Лабораторна робота №4

Тема: «Основи роботи зі стеком. Побудова найпростіших процедур»

Мета роботи: навчитися використовуючи команди роботи зі стеком, а також навчитися будувати найпростіші процедури.

Хід роботи

Завдання на 3 бали

1. Відкомпілювати та відлагодити приклади представлені у роботі. Пояснити їх роботу, стани регістрів та принципи роботи команд.

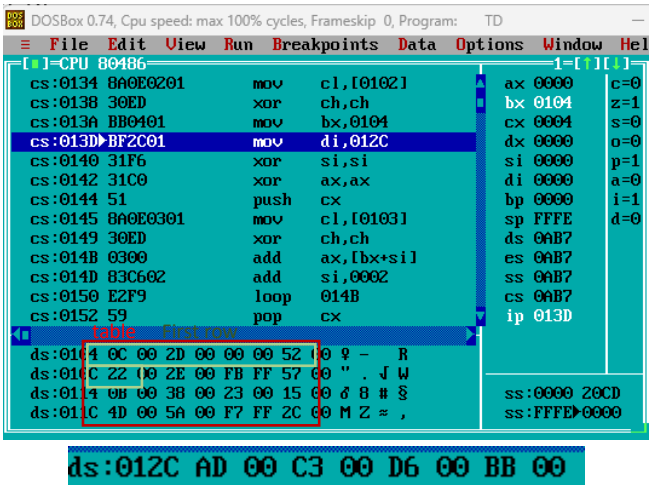
Приклад 1

Є двовимірний масив - таблиця значень зі знаком розміром n рядків на m стовпців. Програма обчислює суму елементів кожного рядка і зберігає результат в масиві sum. Перший елемент масиву буде містити суму елементів першого рядка, другий елемент - суму елементів другого рядка і так далі.

Рішення

```
use16
org 100h

jmp start
; ---------------------------------------------------
; дані
n db 4                ; кількість рядків
m db 5                ; кількість стовпців

; двовимірний масив (таблиця)
table:
    dw 12,45,0,82,34
    dw 46,-5,87,11,56
    dw 35,21,77,90,-9
    dw 44,13,-1,99,32

sum rw 4     ; масив для сум кожного рядка
; ---------------------------------------------------
start:
    mov cl, [n]        ; лічильник рядків у CL
    xor ch, ch        ; очищаемо старшу частину регістру від
сміття
    mov bx, table      ; BX = адреса таблиці
    mov di, sum        ; DI = адреса масиву для сум
    xor si, si         ; зсув елемента від початку таблиці

rows:
    xor ax, ax         ; AX = 0 (сума рядка)
```

```
    push cx             ; зберігаємо кількість рядків

    mov cl, [m]         ; кількість стовпців у CL
    xor ch, ch

calc_sum:
    add ax, [bx + si] ; додаємо елемент рядка
    add si, 2           ; перехід до наступного елемента, 1
елемент = 2 байти
    loop calc_sum       ; цикл підсумовування рядка

    pop cx              ; відновлюємо лічильник рядків
    mov [di], ax        ; зберігаємо суму поточного рядка
    add di, 2           ; адреса для наступної суми
    loop rows           ; цикл по всіх рядках

exit:
    mov ax, 4C00h
    int 21h
```



Робота програми – масив для сум кожного рядка

Приклад 2

Необхідно оголосити в програмі рядок «$!olleH». Написати програму для перевертання рядку з використанням стека (в циклі необхідно розмістити кожен символ в стек, а потім витягнути в зворотному порядку). Вивести отриманий рядок на екран.

Рішення

```
use16
org 100h
      jmp start
; --------------------------
      string db "$!olleH"
```

```
      n db 7
; ---------------------------
start:
   xor si, si          ; зсув елемента від початку рядка
   movzx cx, [n]      ; встановлюємо лічильник для lp1
lp1:
   add si, 1                   ; рухаємось по рядку
   movzx dx, byte [string + si];заносимо у BX елемент рядка
   push dx                     ;заносимо BX у стек
   loop lp1                   ;заносимо всі елементи у стек
   movzx cx, [n]             ;оновлюємо лічильник для lp2
   mov ah, 02h
lp2:
   pop dx
   int 21h
   loop lp2            ;дістаємо з стеку всі елементи
   mov ah, 08h
   int 21h ;очікування натискання будь якої клавіші для
завершення

   mov ax, 4C00h
   int 21h
```

```
C:\>td first_project.com
Turbo Debugger Version 5.0 Copyright (c) 1988,96 Borland International
Hello!
```

