ЛАБОРАТОРНА РОБОТА № 5.

Програмування задач циклічної структури. Робота з масивами.

Мета: навчитися програмувати задачі циклічної структури, набути навики оголошення і опрацювання масивів.

Команди керування програмою.

Команда безумовного переходу **JMP**.

Дана команда додає значення вказане в операнді до значення в регістрі IP, який містить адресу команди, що розташована після JMP. Відповідно, якщо здійснюється перехід назад, то значення операнду буде від'ємне, якщо вперед – додатне.

Команди умовних переходів.

Працюють аналогічно до JMP та LOOP, за винятком того, що довжина переходу повинна бути в межах від -128 до +128 байт.

Переходи для беззнакових даних.

Мнемоніка	Опис	Прапорці, що перевіряються
JE/JZ	Перехід, якщо рівно/нуль	ZF
JNE/JNZ	Перехід, якщо не рівно/не нуль	ZF
JA/JNBE	Перехід, якщо вище/не нижче або рівно	ZF,CF
JAE/JNB	Перехід, якщо вище або рівно/ не нижче	CF
JB/JNAE	Перехід, якщо нижче/не вище або рівно	CF
JBE/JNA	Перехід, якщо нижче або рівно/не вище	CF,AF

Будь яку перевірку можна закодувати двома мнемо кодами. Наприклад JB та JNAE генерують однаковий об'єктний код, хоча позитивну перевірку JB легше зрозуміти аніж заперечну JNAE.

Переходи для знакових даних.

Мнемоніка	Опис	Прапорці, що перевіряються
JE/JZ	Перехід, якщо рівно/нуль	ZF
JNE/JNZ	Перехід, якщо не рівно/не нуль	ZF
JG/JNLE	Перехід, якщо більше/не менше або рівно	ZF,SF,OF
JGE/JNL	Перехід, якщо більше або рівно/ не менше	SF,OF
JL/JNE	Перехід, якщо менше/не більше або рівно	SF,OF
JLE/JNG	Перехід, якщо менше або рівно/не більше	ZF,SF,OF

Команди переходу для умови рівно або нуль (JE/JZ) та не рівно або не нуль (JNE/JNZ) існують як для знакових так і для без знакових даних. Стан рівно/нуль наступає незалежно від знаку.

Спеціальні арифметичні перевірки.

Мнемоніка	Опис	Прапорці, що перевіряються
JS	Перехід, якщо є знак(від'ємне)	SF
JNS	Перехід, якщо немає знаку(додатнє)	SF
JC	Перехід, якщо є перенос (аналогічно JB)	CF
JNC	Перехід, якщо немає переносу	CF
JO	Перехід, якщо є переповнення	OF
JNO	Перехід, якщо немає переповнення	OF
JP/JPE	Перехід, якщо паритет парний	PF
JNP/JP	Перехід, якщо паритет непарний	PF

JCXZ - ще одна команда умовного переходу перевіряє чи рівний вміст регістра CX нулю. Ця команда повинна знаходитися безпосередньо після арифметичної команди чи команди порівняння. Одним з можливих місць застосування команди JCXZ може бути початок циклу, де вона перевіряє чи не містить регістр CX нульового значення.

Оператор циклу **LOOP.**

Передає керування на мітку, вказану як операнд, поки значення в регістрі CX не стане рівне нулю.

```
        mov  cx, 5
Lab:
        mov  ax, bx
        inc ax
        loop Lab
```

тобто оператори (mov ax, bx) та (inc ax) повторяться 5 разів.

Існують додатково також різновиди цієї команди:

1. LOOPE(або LOOPZ) – передає керування за адресою, вказаною в операнді якщо CX містить нульове значення і встановлений прапорець нуля, тобто ZF=1.
2. LOOPNE(або LOOPNZ) передає керування за адресою, вказаною в операнді якщо CX містить нульове значення і прапорець нуля скинуто, тобто ZF=0.

Команди порівняння

CMP - Порівняння двох полів даних.

Команда виконує віднімання другого операнду від першого, але значення операндів не змінює. Операнди повинні бути однакової довжини: байт або слово. Команда може мати наступний формат:

CMP регістр-регістр;
CMP пам'ять-регістр;
CMP безпосереднє значення -регістр;
CMP регістр-пам’ять;
CMP безпосереднє значення –пам’ять

Ця команда впливає на прапорці AF,CF,OF, PF,SF,ZF

CMPS, CMPSB, CMPSW Порівняння рядків довільної довжини. CMPSB – порівнює по байту, CMPSW – порівнює по словах. Перший операнд цих команд адресується регістровою парою DS:SI, а другий – ES:DI. Якщо прапорець DF=0, то порівняння відбувається зліва направо і регістри SI та DI збільшуються після кожного порівняння, інакше – навпаки і регістри SI та DI зменшуються. Як правило, використовуються з префіксами повтору. Впливають на прапорці AF,CF,OF,PF,SF,ZF.

