**Комп’ютерний практикум №5**

**Тема**: макрозасоби мови асемблер

**Завдання**:

1. переписати програму комп‘ютерного практикуму № 2 з використанням макросів;
2. переписати програму комп‘ютерного практикуму № 3 з використанням макросів;
3. переписати одну програму (на вибір викладача) комп‘ютерного практикуму № 4 з використанням макросів.

**Код програми:**

Базові макроси:

EXIT\_PROGRAM MACRO

    mov ax, 4c00h

    int 21h

ENDM

WRITING MACRO string

    push ax

    lea dx, string

    mov ah, 9

    int 21h

    pop ax

ENDM

PRINT\_CHAR MACRO chr

    push ax

    mov ax, chr

    int 29h

    pop ax

ENDM

READING MACRO buffer

    push ax

    lea dx, buffer

    mov ah, 10

    int 21h

    pop ax

    PRINT\_CHAR 10

ENDM

ATOI MACRO buffer

    LOCAL @@end\_label

    LOCAL @@error

    LOCAL @@loop\_label

    LOCAL skip\_whitespaces, step, sign\_check, minus, plus, atoi\_end, make\_neg

    lea bx, buffer + 2

    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 @@loop\_label

    minus:

        mov si, 1

        plus:

            inc bx

            jmp @@loop\_label

    @@loop\_label:

        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 @@loop\_label

    atoi\_end:

        cmp si, 1

        je make\_neg

        jmp @@end\_label

    make\_neg:

        neg ax

        jmp @@end\_label

    @@error:

        mov was\_overflow, 1

        WRITING overflowError

    @@end\_label:

ENDM

ITOA MACRO number

    LOCAL @@loop\_label

    LOCAL positive\_number, output

    mov bx, number

    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

    @@loop\_label:

        xor dx, dx  ; remainder stores here

        div bx

        add dl, '0'

        push dx

        inc cx

        test ax, ax

        jnz @@loop\_label

    output:

        pop ax

        int 29h

        loop output

ENDM

Комп’ютерний практикум №2:

include m\_basic.asm

; STACK SEGMENT

STSEG SEGMENT PARA STACK "STACK"

    DB 64 DUP (0)

STSEG ENDS

; DATA SEGMENT

DSEG SEGMENT PARA PUBLIC "DATA"

    overflowError db "Number is too big. From: -32768 to 32767$"

    startStr db "Enter a number : $"

    inputError db "Number has incorrect chars.", 10, "$"

    bufferSize DB 7     ; 6 chars + RETURN

    inputLength DB 0        ; number of read chars

    buffer DB 7 DUP('?')    ; actual buffer

    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

        WRITING startStr

        READING bufferSize

        ATOI bufferSize

        cmp was\_overflow, 1

        je overflow\_error

        add ax, 78

        jo overflow\_error

        ITOA ax

        jmp end\_main

        overflow\_error:

            WRITING overflowError

        end\_main:

            ret

    MAIN ENDP

CSEG ENDS

END MAIN

Макроси для функцій 3-го КП:

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

    LOCAL @@loop\_label, @@error, @@exit

    xor ax, ax

    mov ax, 34

    mov bx, x

    xor cx, cx

    mov cx, 2

    @@loop\_label:

        imul bx

        jo @@error

        loop @@loop\_label

    mov cx, ax ; x = 34 \* (x ^ 2)

    mov bx, x

    mov ax, y

    sub bx, ax

    jo @@error

    mov ax, y

    imul bx

    jo @@error

    xor dx, dx

    mov bx, ax

    mov ax, cx

    idiv bx

    mov result, ax

    mov remainder, dx

    jmp @@exit

    @@error:

        mov was\_overflow, 1

        WRITING overflowError

    @@exit:

ENDM

SECOND\_ACTION MACRO x, y, remainder, result ;( 1 - x ) / ( 1 + x ). [y = 0]

    LOCAL @@loop\_label, @@overflow\_error, @@exit, @@ax\_positive, @@bx\_positive, @@loop\_negResult, @@zeroDivision\_error

    mov bx, x

    add bx, 1

    jo @@overflow\_error

    mov ax, x

    neg ax

    add ax, 1

    jo @@overflow\_error ; 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\_error

    jg @@bx\_positive

    inc cx ; bx negative

    neg bx

    @@bx\_positive:

    xor dx, dx

    idiv bx

    @@loop\_negResult:

        neg ax

        loop @@loop\_negResult

    mov result, ax

    mov remainder, dx

    jmp @@exit

    @@overflow\_error:

        mov was\_overflow, 1

        WRITING overflowError

    @@zeroDivision\_error:

        mov was\_overflow, 1

        WRITING zeroDivision

    @@exit:

