НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіти до комп'ютерних практикумів з дисципліни

«Системне програмне забезпечення»

Прийняв:
Викладач кафедри ІПІ
Стельмах О. П.
18 грудня 2021 року

Виконав:
Студент групи ІТ-01
Бардін В. Д.

Київ – 2021

Комп’ютерний практикум №1

Тема: Створення програм на асемблері

Завдання:

1. Для програми, наведеної вище, створити файл типу .asm. Ця програма не має засобів виводу даних, тому правильність її виконання треба перевірити за допомогою td.exe.
2. Скомпілювати програму, включивши потрібні опції для налагоджувача та створення файлу лістингу типу .lst.
3. Ознайомитись зі структурою файлу .lst. За вказівкою викладача, для певної команди асемблера розглянути структуру машинної команди і навести її у звіті.
4. Скомпонувати .obj-файл програми. Включити опції для налагодження та створення .mapфайлу.
5. Занести до звіту адреси початку та кінця всіх сегментів з .map-файлу. 6. Завантажити до налагоджувача td.exe одержаний .exe-файл програми.
7. У вікні CPU у полі DUMP знайти початкову адресу сегмента даних та записати його до звіту. Знайти масиви SOURCE та DEST. Дані у масиві SOURCE подаються у шістнадцятковій системі.
8. У покроковому режимі за допомогою клавіші F7 виконати програму. Одержані результати у масиві DEST показати викладачеві.

Текст програми:

```
STSEG SEGMENT PARA STACK "STACK"
DB 64 DUP ( "STACK" )
STSEG ENDS
DSEG SEGMENT PARA PUBLIC "DATA"
SOURCE DB 10, 20, 30, 40
DEST DB 4 DUP ( "?" )
DSEG ENDS
CSEG SEGMENT PARA PUBLIC "CODE"
MAIN PROC FAR
ASSUME CS: CSEG, DS: DSEG, SS: STSEG
; адреса повернення
PUSH DS
MOV AX, 0 ; або XOR AX, AX
PUSH AX
; ініціалізація DS
MOV AX, DSEG
MOV DS, AX
; обнуляємо масив
MOV DEST, 0
MOV DEST+1, 0
MOV DEST+2, 0
MOV DEST+3, 0
; пересилання
MOV AL, SOURCE
MOV DEST+3, AL
MOV AL, SOURCE+1
MOV DEST+2, AL
```

```
MOV AL, SOURCE+2
MOV DEST+1, AL
MOV AL, SOURCE+3
MOV DEST, AL
RET
MAIN ENDP
CSEG ENDS
END MAIN
```

Введені та отримані результати:

Вміст .lst файлу:

```
Turbo Assembler     Version 3.2     09/06/21 12:13:28         Page 1

labs\cp1.asm

      1      0000                    STSEG SEGMENT PARA   STACK "STACK"

      2      0000  40*(53 54     41 43 4B)    DB     64 DUP ( "STACK" )

      3      0140                    STSEG ENDS

      4      0000                    DSEG SEGMENT PARA PUBLIC "DATA"

      5      0000  0A 14 1E 28       SOURCE DB 10, 20, 30, 40

      6      0004  04*(3F)                 DEST DB 4 DUP ( "?" )

      7      0008                    DSEG ENDS

      8      0000                    CSEG SEGMENT PARA PUBLIC "CODE"

      9      0000                    MAIN PROC FAR

     10                             ASSUME CS:    CSEG, DS: DSEG,    SS: STSEG

     11                             ; адреса повернення

     12      0000  1E                PUSH DS

     13      0001  B8 0000                 MOV AX, 0 ; або      XOR AX,      AX

     14      0004  50                PUSH AX

     15                             ; ініціалізація DS

     16      0005  B8 0000s                MOV AX, DSEG

     17      0008  8E D8             MOV DS, AX

     18                             ; обнуляємо масив

     19      000A  C6 06 0004r 00          MOV DEST, 0

     20      000F  C6 06 0005r 00          MOV DEST+1, 0

     21      0014  C6 06 0006r 00          MOV DEST+2, 0
```

```
     22      0019  C6 06 0007r 00          MOV DEST+3, 0
     23                                ; пересилання
     24      001E  A0 0000r                MOV AL, SOURCE
     25      0021  A2 0007r                MOV DEST+3, AL
     26      0024  A0 0001r                MOV AL, SOURCE+1
     27      0027  A2 0006r                MOV DEST+2, AL
     28      002A  A0 0002r                MOV AL, SOURCE+2
     29      002D  A2 0005r                MOV DEST+1, AL
     30      0030  A0 0003r                MOV AL, SOURCE+3
     31      0033  A2 0004r                MOV DEST, AL
     32      0036  CB                RET
     33      0037                    MAIN ENDP
     34      0037                    CSEG ENDS
     35                              END MAIN
Turbo Assembler     Version 3.2     09/06/21 12:13:28        Page 2
Symbol Table
Symbol Name                Type  Value
??DATE                     Text  "09/06/21"
??FILENAME                 Text  "cp1    "
??TIME                     Text  "12:13:28"
??VERSION                  Number 0314
@CPU                       Text  0101H
@CURSEG                          Text  CSEG
@FILENAME                  Text  CP1
@WORDSIZE                  Text  2
DEST                       Byte  DSEG:0004
MAIN                       Far   CSEG:0000
SOURCE                     Byte  DSEG:0000
Groups & Segments          Bit Size Align  Combine Class
CSEG                       16  0037 Para      Public  CODE
DSEG                       16  0008 Para      Public  DATA
```

```
STSEG                       16   0140 Para      Stack        STACK
```

Вміст .map файлу:

```
 Start  Stop   Length Name               Class

 00000H 0013FH 00140H STSEG              STACK

 00140H 00147H 00008H DSEG               DATA

 00150H 00186H 00037H CSEG               CODE

Program entry point at 0015:0000
```

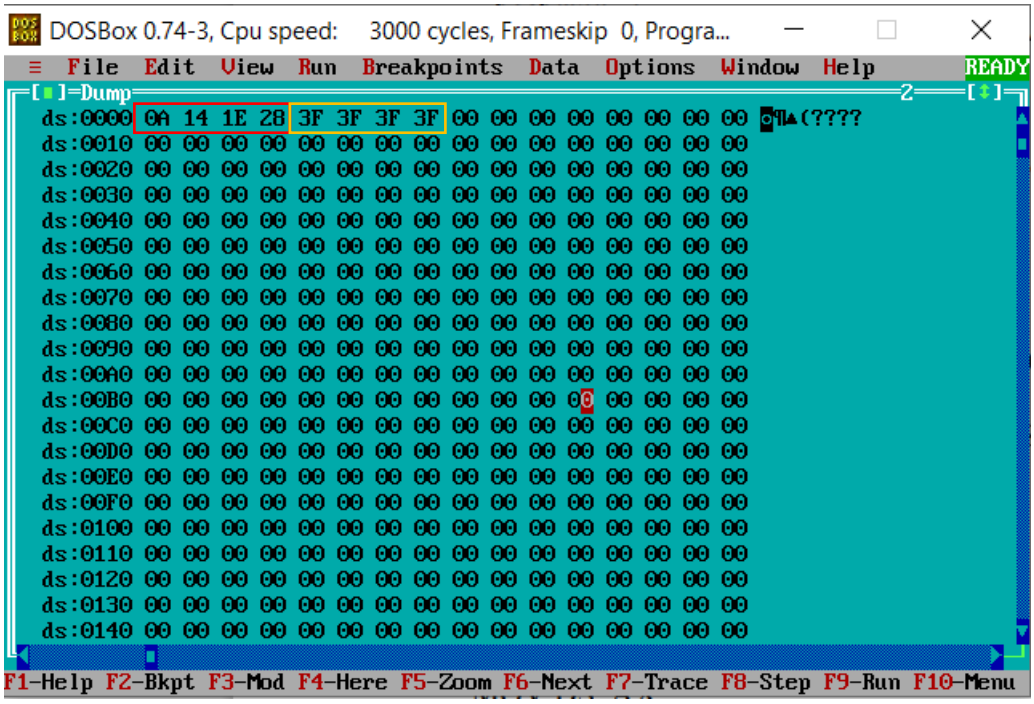
Схема функціонування програми:

Процес компілювання та лінкування програми:

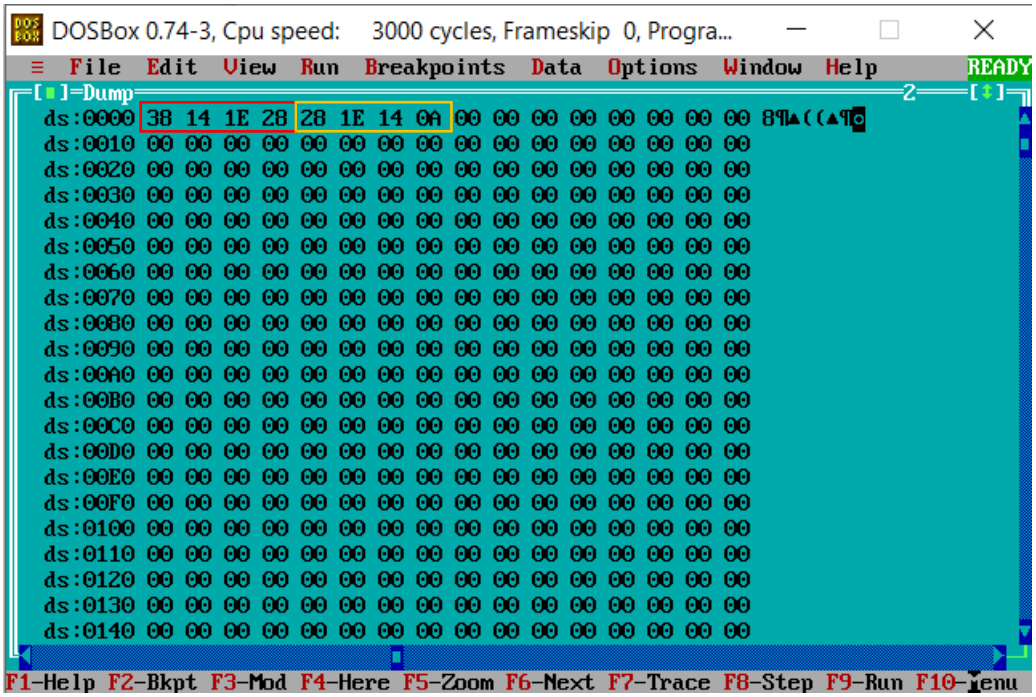
```
Z:\>d:

D:\>set path=c:\tasm

D:\>tasm /zi cp1.ASM >X:\ASM.LOG

Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   cp1.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  465k


D:\>if exist cp1.OBJ tlink /v/3 cp1.obj >X:\LINK.LOG

Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
```

Вікно турбодебагеру:

Дамп пам’яті після ініціалізації:



Дамп пам’яті після виконання програми:



Червоний — це масив SOURCE, жовтий — DEST.

Висновок:

Підчас виконання лабораторної роботи я створив .asm файл, скомілював і злінкував його. Також розібрався, як формуються та яку інформацію містять .map та .lst файли. Також переглянув як працює TurboDebagger.

Комп’ютерний практикум №2

Тема: Засоби обміну даними

Завдання:

- Написати процедуру введення і перетворення цілого числа.
- Виконати математичну дію над числом: +67.
- Перевести число в рядок та вивести його на екран.

Код застосунку:

```
IDEAL
MODEL SMALL
STACK 512

DATASEG;III.ПОЧАТОК СЕГМЕНТУ ДАНИХ

buffer db 254
error_message DB "number out of bounds!", 13, 10, '$'
result_buffer DB "result:       ", 13, 10, '$'

CODESEG
Start:

; адреса повернення
; ініціалізація DS
    mov ax,@data; @data ідентифікатор, що створюються директивою model
    mov ds, ax  ; Завантаження початку сегменту даних в регістр ds
    mov es, ax  ; Завантаження початку сегменту даних в регістр es

    call input_sdec_word

    mov bx, [ds:0002] ;занесення в ax чисельного значення
    ;символу ASCII, що відповідає
    ;знаку,
    ;який введено з клавіатури
    ;cmp
    cmp bl, 02Dh ; c ascii = 2Dh ; Вибір відповідної функції
    je negative;
    jmp positive;

negative:
    cmp ax,67           ;Модуль отрицательного числа должен быть не больше 34
    ja negative_large
    jmp negative_small
positive:
    call math_1
    ;mov dx, offset display_message_1 ; Закоментовані повідомлення у ході
налаштування
    ;call display
    jmp exit
```

```
negative_large:
    call math_2
    ;mov dx, offset display_message_1 ; Закоментовані повідомлення у ході
налаштування
    ;call display
    jmp exit
negative_small:
    call math_3
    ;mov dx, offset display_message_1 ; Закоментовані повідомлення у ході
налаштування
    ;call display
    jmp exit
exit:
    mov ah,04Ch
    mov al,0 ; отримання коду виходу
    int 21h ; виклик функції DOS 4ch

PROC display
 mov ah,9
 int 21h
 xor dx, dx
 ret
ENDP display


PROC input_str
    push cx                 ;Сохранение CX
    mov cx,ax               ;Сохранение AX в CX
    mov ah,0Ah              ;Функция DOS 0Ah - ввод строки в буфер
    mov [buffer],al         ;Запись максимальной длины в первый байт буфера
    mov [buffer+1],0    ;Обнуление второго байта (фактической длины)
    mov dx,offset buffer            ;DX = адрес буфера
    int 21h                 ;Обращение к функции DOS
    mov al,[buffer+1]       ;AL = длина введённой строки
    add dx,2                ;DX = адрес строки
    mov ah,ch               ;Восстановление AH
    pop cx                  ;Восстановление CX
    ret
ENDP

;Процедура ввода слова с консоли в десятичном виде (со знаком)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
PROC input_sdec_word
    push dx                 ;Сохранение DX
    mov al,7                ;Ввод максимум 7 символов (-32768) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_sdec_word   ;Преобразование строки в слово (со знаком)
    pop dx                  ;Восстановление DX
    ret
ENDP
```