Команда LEA

Команда **LEA** здійснює завантаження в регістр так званої ефективної адреси:

LEA регістр, пам’ять

Команда не змінює прапорців. У найпростішому випадку за допомогою команди LEA можна завантажити в регістр адресу змінної або початку масиву:

```
x dd 100 dup (0)
lea ebx, x
```

Масиви

Масиви в асемблері описуються за допомогою директив визначення даних з використанням конструкції повторення dup. Для того, щоб звернутися до елемента масиву, необхідно так чи інакше вказати адресу початку масиву та зміщення до елемента в масиві. Зміщення першого елемента масиву завжди рівне 0. Зміщення інших елементів масиву залежить від розміру елементів.

Нехай X - якийсь масив. Тоді адресу елемента масиву можна обчислити за такою формулою:

Адреса (X[i]) = X + (type X) * i, де i – номер елемента масиву, що починається з 0.

Нагадаємо, що ім'я змінної еквівалентно її адресі (для масиву – адресі початку масиву), а операція type визначає розмір змінної (для масиву визначається розмір елемента масиву відповідно до використаної директиви).

Для зручності в мові асемблера введено операцію модифікації адреси, яка схожа на індексний вираз у мовах високого рівня – до імені масиву треба приписати цілісний вираз або ім'я регістру у квадратних дужках:

```
x[4]
x[ebx]
```

Проте важлива відмінність у тому, що у програмі мовою високого рівня ми вказуємо індекс елемента масиву, а компілятор множить його за розмір елемента масиву, отримуючи зміщення елемента масиву. У програмі на асемблері вказується саме зміщення, тобто програміст повинен сам враховувати розмір елемента масиву. Компілятор мови асемблера просто додає зміщення до вказаної адреси. Наведені вище команди можна записати інакше:

```
x + 4
[x+4]
[x] + [4]
[x][4]
[x+ebx]
[x] + [ebx]
[x][ebx]
```

Зверніть увагу, що при використанні регістру для модифікації адреси наявність квадратних дужок є обов'язковою. Інакше компілятор зафіксує помилку.

Адреса може обчислюватися і за складнішою схемою:

<база> + <множник> * <індекс> + <зміщення>

База – це регістр чи ім'я змінної. Індекс має бути записаний у певному регістрі. Множник – це константа 1 (можна опустити), 2, 4 чи 8. Зміщення – ціле позитивне чи негативне число.

```
mov eax, [ebx + 4*ecx – 32]
mov eax, [x+2*ecx]
```

Контрольні запитання:

1. Що роблять команди умовних переходів?
2. Що робить команда LOOP?
3. Що робить команда LEA
4. Як можна звернутись до і-го елемента масиву?.
5. Як визначити зміщення до і-го елемента масиву?
6. Як завантажити адресу масиву у регістр?

Література:

1.Р.Джордейн.Справочник программиста персональных компъютеров типа IBM PC XT и AT. -М."Финансы и статистика",1992,стор.13-31.

2.Л.О.Березко,В.В.Троценко. Особливості програмування в турбо-асемблері. -Київ,НМК ВО,1992.

3.Л.Дао. Программирование микропроцессора 8088.Пер.с англ.-М."Мир",1988.

4.П.Абель.Язык ассемблера для IBM PC и программирования. Пер. з англ.-М.,"Высшая школа",1992.

ЗАВДАННЯ

1. Створити *.exe програму, яка реалізовує задачу, задану варіантом і зберігає результат в пам'яті. Вхідні дані число A і масив чисел X вважати 32-х розрядними цілими знаковими числами. Розмір масиву X має бути не менше 10 елементів.
2. Скласти звіт про виконану роботу з приведенням тексту програми та коментарів до неї.
3. Дати відповідь на контрольні запитання.