Робота програми

Приклад 3

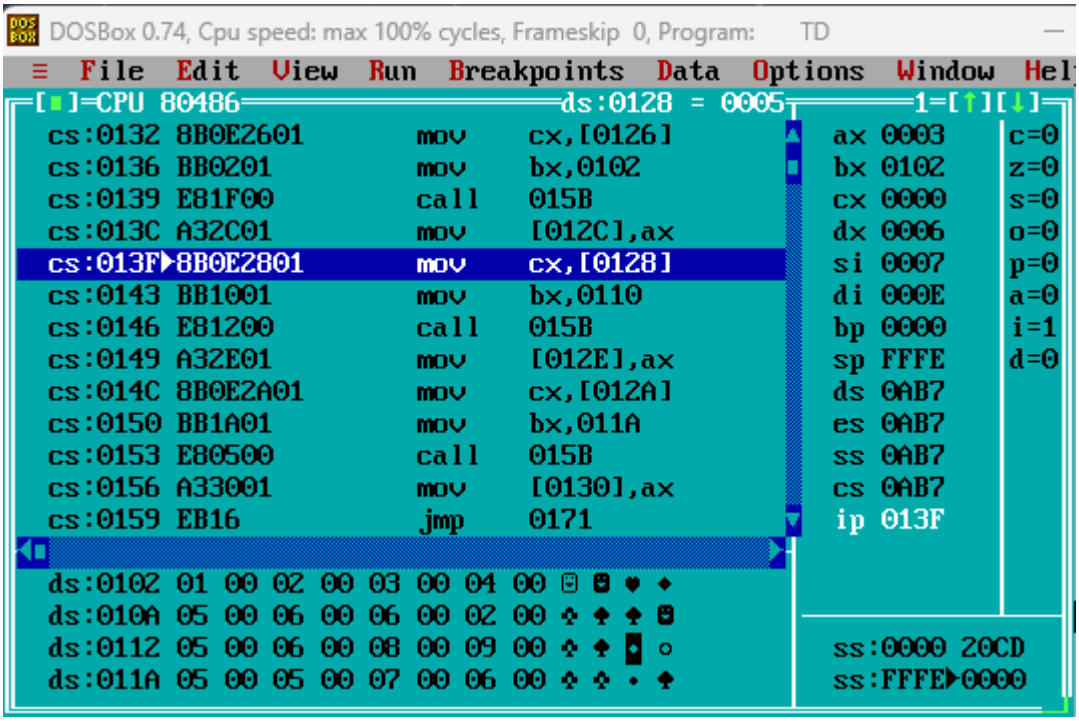
Необхідно оголосити в програмі 3 масиви слів без знаку. Кількість елементів кожного масиву має бути різною і зберігатися в окремій змінній без знаку. Написати процедуру для обчислення середнього арифметичного масиву чисел. У якості параметрів передавати адресу масиву і кількість елементів, а повертати вона повинна розраховане значення. За допомогою процедури обчислити середнє арифметичне кожного масиву.

Рішення

```
use16
org 100h
      jmp start
; -------------------------------
      array1 dw 1,2,3,4,5,6,6
      array2 dw 2,5,6,8,9
      array3 dw 5,5,7,6,8,8
```

```
      n1 dw 7
      n2 dw 5
      n3 dw 6

      sr1 rw 1
      sr2 rw 1
      sr3 rw 1
; ----------------------------------
; Вхід: BX = адреса масиву, CX = кількість елементів
; Вихід: AX = середнє значення, DX = залишок від ділення
; ----------------------------------
start:
      mov cx, [n1]    ;передаємо к-ть елементів
      mov bx, array1  ;передаємо адресу масиву
      call sr_arifm   ;викликаємо процедуру
      mov [sr1], ax   ;передаємо початкове значення 1

      mov cx, [n2]
      mov bx, array2
      call sr_arifm
      mov [sr2], ax

      mov cx, [n3]
      mov bx, array3
      call sr_arifm
      mov [sr3], ax

      jmp quit

sr_arifm:
      xor di, di
      xor si, si
      xor ax, ax
      xor dx, dx    ;очищаємо регістри
      mov si, cx    ;переносимо к-ть елементів у лічильник
lp:
      add ax, [bx + di] ;додаємо до сер.арф. елемент масиву
      add di, 2  ;наступний елемент
      jcxz return
      loop lp
return:
```

```
        div si  ;ділимо суму елементів масиву на їх кількість
                ; DX:AX / SI
        ret     ;повертаємо у AX обчислене значення
quit:
        mov ax, 4c00h
        int 21h
```