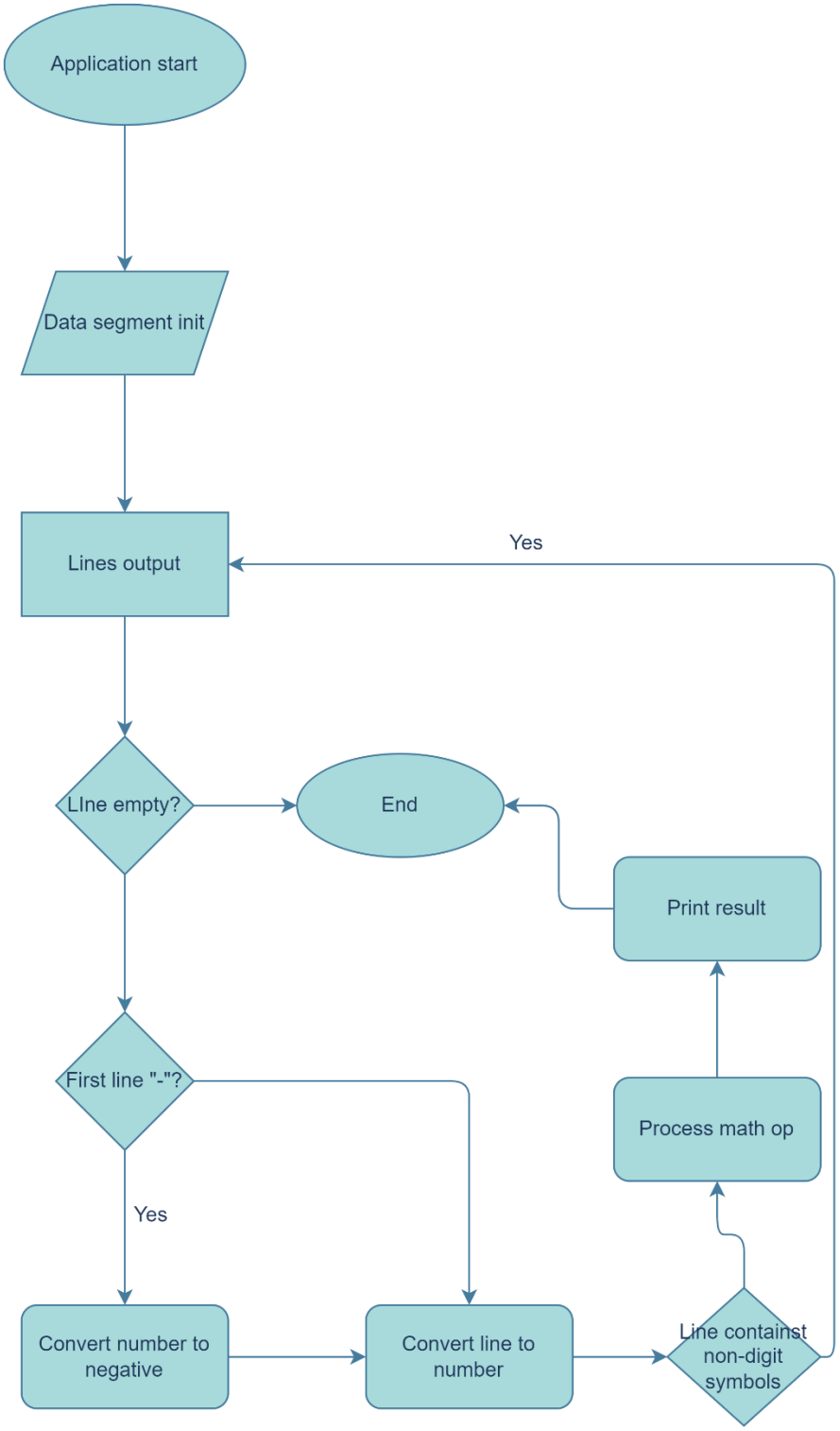
```
;Процедура преобразования десятичной строки в слово со знаком
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
PROC str_to_sdec_word
    push bx                 ;Сохранение регистров
    push dx

    test al,al              ;Проверка длины строки
    jz stsdw_error          ;Если равно 0, возвращаем ошибку
    mov bx,dx               ;BX = адрес строки
    mov bl,[bx]             ;BL = первый символ строки
    cmp bl,'-'              ;Сравнение первого символа с '-'
    jne stsdw_no_sign       ;Если не равно, то преобразуем как число без знака
    inc dx                  ;Инкремент адреса строки
    dec al                  ;Декремент длины строки
stsdw_no_sign:
    call str_to_udec_word   ;Преобразуем строку в слово без знака
    jc stsdw_error          ;Если ошибка, то возвращаем ошибку
    cmp bl,'-'              ;Снова проверяем знак
    jne stsdw_plus          ;Если первый символ не '-', то число положительное
    cmp ax,32734            ;Модуль отрицательного числа должен быть не больше
32768 - 34
    ja stsdw_error          ;Если больше (без знака), возвращаем ошибку
;    neg ax                  ;Инвертируем число
    jmp stsdw_ok            ;Переход к нормальному завершению процедуры
stsdw_plus:
    cmp ax,32767            ;Положительное число должно быть не больше 32767
    ja stsdw_error          ;Если больше (без знака), возвращаем ошибку

stsdw_ok:
    clc                     ;CF = 0
    jmp stsdw_exit          ;Переход к выходу из процедуры
stsdw_error:
    mov dx, offset error_message ;; Закоментовані повідомлення у ході
налаштування
    call display
    xor ax,ax               ;AX = 0
    ;stc                     ;CF = 1 (Возвращаем ошибку
    ;jmp stsdw_exit          ;Переход к выходу из процедуры
    jmp exit
stsdw_exit:
    pop dx                  ;Восстановление регистров
    pop bx
    ret
ENDP

;Процедура преобразования десятичной строки в слово без знака
;  вход: AL - длина строки
```

```
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
PROC str_to_udec_word
    push cx                 ;Сохранение всех используемых регистров
    push dx
    push bx
    push si
    push di

    mov si,dx               ;SI = адрес строки
    mov di,10               ;DI = множитель 10 (основание системы счисления)
    mov cl,al             ;CX = счётчик цикла = длина строки     movzx
cx,al             ;CX = счётчик цикла = длина строки
    jcxz studw_error        ;Если длина = 0, возвращаем ошибку
    xor ax,ax               ;AX = 0
    xor bx,bx               ;BX = 0

studw_lp:
    mov bl,[si]             ;Загрузка в BL очередного символа строки
    inc si                  ;Инкремент адреса
    cmp bl,'0'              ;Если код символа меньше кода '0'
    jl studw_error          ; возвращаем ошибку
    cmp bl,'9'              ;Если код символа больше кода '9'
    jg studw_error          ; возвращаем ошибку
    sub bl,'0'              ;Преобразование символа-цифры в число
    mul di                  ;AX = AX * 10
    jc studw_error          ;Если результат больше 16 бит - ошибка
    add ax,bx               ;Прибавляем цифру
    jc studw_error          ;Если переполнение - ошибка
    loop studw_lp           ;Команда цикла
    jmp studw_exit          ;Успешное завершение (здесь всегда CF = 0)

studw_error:
    xor ax,ax               ;AX = 0
    stc                     ;CF = 1 (Возвращаем ошибку)

studw_exit:
    pop di                  ;Восстановление регистров
    pop si
    pop bx
    pop dx
    pop cx
    ret
ENDP

PROC math_1
    add ax,67     ;AX = AX + 34
    mov bx, offset result_buffer
    call output
    ret
```