ВАРІАНТИ ЗАВДАННЯ:

№	Завдання
1	Задані число *A* і масив чисел *X*. Знайти суму чисел масиву, які менші за *A*.
2	Задані число *A* і масив чисел *X*. Знайти скільки раз в масиві зустрічається *A*.
3	Задано масив чисел *X*. Визначити, скільки із них більші за своїх "сусідів", тобто попереднього і наступного числа (вважаємо, що перше і останнє числа сусідів не мають).
4	Задано масив чисел *X*. Визначити, скільки раз у масиві змінюється знак. (Наприклад, в масиві 1, -34, 8, 14, -5 знак змінюється три рази).
5	Задані число *A* і масив чисел *X*. Знайти скільки раз в масиві зустрічаються числа, більші за *A*.
6	Задано масив чисел *X*. Визначити порядковий номер найбільшого числа.
7	Задано масив чисел *X*. Визначити, скільки із них приймають найбільше значення.
8	Задані число *A* і масив чисел *X*. Отримати суму чисел додатних та більших за A.
9	Задані число *A* і масив чисел *X*. Перевірити чи в масиві зустрічається *A* (1 – зустрічається, 0 – не зустрічається).
10	Задано масив чисел *X*. Знайти суму найбільшого та найменшого чисел.
11	Задано масив чисел *X*. Знайти середньоарифметичне значення масиву.
12	Задані число *A* і масив чисел *X*. Знайти суму *A* перших чисел в масиві (врахувати, що *A* може бути менше нуля і більше, ніж елементів в масиві).
13	Задано масив чисел *X*. Перевірити чи числа в масиві відсортовані за зростанням (1 – відсортовані, 0 – не відсортовані).
14	Задано масив чисел *X*. Визначити яких чисел в масиві більше (1 – додатних, 0 – від'ємних).
15	Задано масив чисел *X*. Визначити, скільки із них менші за наступне число (останнє число наступного не має).
16	Задані число *A* і масив чисел *X*. Знайти суму чисел масиву, які більші або рівні *A*.
17	Задані число *A* і масив чисел *X*. Знайти скільки раз в масиві зустрічаються числа, менші або рівні *A*.
18	Задано масив чисел *X*. Визначити порядковий номер найменшого числа.
19	Задані число *A* і масив чисел *X*. Знайти скільки раз в масиві зустрічаються числа в діапазоні від 0 до *A* включно.
20	Задані число *A* і масив чисел *X*. Отримати суму чисел від'ємних та менших за A.
21	Задані число *A* і масив чисел *X*. Перевірити чи в масиві зустрічається число *A*A* (1 – зустрічається, 0 – не зустрічається).
22	Задані число *A* і масив чисел *X*. Знайти суму *A* останніх чисел в масиві (врахувати, що *A* може бути менше нуля і більше, ніж елементів в масиві).
23	Задано масив чисел *X*. Перевірити чи числа в масиві відсортовані за спаданням (1 – відсортовані, 0 – не відсортовані).
24	Задані число *A* і масив чисел *X*. Визначити яких чисел в масиві більше (1 – більших за *A*, 0 – менших або рівних *A*).
25	Задано масив чисел *X*. Визначити, скільки із них більші за попереднє число (перше число попереднього не має).
26	Задано масив чисел *X*. Знайти найбільше непарне число.
27	Задано масив чисел *X*. Визначити яких чисел в масиві більше (1 – парних, 0 – непарних).
28	Задано масив чисел *X*. Знайти суму парних чисел масиву.
29	Задано масив чисел *X*. Знайти суму непарних чисел в масиву.
30	Задано масив чисел *X*. Знайти найменше парне число.

Приклад виконання.

Завдання: Написати програму, яка знаходить суму чисел заданого масиву і результат записує в пам’ять.

Повний текст програми, що виконує дані обчислення приведений нижче. Остаточний результат обчислень у буде знаходитися в пам’яті за адресою **Res.**

```
.686
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib

.data

X dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10        ;масив
N equ ($-X) / type X                      ;розмір масиву
Res dd ?                                  ;результат

Result db 13, 10, 'Result =           ', 13, 10
NumberOfCharsToWrite dd $-Result
format db '%d', 0
hConsoleOutput dd 0
NumberOfCharsWritten dd 0

.code
start:

;обчислення суми в циклі
mov ecx, N
mov edx, 0
L:
 mov eax, [X + ecx * type X - type X]
 add edx, eax
 loop L
mov Res, edx

;вивід результату
push Res
push offset format
push offset [Result+11]
call wsprintfA
push -11
call GetStdHandle
mov hConsoleOutput, eax
push 0
push offset NumberOfCharsWritten
push NumberOfCharsToWrite
push offset Result
push hConsoleOutput
call WriteConsoleA

push 0
call ExitProcess
end start
```

Частину коду, де відбувається обчислення суми можна записати наступним чином, використовуючи не команду LOOP, а команду умовного переходу:

```
;обчислення суми в циклі
mov ecx, 0
mov edx, 0
mov edi, 0
lea ebx, X
L:
 mov eax, [ebx][edi] ;mov eax, [ebx+edi]
 add edx, eax
 add   edi, 4
 inc ecx
 cmp ecx, N
 jle L
mov Res, edx
```