ENDM

THIRD\_ACTION MACRO x, y, remainder, result ; (x ^ 2) \* (y ^ 2). [y < 0]

    LOCAL @@loop\_label, @@error, @@exit

    mov ax, x

    mov bx, x

    imul bx ; x ^ 2

    jo @@error

    mov bx, y

    xor cx, cx

    mov cx, 2

    @@label\_loop:

        imul bx

        jo @@error

        loop @@label\_loop

    mov result, ax

    mov remainder, dx

    jmp @@exit

    @@error:

        mov was\_overflow, 1

        WRITING overflowError

    @@exit:

ENDM

WRITE\_RESULT\_WITH\_REMAINDER MACRO resultStr, result, remainderStr, remainder

    WRITING resultStr

    ITOA result

    mov ax, 10

    int 29h

    WRITING remainderStr

    ITOA remainder

ENDM

Комп’ютерний прктикум №3:

include m\_basic.asm

include m\_funcs.asm

; 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

        WRITING xStr

        READING xBuffer

        ; READ Y

        WRITING yStr

        READING yBuffer

        ATOI xBuffer

        cmp was\_overflow, 1

        jne x\_good

        EXIT\_PROGRAM

        x\_good:

        mov x, ax

        ATOI yBuffer

        cmp was\_overflow, 1

        jne y\_good

        EXIT\_PROGRAM

        y\_good:

        mov y, ax

        cmp y, 0

        jg maybe\_first

        je second

        jl toThird

        maybe\_first:

            mov ax, y

            cmp x, ax

            jne first

            EXIT\_PROGRAM

        first:

            FIRST\_ACTION x, y, remainder, result

            cmp was\_overflow, 1

            jne toResultWriting

            EXIT\_PROGRAM

        toResultWriting:

            jmp writing\_output

        toThird:

            jmp third

        second:

            SECOND\_ACTION x, y, remainder, result

            cmp was\_overflow, 1

            jne writing\_output

            EXIT\_PROGRAM

        third:

            THIRD\_ACTION x, y, remainder, result

            cmp was\_overflow, 1

            jne writing\_output

            EXIT\_PROGRAM

        writing\_output:

            WRITE\_RESULT\_WITH\_REMAINDER resultStr, result, remainderStr, remainder

        main\_exit:

            ret

    MAIN ENDP

CSEG ENDS

END MAIN

Макрос для знаходження елемента з матриці:

FIND\_ELEM MACRO x, y, array, elem

    LOCAL find\_2d\_out, find\_2d\_in, find\_skip, find\_2d\_in\_exit, find\_2d\_out\_exit

    mov i, 0

    find\_2d\_out:

        mov j, 0

        find\_2d\_in:

            mov ax, i

            mov bx, x

            mul bx

            add ax, j

            mov bx, ax

            shl bx, 1

            mov ax, array[bx]

            cmp ax, elem

            jne find\_skip

            PRINT\_CHAR '['

            ITOA i

            PRINT\_CHAR ','

            ITOA j

            PRINT\_CHAR ']'

            PRINT\_CHAR 10

            find\_skip:

                inc j

                mov ax, j

                cmp ax, x

                jnl find\_2d\_in\_exit

                jmp find\_2d\_in

        find\_2d\_in\_exit:

        inc i

        mov ax, i

        cmp ax, y

        jnl find\_2d\_out\_exit

        jmp find\_2d\_out

     find\_2d\_out\_exit:

ENDM

Комп’ютерний практикум №4:

include m\_basic.asm

include m\_arrays.asm

; STACK SEGMENT

STSEG SEGMENT PARA STACK "STACK"

    DB 64 DUP (0)

STSEG ENDS

; DATA SEGMENT

DSEG SEGMENT PARA PUBLIC "DATA"

    helloStr db "Enter number of elements (1-20) : $"

    elementStr db "Enter an element: $"

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

    numberOfElementsError db "Invalid number!", 10, "$"

    elementError db "String can not be empty"

    findElemStr db "Enter an element to find : $"

    xStr db "Enter X (1-5): $"

    yStr db "Enter Y (1-5): $"

    arrayStr db "Array is : $"

    sumStr db "Sum is : $"

    minStr db "Min is : $"

    sortedStr db "Sorted array is : $"

    buffer db 6, ?, 6 dup ('?')

    xBuffer db 6, ?, 6 dup ('?')

    yBuffer db 6, ?, 6 dup ('?')