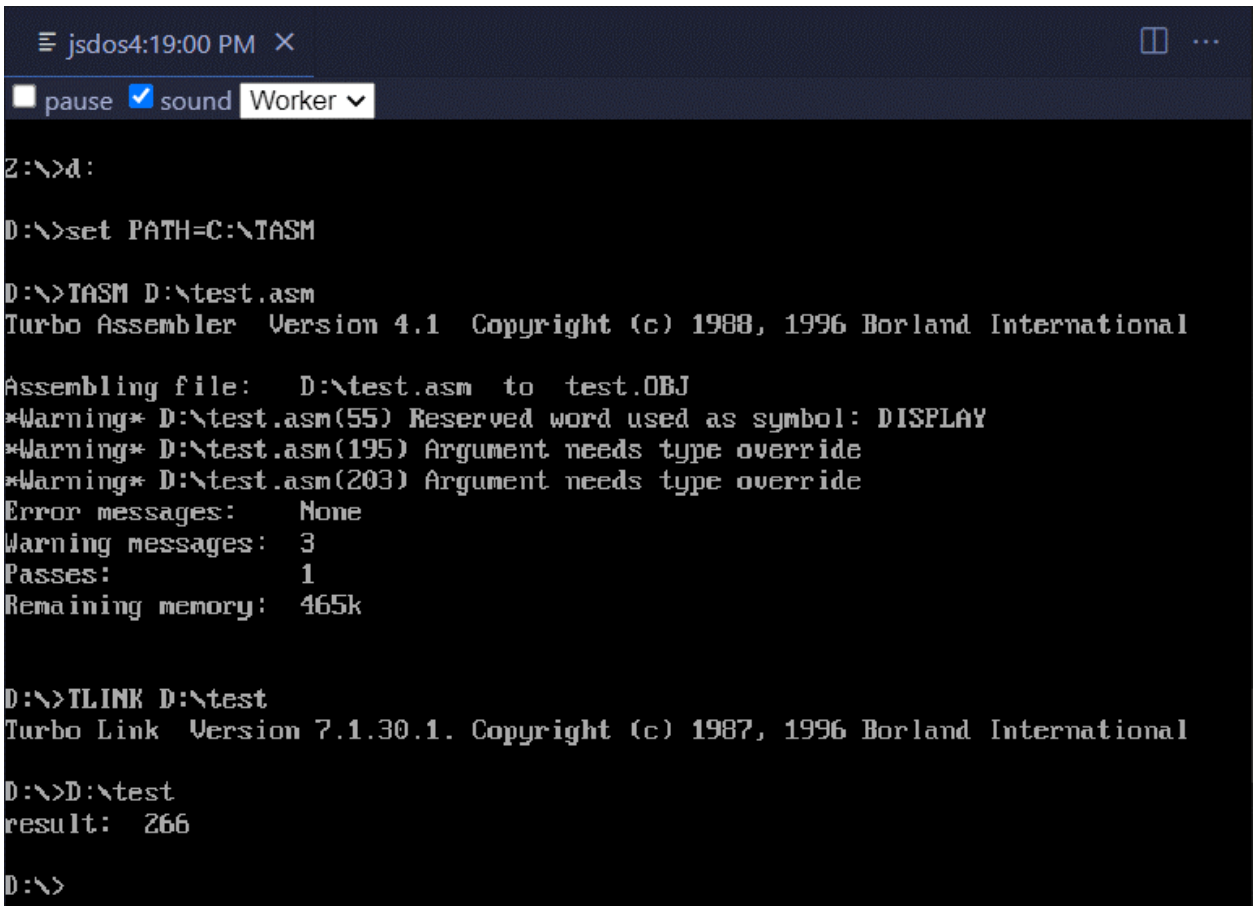
```
ENDP math_1

PROC math_3
    mov bx, ax
    mov ax, 67
    sub ax, bx
    mov bx, offset result_buffer
    mov [bx+8], ' '
    call output
    ret
ENDP math_3

PROC math_2
    sub ax,67     ;AX = AX + 34
    mov bx, offset result_buffer
    mov [bx+8], '-'
    call output
    ret
ENDP math_2


PROC output
        ;ax - число
    mov di,offset result_buffer    ;es:di - адрес буфера приемника
    mov cx, 9
    loop1:
        INC di
        loop loop1
    push cx ;сохраняем регистры
    push dx
    push bx
    mov bx,10   ;основание системы
    XOR CX,CX   ;в cx будет количество цифр в десятичном числе
m1:   XOR dx,dx
    DIV bx      ;делим число на степени 10
    PUSH DX     ;и сохраняем остаток от деления(коэффициенты при степенях) в стек
    INC CX
    TEST AX,AX
    JNZ m1
m2:   POP AX
    ADD AL,'0'  ;преобразовываем число в ASCII символ
    STOSb       ;сохраняем в буфер
    LOOP m2       ;все цифры
    pop bx      ;восстанавливаем регистры
    POP dx
    POP cx

    mov dx, offset result_buffer ; Закоментовані повідомлення у ході налаштування
    call display
    call exit
ret
```

```
ENDP output
end Start
```

Блок-схема:



Скріншот консолі DOS box-у, де показано компоновку та лінкування програми, а також результат запуску програми.



Висновок:

У цій ЛР було розроблено просту програму, яка вміє робити прості математичні дії над числоми.

Комп’ютерний практикум №3

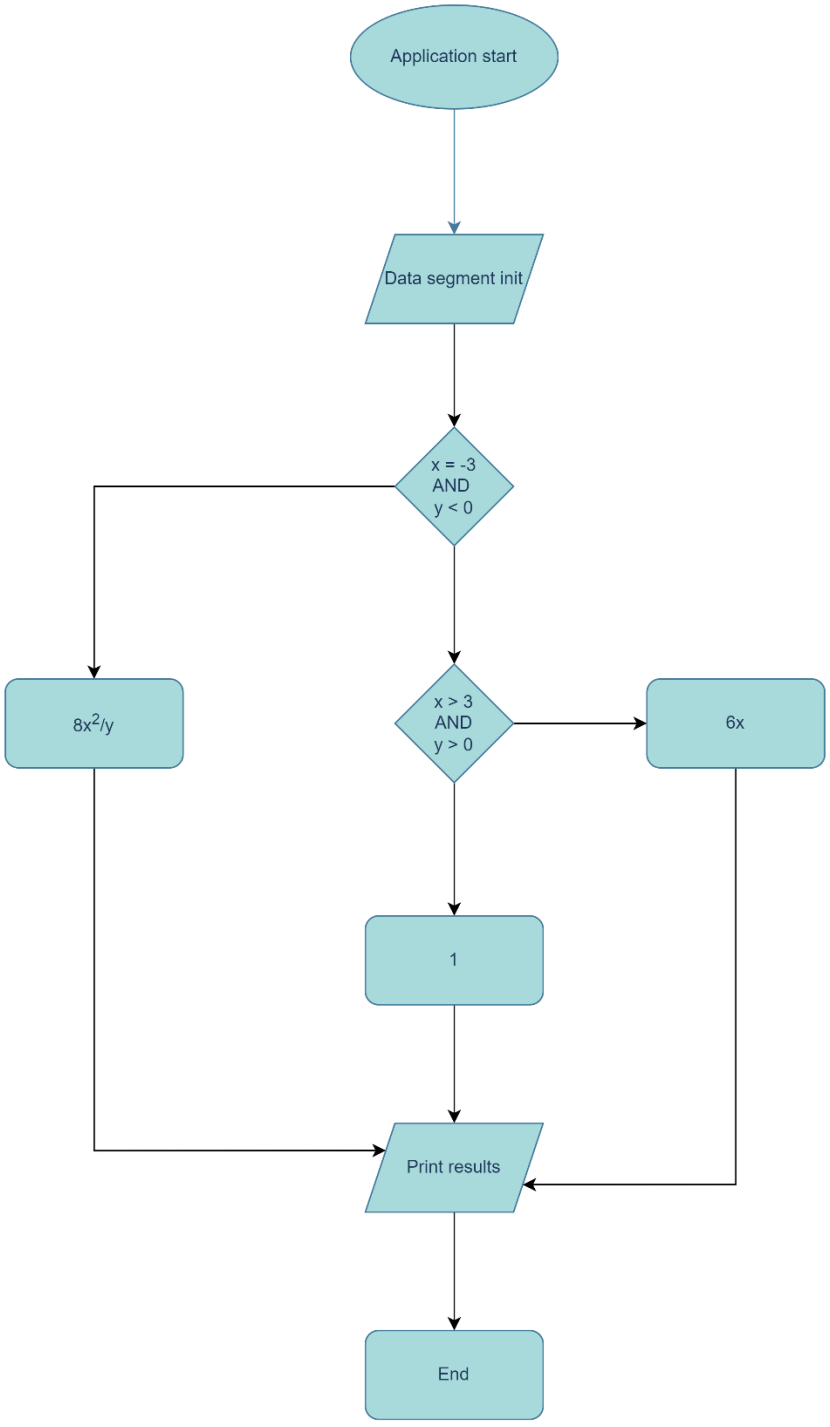
Тема: Програмування розгалужених алгоритмів

Завдання:

Написати програму, яка буде обчислювати значення функції.

