Комп’ютерний практикум №1
Завдання

Для наведеної програми створити файл типу **.asm**. Ця програма не має команд виведення даних на екран, тому правильність її виконання потрібно перевірити за допомогою відладчика **td.exe**.

Скомпілювати програму, включивши потрібні відлагоджувальні опції і опції створення файлу лістингу типу **.lst**.

Ознайомитися зі структурою файлу **.lst**. За вказівкою викладача, для певної команди асемблера розглянути структуру машинної команди і відобразити її в звіті.

Скомпонувати .obj-файл програми. Включити опції для налагодження та створення **.map-**файлу.

Занести в звіт адреси початку і кінця всіх сегментів з **.map**-файлу.
Завантажити в відладчик **td.exe** отриманий **.exe-**файл програми.
У вікні **CPU** в поле **DUMP** знайти початкова адреса сегменту даних і записати його в звіт. Знайти масиви **SOURCE** і **DEST**. Дані в масиві **SOURCE** відображаються в шістнадцятковій системі.
У покроковому режимі за допомогою клавіші ***F7*** виконати програму. Отримані результати в масиві **DEST** показати викладачеві.

```
; опис сегмента стека
STSEG  SEGMENT  PARA  STACK  “STACK”
DB 64 DUP ( “STACK” )
STSEG  ENDS
; опис сегмента даних
DSEG  SEGMENT  PARA  PUBLIC  “DATA”
SOURCE  DB 10, 20, 30, 40
DEST  DB 4 DUP ( “?” )
DSEG  ENDS
; опис сегмента коду
CSEG  SEGMENT PARA PUBLIC  “CODE”
; код основної функції
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
```

```
; обнулення масиву
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
```

```
; опис сегмента стека
STSEG  SEGMENT  PARA  STACK  “STACK”

DB 64 DUP ( “STACK” )

STSEG  ENDS
```

Слово, по якому виконується об’єднання

Потрібно для подальшого об’єднання сегментів, якщо є необхідність

Ім’я сегменту

Зарезервоване слово

Тип вирівнювання даних в сегменті кратне 16 (PARAGRAPH)

```
; опис сегмента даних
DSEG  SEGMENT  PARA  PUBLIC  “DATA”

SOURCE  DB 10, 20, 30, 40
DEST  DB 4 DUP ( “?” )

DSEG  ENDS


; опис сегмента коду
CSEG  SEGMENT PARA PUBLIC  “CODE”

; код основної функції
MAIN  PROC  FAR

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

Повідомляє транслятору про те, який сегмент до якого сегментного регістру прив'язаний

