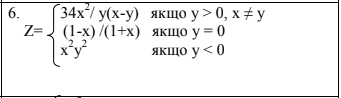
**Комп’ютерний практикум №3**

**Тема:** програмування розгалужених алгоритмів

**Завдання**:

Написати програму, яка буде обчислювати значення функції.

**Варіант:** №6



**Код програми**:

; STACK SEGMENT

STSEG SEGMENT PARA STACK "STACK"

    DB 64 DUP (0)

STSEG ENDS

; DATA SEGMENT

DSEG SEGMENT PARA PUBLIC "DATA"

    xStr db "Enter x : $"

    yStr db "Enter y : $"

    resultStr db "Result : $"

    remainderStr db "Reminder of division : $"

    overflowError db "Overflow error", 10, "$"

    zeroDivision db "Zero division", 10, "$"

    xBuffer db 6, ?, 6 dup ('?')

    yBuffer db 6, ?, 6 dup ('?')

    x dw ?

    y dw ?

    result dw ?

    remainder dw ?

    was\_overflow db 0

DSEG ENDS

; CODE SEGMENT

CSEG SEGMENT PARA PUBLIC "CODE"

    MAIN PROC FAR

        ASSUME cs: CSEG, ds: DSEG, ss:STSEG

        push ds

        xor ax, ax

        push ax

        mov ax, DSEG

        mov ds, ax

        ; READ X

        lea dx, xStr

        call far ptr WRITING

        lea dx, xBuffer

        call far ptr READING

        mov ax, 10

        int 29h

        ; READ Y

        lea dx, yStr

        call far ptr WRITING

        lea dx, yBuffer

        call far ptr READING

        mov ax, 10

        int 29h

        lea bx, xBuffer + 2

        call far ptr ATOI

        cmp was\_overflow, 1

        je overflow\_error\_main

        mov x, ax

        lea bx, yBuffer + 2

        call far ptr ATOI

        cmp was\_overflow, 1

        je overflow\_error\_main

        mov y, ax

        cmp y, 0

        jg maybe\_first

        je second

        jl third

        maybe\_first:

            mov ax, y

            cmp x, ax

            je main\_exit

            jmp first

        first:

            call far ptr FIRST\_ACTION

            mov remainder, dx

            cmp was\_overflow, 1

            je main\_exit

            jmp writing\_output

        second:

            call far ptr SECOND\_ACTION

            mov remainder, dx

            cmp was\_overflow, 1

            je main\_exit

            jmp writing\_output

        third:

            call far ptr THIRD\_ACTION

            mov remainder, dx

            cmp was\_overflow, 1

            je main\_exit

            jmp writing\_output

        overflow\_error\_main:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, overflowError

            call far ptr WRITING

            jmp main\_exit

        writing\_output:

            mov result, ax

            call far ptr WRITE\_OUTPUT

        main\_exit:

            ret

    MAIN ENDP

    WRITE\_OUTPUT PROC FAR

        lea dx, resultStr

        call far ptr WRITING

        mov bx, result

        call far ptr ITOA

        mov ax, 10

        int 29h

        lea dx, remainderStr

        call far ptr WRITING

        mov bx, remainder

        call far ptr ITOA

        ret

    WRITE\_OUTPUT ENDP

    FIRST\_ACTION PROC FAR   ; (34 \* (x ^ 2)) / (y \* (x - y)). [y > 0, x <> y]

        xor ax, ax

        mov ax, 34

        mov bx, x

        xor cx, cx

        mov cx, 2

        loop\_label:

            imul bx

            jo overflow\_error\_first

            loop loop\_label

        mov cx, ax ; x = 34 \* (x ^ 2)

        mov bx, x

        mov ax, y

        sub bx, ax

        jo overflow\_error\_first

        mov ax, y

        imul bx

        jo overflow\_error\_first

        xor dx, dx

        mov bx, ax

        mov ax, cx

        idiv bx

        jmp exit\_first

        overflow\_error\_first:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, overflowError

            call far ptr WRITING

        exit\_first:

            ret

    FIRST\_ACTION ENDP

    SECOND\_ACTION PROC FAR  ;( 1 - x ) / ( 1 + x ). [y = 0]

        mov bx, x

        add bx, 1

        jo overflow\_error\_second

        mov ax, x

        neg ax

        add ax, 1

        jo overflow\_error\_second ; ax=1-x  bx=1+x

        xor cx, cx

        cmp ax, 0

        jg ax\_positive

        inc cx ; ax negative

        neg ax

        ax\_positive:

        cmp bx, 0

        je zeroDivision\_label

        jg bx\_positive

        inc cx ; bx negative

        neg bx

        bx\_positive:

        xor dx, dx

        idiv bx

        cmp cx, 0

        je exit\_second

        loop\_negResult:

            neg ax

            loop loop\_negResult

        jmp exit\_second

        overflow\_error\_second:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, overflowError

            call far ptr WRITING

        zeroDivision\_label:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, zeroDivision

            call far ptr WRITING

        exit\_second:

            ret

    SECOND\_ACTION ENDP

    THIRD\_ACTION PROC FAR ; (x ^ 2) \* (y ^ 2). [y < 0]

        mov ax, x

        mov bx, x

        imul bx ; x ^ 2

        jo overflow\_error\_third

        mov bx, y

        xor cx, cx

        mov cx, 2

        label\_loop\_third:

            imul bx

            jo overflow\_error\_third

            loop label\_loop\_third

        jmp exit\_third

        overflow\_error\_third:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, overflowError

            call far ptr WRITING

        exit\_third:

            ret

    THIRD\_ACTION ENDP

    ITOA PROC FAR

        or bx, bx

        jns positive\_number

        xor ax, ax

        mov al, '-'

        int 29h

        neg bx

        positive\_number:

            mov ax, bx

            xor cx, cx  ; chars number

            mov bx, 10  ; diviator

            itoa\_loop:

                xor dx, dx  ; remainder stores here

                div bx

                add dl, '0'

                push dx

                inc cx

                test ax, ax

                jnz itoa\_loop

            output:

                pop ax

                int 29h

                loop output

            ret

        ITOA ENDP

    ATOI PROC FAR

        xor ax, ax      ; result

        xor cx, cx      ; char

        xor di, di      ; 10

        mov di, 10

        xor si, si      ; sign

        skip\_whitespaces:

            mov cl, BYTE PTR [bx]

            cmp cl, 32  ; ' '

            je step

            cmp cl , 9  ; '\t'

            je step

            jmp sign\_check

        step:

            inc bx

            jmp skip\_whitespaces

        sign\_check:

            cmp cl, 45  ; '-'

            je minus

            cmp cl, 43  ; '+'

            je plus

            jmp atoi\_loop

        minus:

            mov si, 1

            plus:

                inc bx

                jmp atoi\_loop

        atoi\_loop:

            mov cl, BYTE PTR [bx]

            cmp cl, 48  ; '0'

            jl atoi\_end

            cmp cl, 57  ; '9'

            jg atoi\_end

            imul di

            jo error

            sub cl, 48

            add ax, cx

            jo error

            inc bx

            jmp atoi\_loop

        atoi\_end:

            cmp si, 1

            je make\_neg

            jmp exit\_atoi

        make\_neg:

            neg ax

            jmp exit\_atoi

        error:

            mov was\_overflow, 1

            xor dx, dx

            lea dx, overflowError

            call far ptr WRITING

        exit\_atoi:

            ret

    ATOI ENDP

    READING PROC FAR

        mov ah, 10

        int 21h

        ret

    READING ENDP

    WRITING PROC FAR

        mov ah, 9

        int 21h

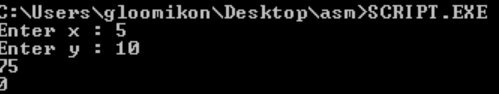
        ret

    WRITING ENDP

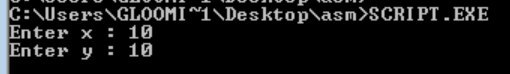
CSEG ENDS

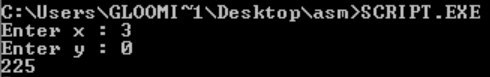
END MAIN

**Скріншоти:**



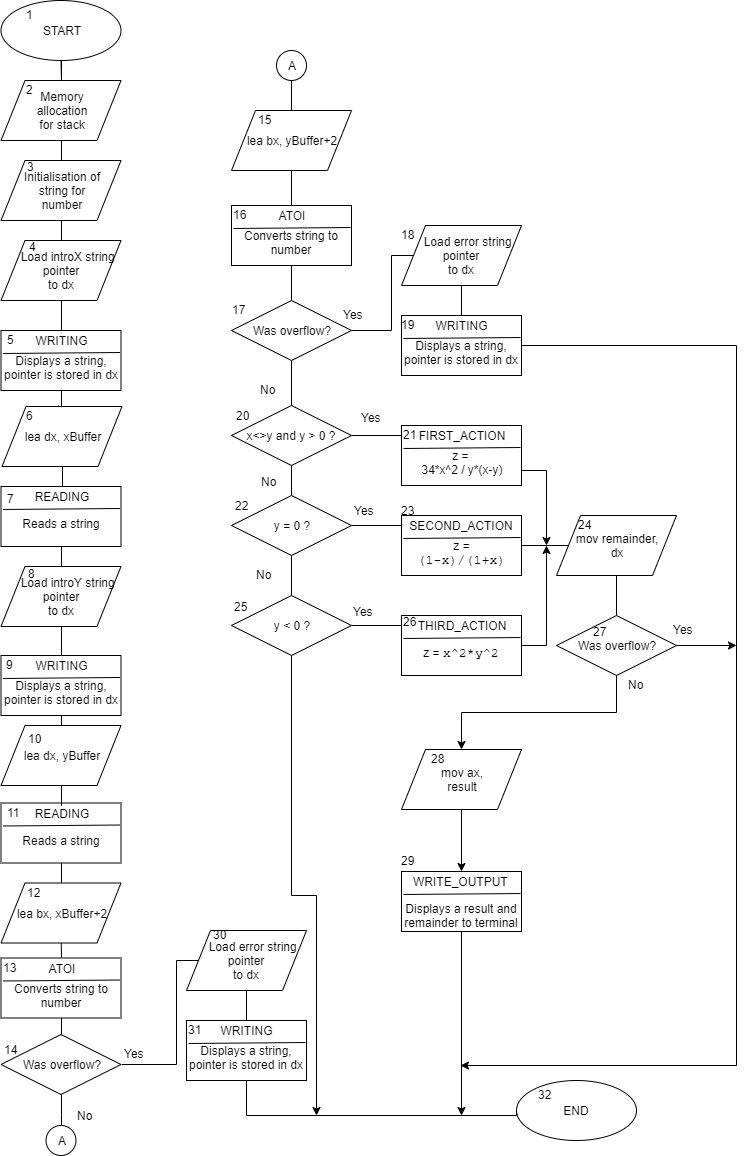


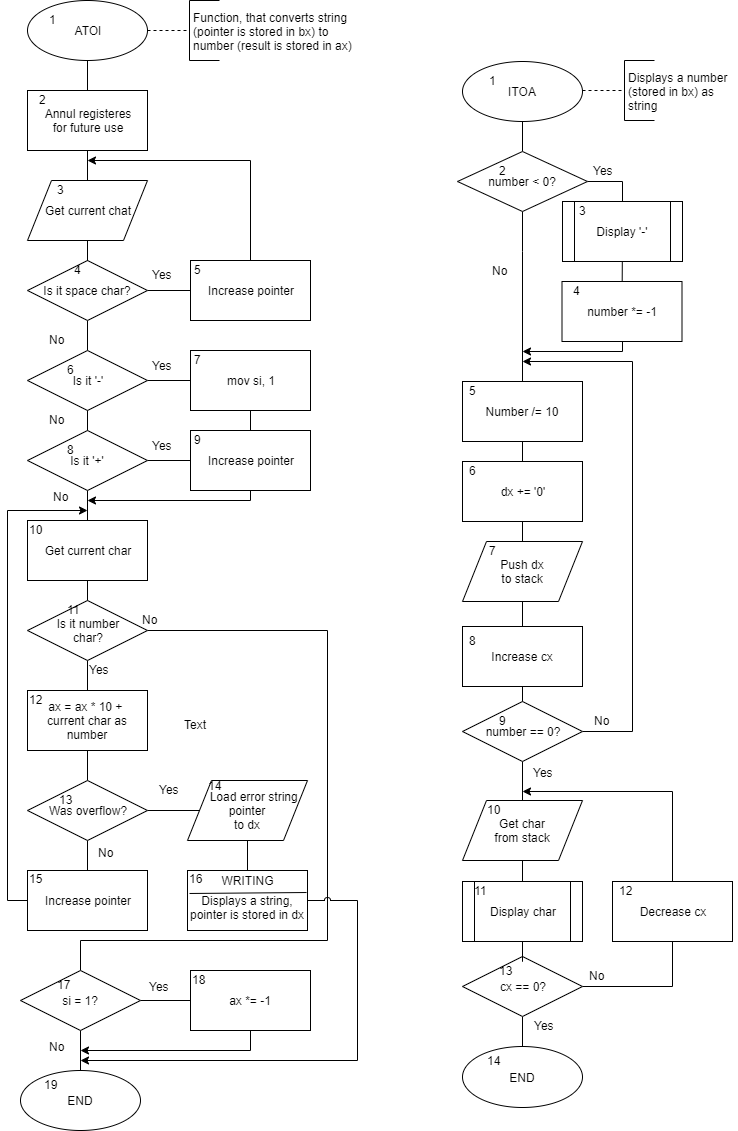


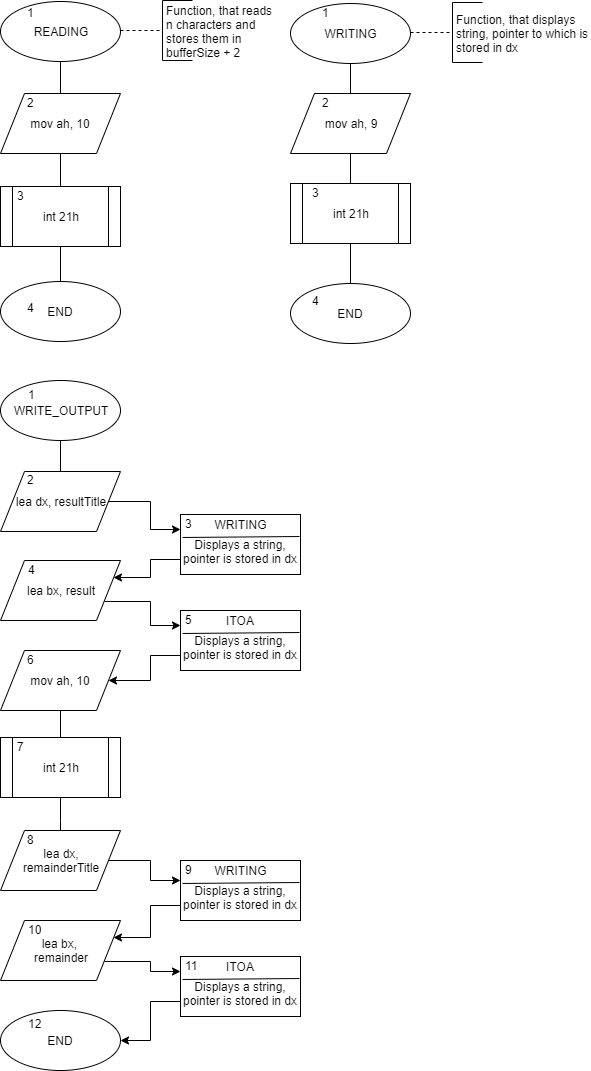


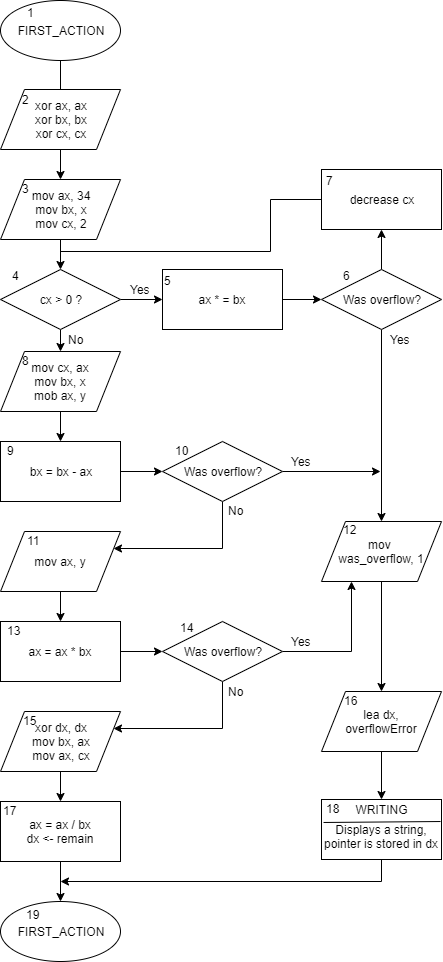


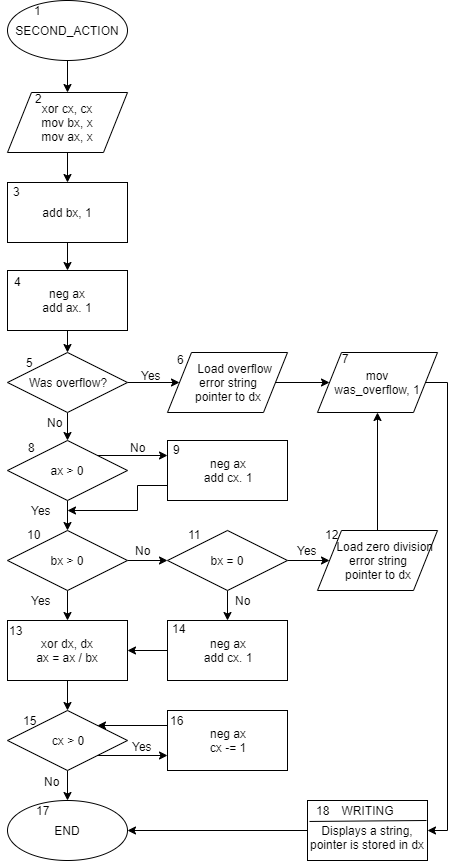
**Блок-схема**:

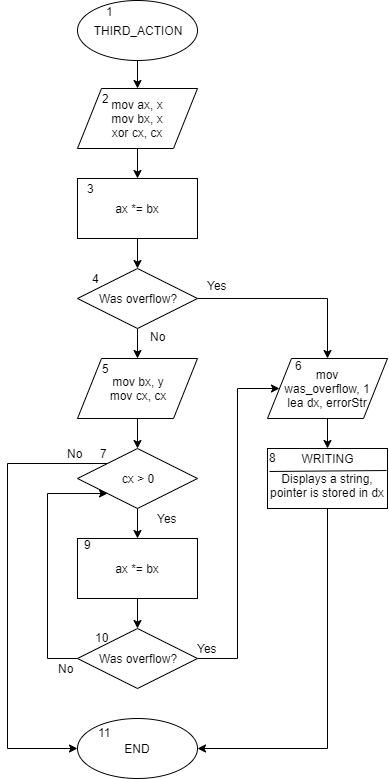












**Висновки**:

1. Був засвоєний принцип роботи з розгалуженнями
2. Були розібрані та використані мітки умовного переходу
3. Були засвоєні та використані арифметичні операції додавання, віднімання, множення та ділення
4. Був написаний структурований код
5. Були ще краще засвоєні синтаксичні конструкції мови асемблер