1. $$Z=\begin{cases} 8x^2/y & \text{якщо } x = -5,\ y < 0 \\ 6x & \text{якщо } x > 3,\ y > 0 \\ 1 & \text{в інших випадках} \end{cases}$$

Блок-Схема:



Код застосунку:

```
IDEAL
MODEL SMALL
STACK 512


MACRO main_read
; Початок макросу
    MOV bx, [ds:si] ;занесення в ax чисельного значення
    MOV bl, bh
    CMP bl, 030h ; c ascii = 2SDh ; Вибір відповідної функції
    jl exit22;
    CMP bl, 039h ; c ascii = 2SDh ; Вибір відповідної функції
    ja exit22;
    MOV bh, '0'
ENDM main_read



DATASEG;III.ПОЧАТОК СЕГМЕНТУ ДАНИХ


X DB "             ", 13, 10, '$'
Y DB "             ", 13, 10, '$'

out_of_bounds DB "incorrect number!", 13, 10, '$'
final_message DB "result:       ", 13, 10, '$'

input1 DB "please, input X", 13, 10, '$'
input2 DB "please, input Y", 13, 10, '$'
result_out_of_bounds DB "result out of bounds!", 13, 10, '$'
final_message_rest DB "rest:             ", 13, 10, '$'

CODESEG
Start:

    MOV ax, @data
    MOV ds, ax
    MOV es, ax

    MOV dx, offset input1 ;; Закоментовані повідомлення у ході налаштування
    MOV ah,9
    INT 21h
    XOR dx, dx


    MOV dx, offset X
    PUSH dx
    MOV al,7
    PUSH cx
    MOV cx,ax
```

```
    MOV ah,0Ah              ;ввод строки в буфер
    MOV [X],al
    MOV [X+1],0
    MOV dx,offset X           ;DX = адрес буфера
    INT 21h
    MOV al,[X+1]       ;AL = длина введённой строки
    add dx,2                ;DX = адрес строки
    MOV ah,ch               ;Восстановление AH
    POP cx
    CALL str_transformation   ;Преобразование строки в слово (со знаком)
    MOV dx, offset input2 ;; Закоментовані повідомлення у ході налаштування

    MOV ah,9
    INT 21h
    XOR dx, dx
    MOV dx, offset Y
    MOV al,7
    PUSH cx
    MOV cx,ax
    MOV ah,0Ah              ;ввод строки в буфер
    MOV [Y],al
    MOV [Y+1],0
    MOV dx,offset Y           ;DX = адрес буфера
    INT 21h
    MOV al,[Y+1]       ;AL = длина введённой строки
    add dx,2                ;DX = адрес строки
    MOV ah,ch               ;Восстановление AH
    POP cx
    CALL str_transformation   ;Преобразование строки в слово (со знаком)
    POP dx


    MOV si, 1Bh
    call read_number
    CMP ax, 0
    JE general

    MOV si, 0Bh
    call read_number

    MOV bx,[ds:02h]              ;BL = первый символ строки
    CMP bl,'-'              ;Сравнение первого символа с '-'
    JE negative
    JMP positive
negative:
    CMP ax, 5
    JE f_1
    JMP general
f_1:
    MOV bx,[ds:12h]              ;BL = первый символ строки
    CMP bl,'-'
```

```
    JE case1
    JMP general
positive:
    CMP ax, 3
    JG f_2
    JMP general
f_2:
    MOV bx,[ds:12h]               ;BL = первый символ строки
    CMP bl,'-'
    JE general
    JMP case2
general:
    MOV ax, 1
    CALL print
    JMP exit
case1:
    MOV si, 1Bh
    CALL read_number
    MOV bx, ax
    MOV ax, 200
    DIV bx

    MOV bx, offset final_message
    MOV [bx+8], '-'
    CALL print
    CALL print_rest
    CALL exit
case2:
    MOV bx, 6
    MUL bx

    CMP dx, 0
    JG error_11
    JMP skip_error_11
error_11:
    MOV dx, offset result_out_of_bounds
    MOV ah,9
    INT 21h
    XOR dx, dx
    CALL exit
skip_error_11:
    CALL print
    CALL exit

exit:
    MOV ah,04Ch
    MOV al,0 ; отримання коду виходу
    INT 21h ; виклик функції DOS 4ch
```

```
PROC save_input
    PUSH cx                 ;Сохранение всех используемых регистров
    PUSH dx
    PUSH bx
    PUSH si
    PUSH di

    MOV si,dx               ;SI = адрес строки
    MOV di,10               ;DI = множитель 10 (основание системы счисления)
    MOV cl,al             ;CX = счётчик цикла = длина строки     movzx
cx,al             ;CX = счётчик цикла = длина строки
    JCXZ error_end          ;Если длина = 0, возвращаем ошибку
    XOR ax,ax               ;AX = 0
    XOR bx,bx               ;BX = 0

loop21:
    MOV bl,[si]             ;Загрузка в BL очередного символа строки
    INC si                  ;Инкремент адреса
    CMP bl,'0'              ;Если код символа меньше кода '0'
    JL error_end          ; возвращаем ошибку
    CMP bl,'9'              ;Если код символа больше кода '9'
    JG error_end          ; возвращаем ошибку
    SUB bl,'0'              ;Преобразование символа-цифры в число
    MUL di                  ;AX = AX * 10
    JC error_end          ;Если результат больше 16 бит - ошибка
    ADD ax,bx               ;Прибавляем цифру
    JC error_end          ;Если переполнение - ошибка
    LOOP loop21           ;Команда цикла
    JMP exit21          ;Успешное завершение (здесь всегда CF = 0)

error_end:
    XOR ax,ax               ;AX = 0
    STC                     ;CF = 1 (Возвращаем ошибку)

exit21:
    POP di                  ;Восстановление регистров
    POP si
    POP bx
    POP dx
    POP cx
    ret
ENDP

PROC read_error
    MOV dx, offset out_of_bounds ;; Закоментовані повідомлення у ході
налаштування
    MOV ah,9
    INT 21h
    XOR dx, dx
    XOR ax,ax               ;AX = 0
    CALL exit
```

