедури,**

де модифікатор може бути near або far.

Команда CALL перериває виконання програми і передає управління за адресою викликові, але при цьому заносить у стек адресу повернення.

Адреса повернення містить адресу команди, яка знаходиться за командою CALL. Розрізняють передачу внутрішньосегментну та міжсегментну. При внутрішньосегментному переході процедура, яку викликаємо, знаходиться в даному сегменті, і для повернення потрібно запам'ятати тільки стан регістра команд IP. Команди внутрішньо сегментного переходу мають модифікатор near, який може не вказуватися. При організації між сегментного переходу обов'язково потрібно вказувати модифітатор far, а для організації повернення потрібно запам'ятати адресу сегменту та адрес команди – стан регістрів CS та IP. Для повернення керування перерваній програмі використовується команда RET, яка знаходиться в процедурі і по закінчення її роботи зчитує із стека адресу повернення та перезаписує в регістри CS та IP.

**Організація циклів**

Для організації багатократного звертання до виділеного фрагменту програми використовують команди умовної та безумовної передачі керування. Але існує спеціальна команда організації циклів, яка має декілька модифікацій: LOOP, LOOPE або LOOPZ, LOOPNE або LOOPNZ.

Для завдання кількості повторів циклу в цьому випадку використовують регістр CX. Розміщення мітки для команди LOOP не повинне виходити з діапазону від –128 до +127 байт. Команди організація циклу та її модифікації мають наступну структуру:

loop мітка

loope/loopz мітка

loopne/loopnz мітка

При виконанні команди циклу спочатку зменшується на 1 вміст регістра CX і проводиться перевірка його вмісту. Якщо CX > 0, то керування передається на мітку переходу і виконується цикл, в іншому випадку керування передається на наступну після loop команду. Якщо задати нульове значення регістра CX , то цикл буде повторюватися 2^16 = 65 536 разів.

Команди LOOPE/LOOPZ й LOOPNE/LOOPNZ дозволяють організувати достроковий вихід із циклу, провівши додатково аналіз прапора ZF.

Команди LOOPE й LOOPZ виконують однакові дії. Вони дозволяють повторювати цикл, поки CX > 0 або ZF = 1. Якщо CX = 0 або ZF = 0, то керування передається на наступну після POOPE/LOOPZ команду.

Для запобігання виконання циклу при нульовому значенні CX перед початком циклу необхідно перевірити його вміст за допомогою команди JCXZ (Jump if CX = Zero), яка виконує обхід циклу при CX = 0. Команда має наступну структуру:

jcxz мітка.

Команди LOOPE/LOOPZ й LOOPNZ/LOOPNE зручно використати разом з командами, які в результаті своєї роботи міняють значення прапора zf. Наприклад, організувати заповнення масиву в програмі.

**Завдання:**

1. Сформувати матрицю 10х10, та поміняти місцями 5 рядок та 5 стовпець.

**Рішення:**

Dat Segment

arr db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 2,2,2,2,3,2,2,2,2,2,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

db 0,0,0,0,1,0,0,0,0,0,

len equ 10 ;размерность массива

n equ 5 ;номер строки и столбца, которые нужно поменять

Dat EndS

Codl Segment

assume ds:dat, cs:codl

start:

mov ax, dat

mov ds, ax

xor ax, ax

mov si, offset arr

mov cx, len

mov bx, 0

mov dx, 0

L1:

push cx

mov al, [si + (n - 1) \* len + bx]

push bx

mov bx, dx

mov cl, [si + (n - 1) + bx]

mov [si + (n - 1) + bx], al

pop bx

mov [si + (n - 1) \* len + bx], cl

pop cx

add bx, 1

add dx, len

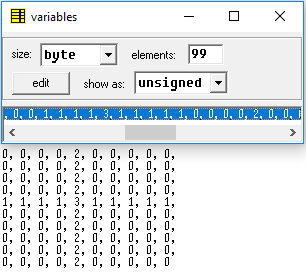
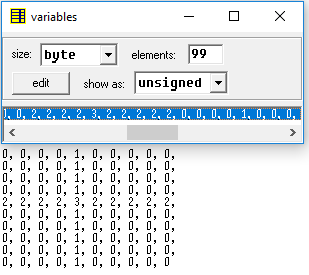
loop L1

Codl EndS

End start

**Результат роботи програми:**

Before: After:



**Висновок**: на цій лабораторній роботі я навчився розробляти програми на мові Асемблер з використанням команд передачі керування та організації циклів при обробці інформації.