Робота програми

Завдання на 4 бали

1. Виконати завдання на 3 бали.
2. Написати програму, для визначення суми:

$$\sum_{i=1}^{11}\left(F(a,b,c,d)+q\cdot i\right)$$

Варіант	F(a, b, c, d)	a	b	c	d
6	a*b + (c-d)	16	7	29	11

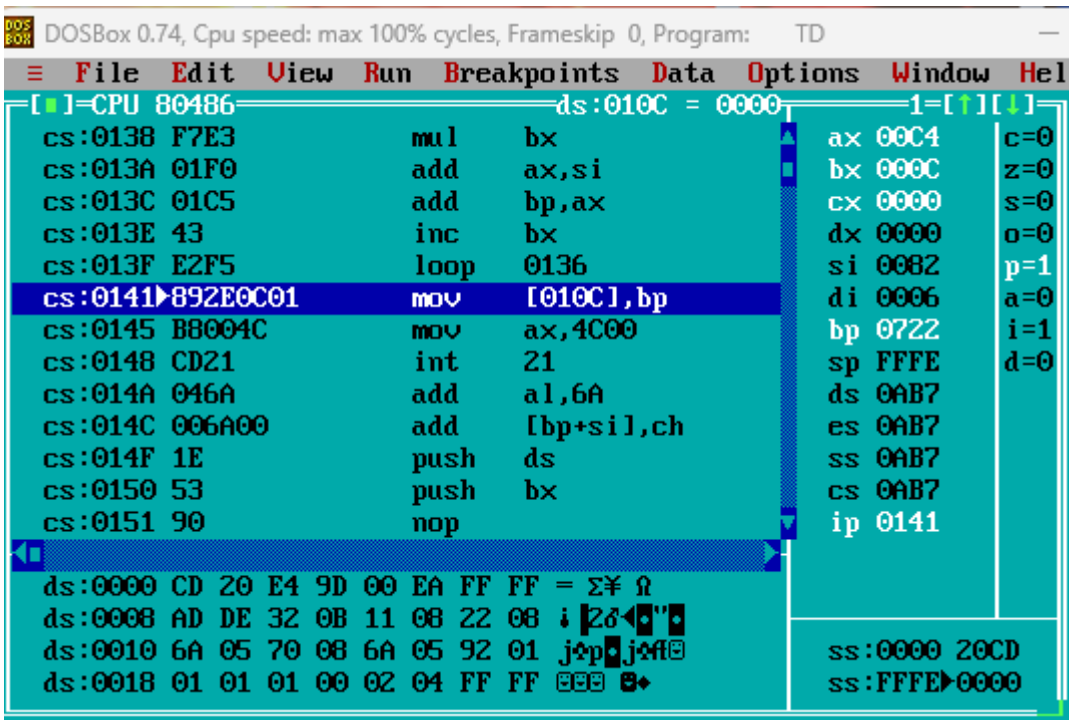
$$\sum_{i=1}^{11}(a\times b)+(c-d)+q\times i$$

$a=16; b=7; c=29; d=11; q=6$

Рішення

```
use16
org 100h
jmp start
; ----------------------------
a dw 16
```

```
b dw 7
c dw 29
d dw 11
q dw 6
sum dw 0
; ------------------------------
start:
    push [a]
    push [b]
    push [c]
    push [d]

    pop dx      ; d
    pop cx      ; c
    pop bx      ; b
    pop ax      ; a

    ;(c - d)
    mov si, cx
    sub si, dx     ; si = (c - d)= 18 (12)

    ;(a * b)
    imul bx         ; ax = a * b = 122 (70)
    add si, ax      ; si = (a*b) + (c-d) = 130 (82)

    xor bp, bp      ; bp буде для результату
    mov bx, 1       ; i = 1
    mov cx, 11      ; 11 ітерацій циклу
    mov di, [q]     ; q = 6

sum_loop:
    mov ax, di       ; ax = q = 6
    mul bx           ; ax = q * i
    add ax, si     ; (a*b)+(c-d)+q*i = 136 (88(перший цикл))
    add bp, ax      ; складаємо цикли
    inc bx
    loop sum_loop

    mov [sum], bp   ; результат = 1826 (722)

    mov ax, 4C00h
    int 21h
```