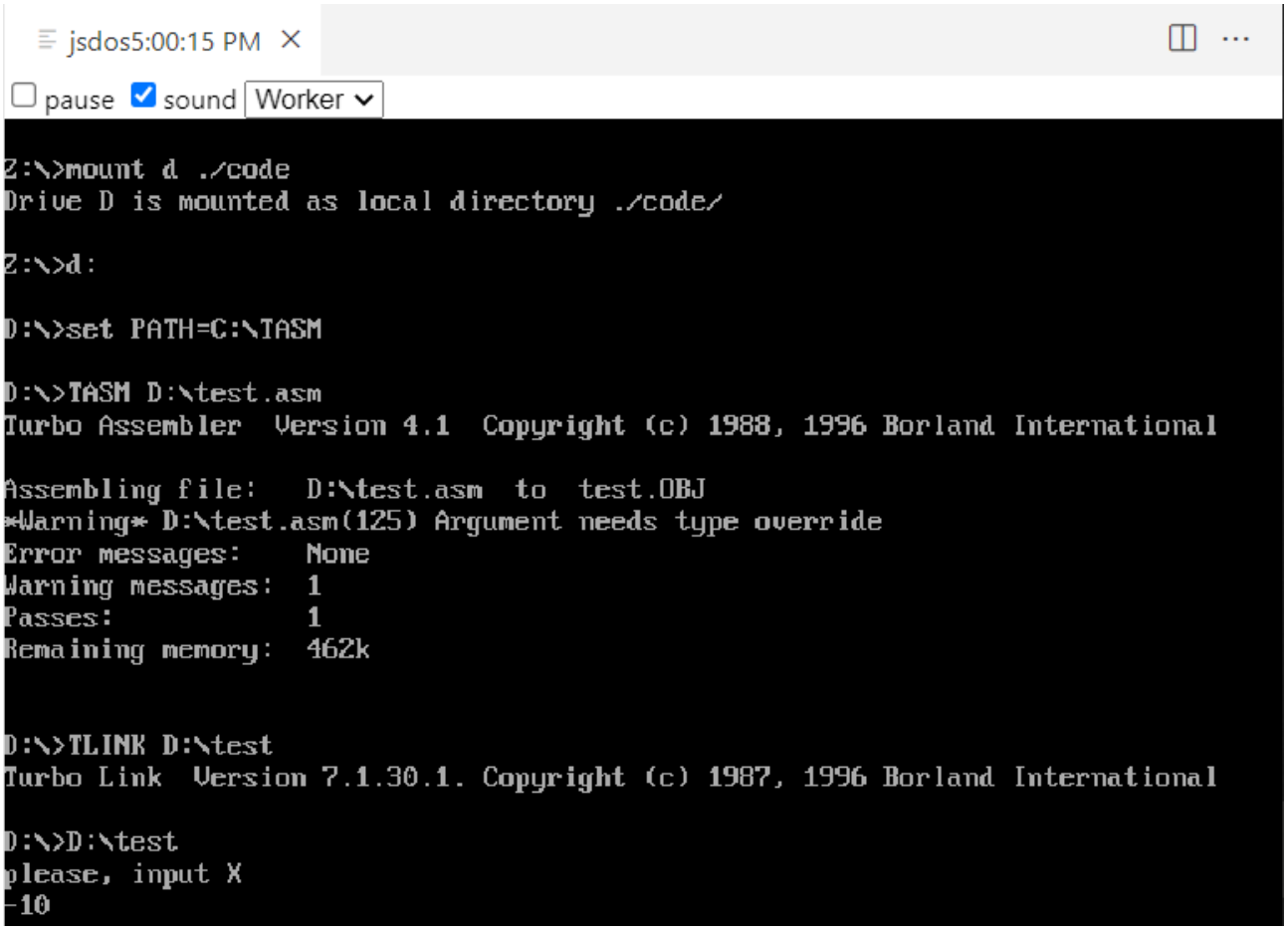
```
    RET
ENDP
PROC read_number
    PUSH cx
    PUSH dx
    PUSH bx
    XOR ax, ax
loop1:
    MOV bx, [ds:si] ;занесення в ax чисельного значення
    MOV bl, bh
    XOR bh,bh
    CMP bl, 030h ; с ascii = 2SDh ; Вибір відповідної функції
    jl skip1;
    CMP bl, 039h ; с ascii = 2SDh ; Вибір відповідної функції

    main_read
    sub bl, bh
    MOV al,bl
    dec si

    main_read
    sub bl, bh
    MOV cx, ax
    MOV ax, 10
    XOR bh,bh
    MUL bx
    add ax, cx
    dec si

    main_read
    sub bl, bh
    MOV cx, ax
    MOV ax, 100
    XOR bh,bh
    MUL bx
    add ax, cx
    dec si

    main_read
    sub bl, bh
    MOV cx, ax
    MOV ax, 1000
    XOR bh,bh
    MUL bx
    add ax, cx
    dec si
    JMP last_check

skip1:
    dec si
    JMP loop1
```

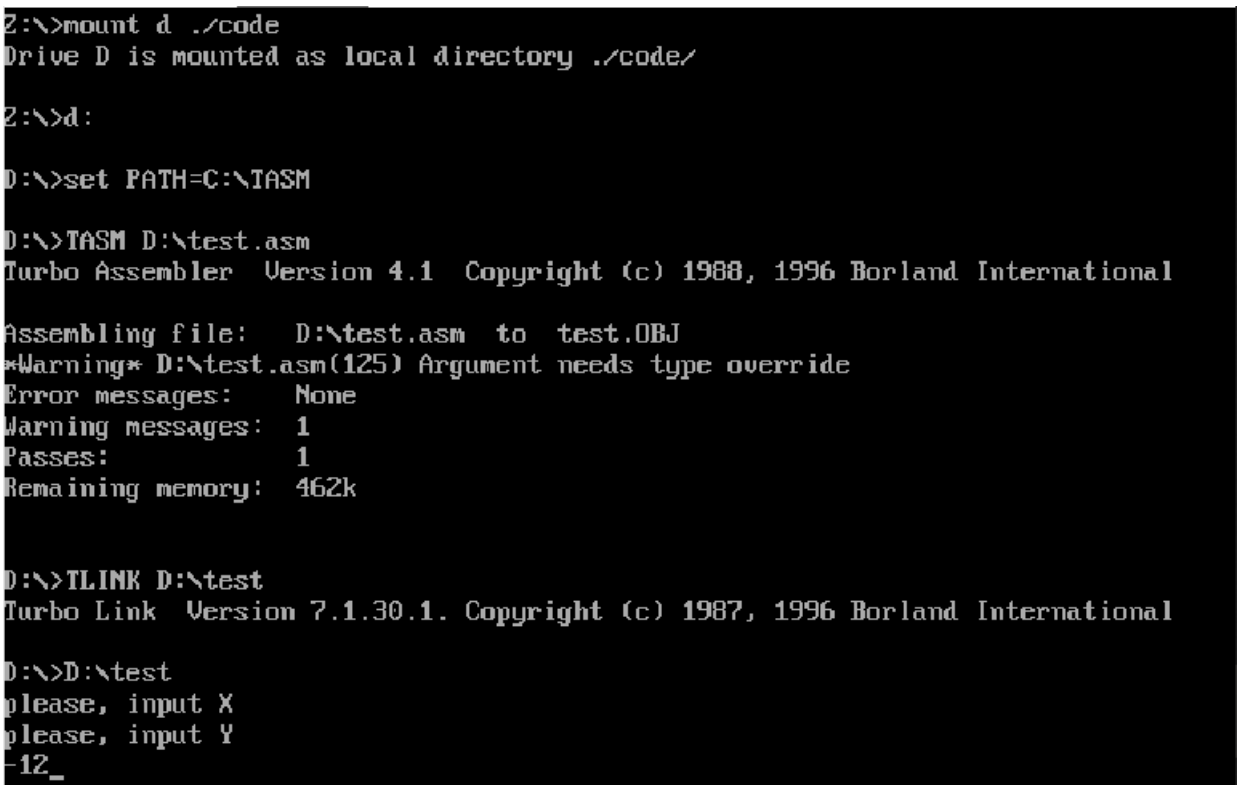
```
exit22:
XOR dx, dx
POP bx
POP dx
POP cx
    RET
last_check:
    main_read
    sub bl, bh
    MOV cx, ax
    MOV ax, 10000
    XOR bh,bh
    MUL bx
    add ax, cx
    dec si
    JMP exit22
ENDP


PROC str_transformation
    PUSH bx
    PUSH dx

    test al,al              ;Проверка длины строки
    jz f2
    MOV bx,dx
    MOV bl,[bx]
    CMP bl,'-'              ;проверка знака
    jne f1
    inc dx
    dec al
f1:
    CALL save_input
    jc f2
    CMP bl,'-'
    jne f3          ;Если первый символ не '-', то число положительное
    CMP ax,32734            ;проверка модуля числа
    ja f2
    JMP f4
f3:
    CMP ax,32767            ;проверка модуля числа
    ja f2
f4:
    clc
    JMP f5
f2:
    CALL read_error
f5:
    POP dx                  ;Восстановление регистров
```