    x dw ?

    y dw ?

    elements dw 20 dup (?)

    twoDArr dw 25 dup (?)

    n dw 0

    i dw 0

    j dw 0

    tmp dw 0

    sum dw 0

    min dw ?

    find dw ?

    was\_overflow db 0

    was\_empty 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

        WRITING helloStr

        READING buffer

        ATOI buffer

        cmp was\_overflow, 1

        je boo

        mov n, ax

        cmp n, 1

        jl boo1

        cmp n, 20

        jg boo1

        mov ax, n

        mov i, ax

        jmp reading\_loop

        boo: jmp foo

        boo1: jmp foo1

        reading\_loop:

            WRITING elementStr

            READING buffer

            ATOI buffer

            cmp was\_overflow, 1

            je foo

            mov bx, n

            sub bx, i

            shl bx, 1

            mov elements[bx], ax

            dec i

            jz reading\_loop\_exit

            jmp reading\_loop

        reading\_loop\_exit:

        WRITING arrayStr

        mov ax, n

        mov i, ax

        jmp output\_array

        foo1: jmp error\_main

        foo: jmp end\_main

        output\_array:

            mov bx, n

            sub bx, i

            shl bx, 1

            mov ax, elements[bx]

            ITOA ax

            PRINT\_CHAR ' '

            dec i

            jnz output\_array

        mov ax, 10

        int 29h

        WRITING sumStr

        call far ptr ARRAY\_SUM

        cmp was\_overflow, 1

        je foo5

        ITOA sum

        PRINT\_CHAR 10

        WRITING minStr

        call far ptr ARRAY\_MIN

        ITOA min

        PRINT\_CHAR 10

        jmp notfoo5

        foo5: jmp foo4

        notfoo5:

        WRITING sortedStr

        call far ptr ARRAY\_SORT

        mov ax, n

        mov i, ax

        jmp output\_array1

        foo4: jmp foo3

        output\_array1:

            mov bx, n

            sub bx, i

            shl bx, 1

            mov ax, elements[bx]

            ITOA ax

            PRINT\_CHAR ' '

            dec i

            jnz output\_array1

        PRINT\_CHAR 10

        WRITING xStr

        READING xBuffer

        WRITING yStr

        READING yBuffer

        ATOI xBuffer

        cmp was\_overflow, 1

        je foo6

        mov x, ax

        ATOI yBuffer

        cmp was\_overflow, 1

        je foo3

        jmp notfoo6

        foo6: jmp foo3

        notfoo6:

        mov y, ax

        cmp x, 1

        jl foo3

        cmp x, 5

        jg foo3

        cmp y, 1

        jl foo3

        cmp y, 5

        jg foo3

        mov i, 0

        jmp reading\_2d\_out

        foo3: jmp foo2

        reading\_2d\_out:

            mov j, 0

            reading\_2d\_in:

                PRINT\_CHAR '['

                ITOA i

                PRINT\_CHAR ','

                ITOA j

                PRINT\_CHAR ']'

                PRINT\_CHAR ' '

                READING buffer

                ATOI buffer

                cmp was\_overflow, 1

                je foo2

                mov tmp, ax

                mov ax, i

                mov bx, x

                mul bx

                add ax, j

                mov bx, ax

                shl bx, 1

                mov ax, tmp

                mov twoDArr[bx], ax

                inc j

                mov ax, j

                cmp ax, x

                jnl reading\_2d\_in\_exit

                jmp reading\_2d\_in

            reading\_2d\_in\_exit:

            inc i

            mov ax, i

            cmp ax, y

            jnl reading\_2d\_out\_exit

            jmp reading\_2d\_out

        reading\_2d\_out\_exit:

        mov i, 0

        jmp writing\_2d\_out

        foo2: jmp end\_main

        writing\_2d\_out:

            mov j, 0

            writing\_2d\_in:

                mov ax, i

                mov bx, x

                mul bx

                add ax, j

                mov bx, ax

                shl bx, 1

                mov ax, twoDArr[bx]

                ITOA ax

                PRINT\_CHAR ' '

                inc j

                mov ax, j

                cmp ax, x

                jl writing\_2d\_in

            mov ax, 10

            int 29h

            inc i

            mov ax, i

            cmp ax, y

            jl writing\_2d\_out

        WRITING findElemStr

        READING buffer

        ATOI buffer

        cmp was\_overflow, 1

        je foo7

        mov find, ax

        jmp find\_elem\_label

        foo7: jmp end\_main

        find\_elem\_label:

        FIND\_ELEM x, y, twoDArr, find

        jmp end\_main

        error\_main:

            WRITING numberOfElementsError

        end\_main:

            ret

    MAIN ENDP

    ARRAY\_SORT PROC FAR

    out\_loop\_sort:

        mov ax, i

        mov min, ax

        mov j, ax

        inc j

        mov ax, n

        cmp j, ax

        jge sort\_end

        inner\_loop\_sort:

            mov bx, min

            shl bx, 1

            mov ax, elements[bx]

            mov bx, j

            shl bx, 1

            mov cx, elements[bx]

            cmp ax, cx

            jng skip\_update

            mov ax, j

            mov min, ax

            skip\_update:

            inc j

            mov ax, j

            cmp ax, n

            jl inner\_loop\_sort

        mov bx, i

        shl bx, 1

        mov ax, elements[bx]

        mov bx, min

        shl bx, 1

        mov cx, elements[bx]

        mov dx, cx

        mov bx, min

        shl bx, 1

        mov elements[bx], ax

        mov bx, i

        shl bx, 1

        mov elements[bx], dx

        inc i

        mov ax, n

        cmp i, ax

        jl out\_loop\_sort

        sort\_end:

            ret

    ENDP ARRAY\_SORT

    ARRAY\_SUM PROC FAR

        mov ax, n

        mov i, ax

        sum\_loop:

            mov bx, n

            sub bx, i

            shl bx, 1

            mov ax, elements[bx]

            add sum, ax

            jo error\_sum

            dec i

            jnz sum\_loop

        jmp sum\_end

        error\_sum:

            mov was\_overflow, 1

            WRITING overflowError

        sum\_end:

            ret

    ENDP ARRAY\_SUM

    ARRAY\_MIN PROC FAR

        mov ax, elements[0]

        mov min, ax

        mov ax, n

        mov i, ax

        min\_loop:

            mov bx, n

            sub bx, i

            shl bx, 1

            mov ax, elements[bx]

            cmp ax, min

            jg iterate\_min

            mov min, ax

            iterate\_min:

                dec i

                jnz min\_loop

        ret

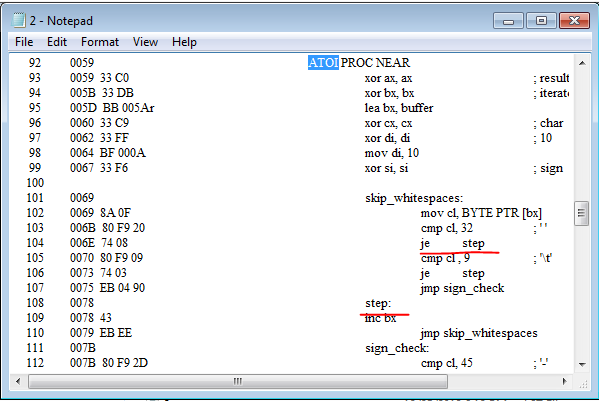
    ENDP ARRAY\_MIN

CSEG ENDS

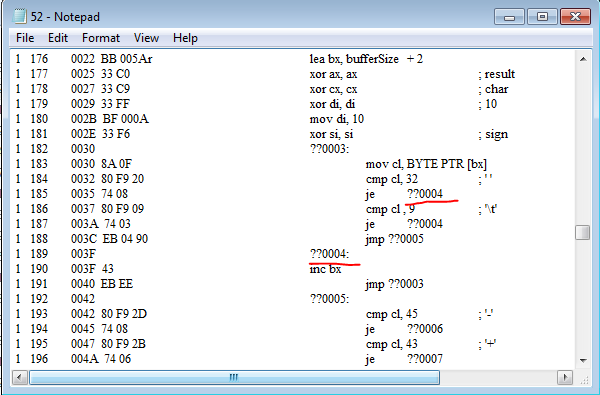
END MAIN

За допомогою lst-файлу можна пересвідчитись у коректності коду для макросів з використанням локальних міток. Наприклад, тут можна помітити, що тут у макросі ATOI мітка step в двох випадках різна.

Мітка в процедурі з КП2:



Мітка в макросі з КП5:



Висновки:

1. Був засвоєний механізм роботи з макрозасобами мови асемблер:

1.1. Були створені макроси, що усувають потребу повторювати ідентичний або схожий код;

1.2. Були написані макроси, що приймають аргументи;

1.3. Усі макроси були написані з використанням локальних міток;

1.4. Базові макроси були зібрані у один файл, що можна використовувати у інших програмах.

1. Попередні КП були переписані з використанням макросів;
2. Був засвоєний принцип роботи директиви include