Робота програми

Завдання на 5 балів

1. Виконати завдання на 3 та на 4 бали.
2. Виконати завдання із таблиці 4.2

Написати функцію яка знаходить суму головної та побічної діагоналі матриці

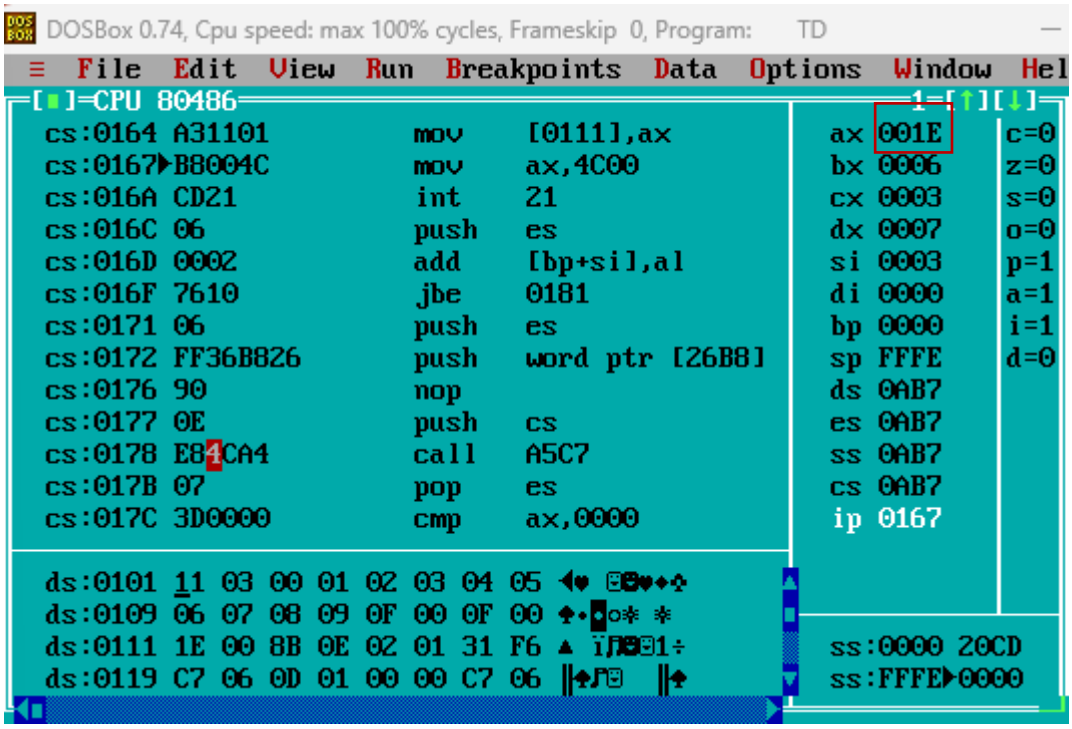
Головна діагональ: 1 + 5 + 9 = 15

Побічна діагональ: 3 + 5 + 7 = 15

Результат = 30

```
use16
org 100h
jmp start

; --------------------
n            dw 3
matrix       db 1,2,3,
                4,5,6,
                7,8,9
sum_main     dw 0
sum_side     dw 0
total_sum    dw 0
; --------------------
start:
    mov cx, [n]          ; cx = n
    xor si, si           ; i = 0

    mov word [sum_main], 0
```

```
    mov word [sum_side], 0
    mov word [total_sum], 0
loop_i:
    ; ------- main_index = i*n + i -------
    mov ax, si          ; ax = i
    mul cx              ; AX = i * n
    add ax, si          ; AX = i*n + i
    mov bx, ax
    mov dl, [matrix + bx]   ; dl = matrix[ main_index ]

    xor ah, ah
    mov al, dl
    add [sum_main], ax

    ; ------- side_index = i*n + (n-1-i) -------
    mov ax, si
    mul cx              ; AX = i * n
    mov bx, ax          ; bx = i*n
    mov dx, cx
    dec dx              ; dx = n - 1
    sub dx, si          ; dx = n - 1 - i
    add bx, dx          ; bx = i*n + (n-1-i)
    mov dl, [matrix + bx]   ; dl = matrix[ side_index ]

    xor ah, ah
    mov al, dl
    add [sum_side], ax

    inc si
    cmp si, cx
    jl loop_i

    ; total_sum = sum_main + sum_side
    mov ax, [sum_main]
    add ax, [sum_side]
    mov [total_sum], ax

    mov ax, 4C00h
    int 21h
```



Контрольні запитання

1. Що таке стек? Пояснити роботу стеку за принципом LIFO та FIFO.

Стек – це структура даних, що працює за принципом LIFO («останній прийшов – перший вийшов»): елемент, доданий останнім, витягується першим. Принцип FIFO («перший прийшов – перший вийшов») діє навпаки – першим обробляється елемент, який був доданий раніше.

2. Які існують команди роботи зі стеком?

PUSH – додавання елемента у стек;
POP – витягнення елемента зі стеку;
PUSHA/POPA – збереження/відновлення всіх регістрів;
PUSHF/POPF – збереження/відновлення прапорів.

3. Що таке функція, процедура та підпрограма?

Функція, процедура або підпрограма – це окремий блок програми, який виконує певну операцію й може викликатися з різних частин коду. Процедура зазвичай не повертає значення, тоді як функція передає результат виконання через спеціальний регістр.