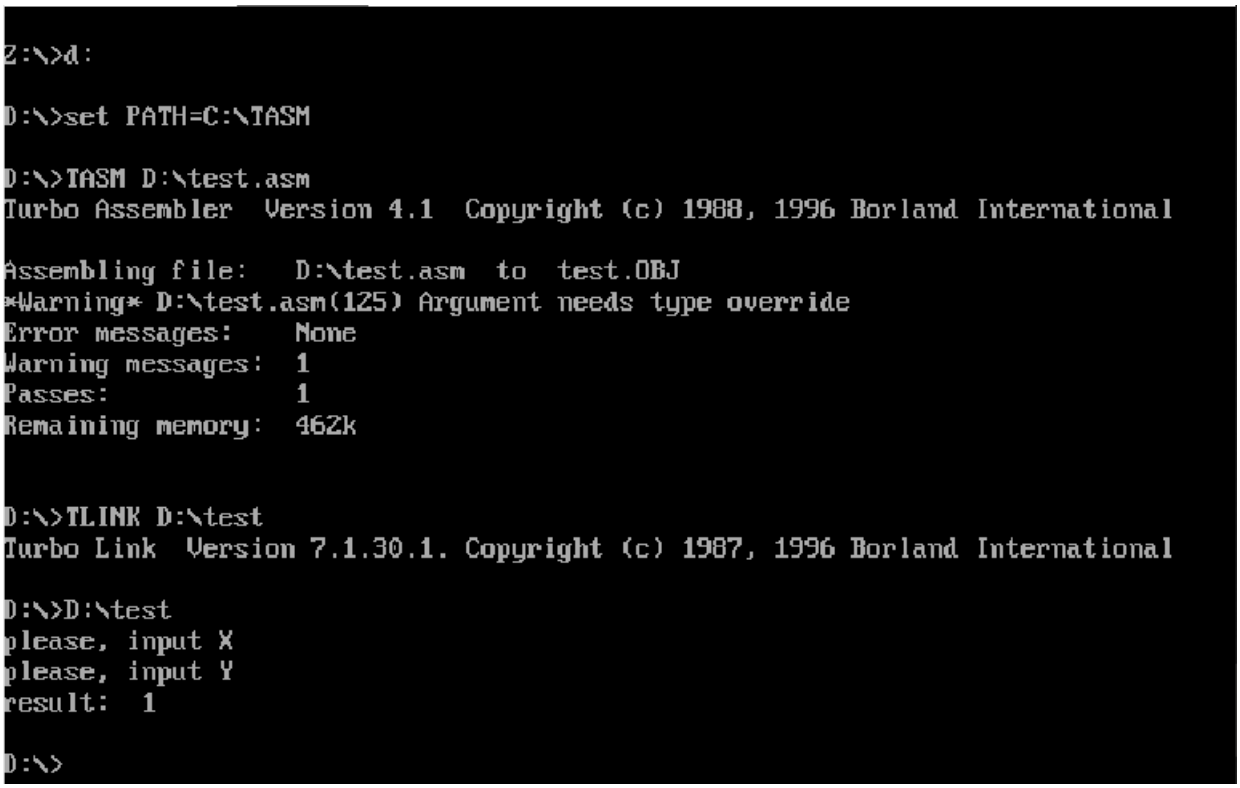
```
    POP bx
    RET
ENDP

PROC print
    PUSH dx
    MOV di,offset final_message
    add di,9
    PUSH cx
    PUSH bx
    MOV bx,10
    XOR CX,CX
print1:   XOR dx,dx
    DIV bx
    PUSH DX
    INC CX
    TEST AX,AX
    JNZ print1
print2:   POP AX
    ADD AL,'0'
    STOSb
    LOOP print2
    POP bx
    POP cx
    jc print3
    JMP print4
print3:
    MOV dx, offset result_out_of_bounds
    MOV ah,9
    INT 21h
    XOR dx, dx
    CALL exit
print4:
    MOV dx, offset final_message
    MOV ah,9
    INT 21h
    XOR dx, dx
    POP dx
RET
ENDP print


PROC print_rest
    MOV ax, dx
    MOV di,offset final_message_rest
    add di, 7
    PUSH cx
    PUSH dx
    PUSH bx
    MOV bx,10
    XOR CX,CX
```

```
rest1:   XOR dx,dx
    DIV bx
    PUSH DX
    INC CX
    TEST AX,AX
    JNZ rest1
rest2:   POP AX
    ADD AL,'0'
    STOSb
    LOOP rest2
    POP bx
    POP dx
    POP cx
    MOV dx, offset final_message_rest
    MOV ah,9
    INT 21h
    XOR dx, dx
    RET
ENDP print_rest

end Start
```

Скріншоти:







Висновок: У цій ЛР було розглянуто використання команд для умовних переходів, у реалізації практичної задачі, реалізації розргалуженного алгоритму.

Комп’ютерний практикум №4

Тема: Масиви

Завдання:

- Написати програму знаходження суми елементів одномірного масиву, елементи вводить користувач.
- Написати програму пошуку максимального (або мінімального) елемента одномірного
- масиву, елементи вводить користувач.
- Написати програму сортування одномірного масиву цілих чисел загального вигляду.
- Написати програму пошуку координат всіх входжень заданого елемента в двомірному масиві, елементи масиву та пошуковий вводить користувач.

Код застосунку:

```
IDEAL
MODEL SMALL
STACK 512

DATASEG
X DB "             ", 13, 10, '$'
matrix_main dw 0h, 0h ,0h ,0h
      dw 0h, 0h ,0h ,0h
      dw 0h, 0h ,0h ,0h
      dw 0h, 0h ,0h ,0h
uncorrect DB "uncorrect data!", 13, 10, '$'
start111 DB "input numbers:     ", 13, 10, '$'
buffe01 DB
"
        ", 13, 10, '$'
temp DB "///////", 13, 10, '$'
Task__4 DB "Task 4 - input number:     ", 13, 10, '$'
Task__3 DB "Task 3 - final Task__3 matrix:", 13, 10, '$'
Task__2_min DB "Task 2 - min number: ",'$'
Task__2_max DB "Task 2 - max number: ",'$'
Task__1 DB "Task 1 - sum: ", '$'
new_line DB "", 13, 10, '$'
separator DB "//*********||*********\\",13, 10,  '$'

Len dw 16



CODESEG
Start:
```

```
proc main
    mov ax, @data
    mov ds, ax
    mov es, ax

    call start1

    mov dx, offset separator
    call output_str
    call part1

    mov dx, offset separator
    call output_str
    call part4

    mov dx, offset separator
    call output_str
    call part3

    mov dx, offset separator
    call output_str
    call part2

exit:
    mov ah,04Ch
    mov al,0
    int 21h ; виклик функції DOS 4ch
endp main

proc start1
    mov dx, offset start111
    call output_str
    mov cx, 16
    xor si, si
    l1:
        call input
        mov dx, offset new_line
        call output_str
        mov [ds:10h+si], ax
        add si, 2
        loop l1
    call show_result
    ret
endp

proc part1
    mov cx, 16
    xor si, si
    xor ax,ax
    l3:
        add ax, [ds:10h+si]
```

```
        add si, 2
        loop l3
    mov dx, offset Task__1
    call output_str
    call output_register
    mov dx, offset new_line
    call output_str
    ret
endp
proc part2
    mov dx, offset Task__2_min
    call output_str
    mov ax, [ds:10h]
    call output_register
    mov dx, offset new_line
    call output_str

    mov dx, offset Task__2_max
    call output_str
    mov ax, [ds:2Eh]
    call output_register
    mov dx, offset new_line
    call output_str
    ret
endp

proc part3
    lea si, matrix_main
    mov cx, Len
    mov dx, offset Task__3
    call output_str
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    call sort
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    call show_result
    ret
endp

proc part4
    mov dx, offset Task__4
    call output_str
```

```
MOV dx, offset X
MOV al,7
PUSH cx
MOV cx,ax
MOV ah,0Ah              ;ввод строки в буфер
MOV [X],al
MOV [X+1],0
MOV dx,offset X
INT 21h
MOV al,[X+1]
add dx,2
MOV ah,ch
POP cx
CALL str_transformation

mov cx, 16
xor si, si
l5:
    mov bx,[ds:10h+si]
    push ax
    cmp ax,bx
    je point1
    jmp point2
point1:
    mov ax, si
    add ax, 2
    mov bl, 2
    div bl
    mov bl, 4
    div bl
    xor bx,bx
    mov bl, ah
    xor ah, ah
    add al, 1

    cmp bl, 0
    je point3
    jmp point5
point3:
    sub al,1
    mov bl, 4
    jmp point5

point5:
    mov dx, offset new_line
    call output_str
    call output_register
point5_5:
    xor ax,ax
    mov al, bl
```

```
        call output_register
        mov dx, offset new_line
        call output_str
    point2:
    pop ax
        add si, 2
        loop l5
        ret
endp part4

PROC show_result
    mov cx, 4
    xor si, si
    mov di, offset buffe01
    l6:
        mov ax,[ds:10h+si]
        call output_register_ax
        add si, 2
        add di, 4
        loop l6
    mov [ds:090h], 0A0Dh
    mov [ds:092h], 24h
    mov [ds:di-1h],' '

    mov dx, offset buffe01
    mov ah,09h
    int 21h

    mov cx, 4
    mov di, offset buffe01
    push si
    push cx
    push dx
    mov cx, 20h
    xor si, si
    loop1111:
        mov [ds:076h+si], ' '
        add si, 2
        loop loop1111
    pop dx
    pop cx
    pop si
    loop2222:
        mov ax,[ds:10h+si]
        call output_register_ax
        add si, 2
        add di, 4
        loop loop2222
    mov [ds:090h], 0A0Dh
    mov [ds:092h], 24h
    mov [ds:di-1h],' '
```

```
    mov dx, offset buffe01
    mov ah,09h
    int 21h
    mov cx, 4
    mov di, offset buffe01
    loop3333:
        mov ax,[ds:10h+si]
        call output_register_ax
        add si, 2
        add di, 4
        loop loop3333
    mov [ds:090h], 0A0Dh
    mov [ds:092h], 24h
    mov [ds:di-1h],' '
    mov dx, offset buffe01
    mov ah,09h
    int 21h
    mov cx, 4
    mov di, offset buffe01
    loop4444:
        mov ax,[ds:10h+si]
        call output_register_ax
        add si, 2
        add di, 4
        loop loop4444
    mov [ds:090h], 0A0Dh
    mov [ds:092h], 24h
    mov [ds:di-1h],' '
    mov dx, offset buffe01
    mov ah,09h
    int 21h
    mov cx, 4
    mov di, offset buffe01
    ret
ENDP show_result


PROC output_register
    mov [ES:0235h],' '
    mov [ES:0234h],' '
    mov [ES:0233h],' '
    mov [ES:0232h],' '
    mov [ES:0231h],' '
    mov di,0230h
    push cx
    push dx
    push bx
    mov bx,10
    XOR CX,CX
point_01:   XOR dx,dx
    DIV bx
```

```
    PUSH DX
    INC CX
    TEST AX,AX
    JNZ point_01
point_012:   POP AX
    ADD AL,'0'
    STOSb
    LOOP point_012
    pop bx
    POP dx
    POP cx
mov [ES:0235h],'$'
    mov dx, 230h
    mov ah,09h
    int 21h
ret
ENDP output_register


PROC str_transformation
    PUSH bx
    PUSH dx

    test al,al              ;Проверка длины строки
    jz point_str2
    MOV bx,dx
    MOV bl,[bx]
    CMP bl,'-'              ;проверка знака
    jne point_str1
    inc dx
    dec al
point_str1:
    CALL save_input
    jc point_str2
    CMP bl,'-'
    jne point_str3          ;Если первый символ не '-', то число положительное
    CMP ax,32734            ;проверка модуля числа
    ja point_str2
    JMP point_str4
point_str3:
    CMP ax,32767            ;проверка модуля числа
    ja point_str2
point_str4:
    clc
    JMP point_str5
point_str2:
    MOV dx,offset uncorrect ;; Закоментовані повідомлення у ході налаштування
    MOV ah,9
    INT 21h
    XOR dx, dx
    XOR ax,ax               ;AX = 0
```

```
    CALL exit
point_str5:
    POP dx                  ;Восстановление регистров
    POP bx
    RET
ENDP


PROC save_input
    PUSH cx                 ;Сохранение всех используемых регистров
    PUSH dx
    PUSH bx
    PUSH si
    PUSH di

    MOV si,dx
    MOV di,10
    MOV cl,al
    JCXZ end9           ;Если длина = 0, возвращаем ошибку
    XOR ax,ax
    XOR bx,bx

loop454:
    MOV bl,[si]
    INC si
    CMP bl,'0'
    JL end9
    CMP bl,'9'
    JG end9
    SUB bl,'0'              ;Преобразование символа-цифры в число
    MUL di
    JC end9
    ADD ax,bx
    JC end9             ;Если переполнение - ошибка
    LOOP loop454
    JMP exit21

end9:
    XOR ax,ax
    STC

exit21:
    POP di                  ;Восстановление регистров
    POP si
    POP bx
    POP dx
    POP cx
    ret
ENDP

proc sort
```

```
        mov bx, si
        mov dx, cx
        dec dx
        shl dx, 1
        dec cx
        mov si, 0
point_start:   mov di, dx
point121:   mov ax, [bx+di-2]
        cmp ax, [bx+di]
        jbe f_2
        xchg ax, [bx+di]
        xchg ax, [bx+di-2]
        xchg ax, [bx+di]
f_2:  sub di, 2
        cmp di, si
        ja point121
        add si, 2
        loop point_start
        ret
endp sort

PROC output_str
 push ax
 mov ah,9
 int 21h
 xor dx, dx
 pop ax
 ret
ENDP output_str

PROC input
    PUSH dx
    MOV al,7
    PUSH cx
    MOV cx,ax
    MOV ah,0Ah
    MOV [X],al
    MOV [X+1],0
    MOV dx,offset X
    INT 21h
    MOV al,[X+1]
    add dx,2
    MOV ah,ch
    POP cx
    CALL str_transformation
    POP dx
    RET
ENDP
PROC output_register_ax
    mov [ds:di+0Bh],20h
    mov [ds:di+0Ah],20h
```

```
    mov [ds:di+9h],20h
    mov [ds:di+8h],20h
    mov [ds:di+7h],20h
    mov [ds:di+6h],20h
    mov [ds:di+5h],20h
    mov [ds:di+4h],20h
    mov [ds:di+3h],20h
    mov [ds:di+2h],20h
    mov [ds:di+1h],20h
    push cx ;сохраняем регистры
    push dx
    push bx
    mov bx,10
    XOR CX,CX
circle1:   XOR dx,dx
    DIV bx      ;делим число на степени 10
    PUSH DX
    INC CX
    TEST AX,AX
    JNZ circle1
circle2:   POP AX
    ADD AL,'0'  ;преобразовываем число в ASCII символ
    STOSb
    LOOP circle2
    pop bx      ;восстанавливаем регистры
    POP dx
    POP cx
ret
ENDP output_register_ax
end Start
```

Висновок:

У цій ЛР було розглянуто використання масивів в Асемблері, для розв’язання задач, де потрібно зберігати багато однотипних даних.

Комп’ютерний практикум №5

Тема: Макрозасоби мови Асемблер

Завдання:

- Написати програму знаходження суми елементів одномірного масиву, елементи вводить користувач.
- Написати програму пошуку максимального (або мінімального) елемента одномірного
- масиву, елементи вводить користувач.
- Написати програму сортування одномірного масиву цілих чисел загального вигляду.
- Написати програму пошуку координат всіх входжень заданого елемента в двомірному масиві, елементи масиву та пошуковий вводить користувач.

Код застосунку:

У цій ЛР вийшло доволі багато коду(переробка 2-4 лаб на макроси), тож тут я залишу посилання на GitHub де знаходиться код 5-ї ЛР.

https://github.com/Bardin08/KPI-Third-Term/tree/master/SPS/CP5

Висновок:

У цій ЛР було розглянуто використання макросів для організаціх коду застосунку, а такод проведене рефакторинг коду попередніх робіт.