```
; обнулення масиву
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
```

ознака кінця процедури

ознака кінця сегменту коду

ознака кінця програми

Виконання програми на асемблері на ЕОМ складається з **4-х етапів**:

1. Створення текстового файлу типу **.asm** в будь-якому текстовому редакторі;

2. Компіляція створеного файлу, в результаті чого отримуємо об'єктний файл типу **.obj**.

Команда компілятора має структуру:

tasm [опції] source [, object] [, listing]

де source - ім'я **.asm** файлу-програми;
object - ім'я **.obj** файлу-трансляції;
listing - ім'я **.lst** файлу-лістингу;

елементи в дужках є необов'язковими.

Якщо відсутні **object** і **listing**, то імена відповідних файлів будуть такі ж, як і ім'я **.asm-**файлу.

Для створення **.lst**-файла до команди потрібно включити опцію **/ l**

Для створення *додаткової налагоджувальної інформації* в **.obj**-файл - опцію **/ zi**.

Компіляція **.asm**-файла здійснюється програмою **tasm.exe**

3. Компонування об'єктного файлу **(.obj**-файл)
Команда компоновщика має структуру:

tlink objfiles [, exefile] [, mapfile]

де **objfiles** - імена об'єктних файлів;
exefile - ім'я **.exe**-файлу;
mapfile - ім'я **.map**-файлу карти пам'яті.

При відсутності двох останніх компонентів їхні імена визначаються ім'ям об'єктного файлу.

Для створення карти пам'яті або **.map**-файлу в рядок потрібно включити опцію **/m**, для налагодження - опцію **/v**, для створення **.com**-файлу в команду включається опція **/t**.

В результаті чого отримуємо або багатосегментний **.exe**-файл, або односегментной **.com**-файл.

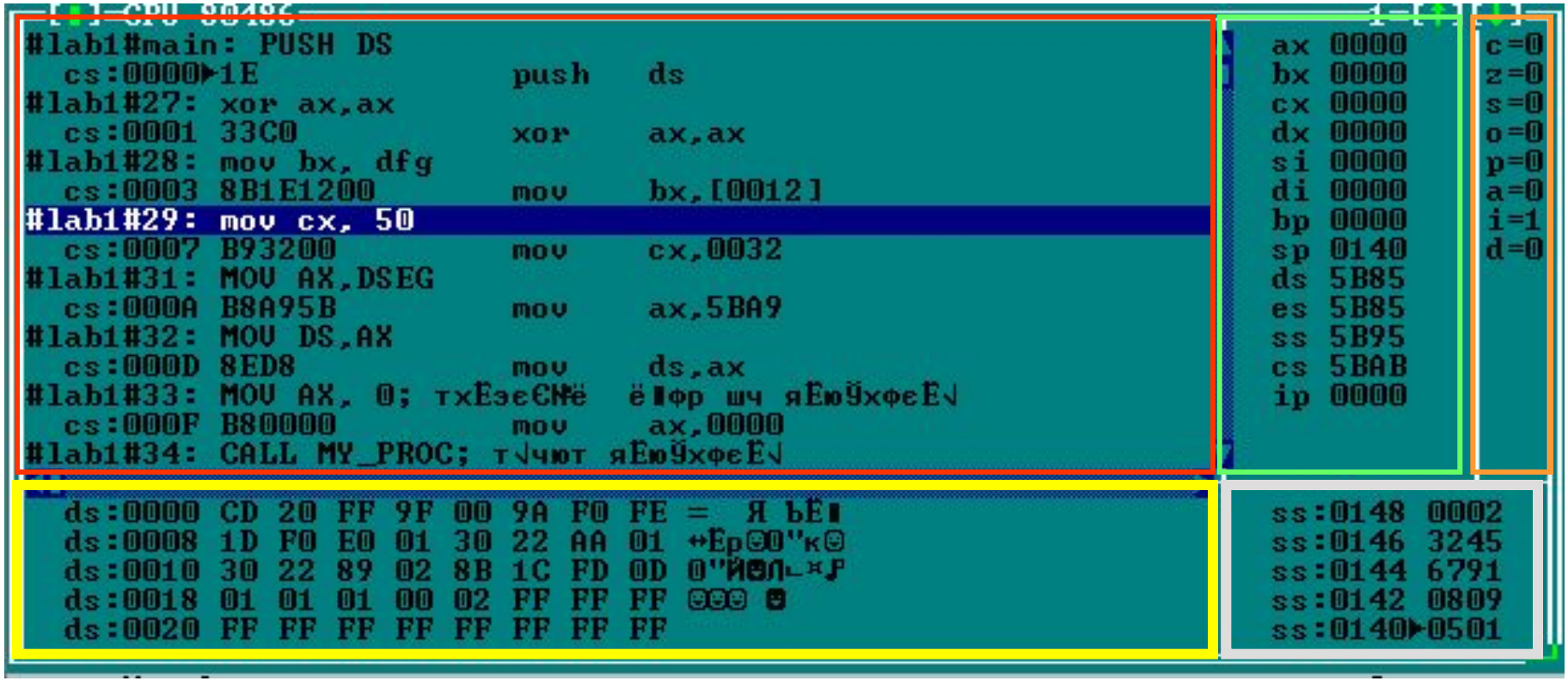
Компонування **.obj**-файла здійснюється програмою-компоновщиком **tlink.exe**.

4. Завантаження і виконання програми (**.exe**-файл) в відладчик **td.exe**.

Сегмент коду

Регістри

Прапорці



Сегмент даних

Сегмент стеку

Десяткове значення ***42936***

Перетворення десяткового формату в шістнадцятковий методом ділення числа на **16**

	Частка	Залишок	Шістнадцятковий	
42936/16	2683	8	8	Молодша цифра
2683/16	1676	11	B	
167/16	10	7	7	Старша цифра
10/16	0	10	A	

42936_{10} → $A7\ B8_{16}$

Перетворення шістнадцяткового числа в десятковий

$A7\ B8_{16}$ → 42936_{10}

Перша цифра: A (10) множимо на 16 →

```
  10
*  16
-----
 160
```

Додати наступну цифру: 7 →

```
+   7
-----
 167
```

Множимо на 16 →

```
*  16
-----
2672
```

Додати наступну цифру: B (11) →

```
+  11
-----
2683
```

Множимо на 16 →

```
*  16
-----
42928
```

Додати наступну цифру: 8 →

```
+   8
-----
42936
```

Перетворення **десяткового** формату в **двійковий** методом ділення числа на 2

44_{10}  101100_2

Для переведення необхідно ділити число із залишком на основу числення до тих пір, поки частка більше основи числення

44_{10} переведемо в двійкову систему

44 ділимо на 2 -- частка 22, остаток 0
22 ділимо на 2 -- частка 11, остаток 0
11 ділимо на 2 -- частка 5, остаток 1
5 ділимо на 2 -- частка 2, остаток 1
2 ділимо на 2 -- частка 1, остаток 0
1 ділимо на 2 -- частка 0, остаток 1

Частка дорівнює нулю, поділ закінчено.
Тепер записавши всі остатки справа наліво отримаємо число 101100_2

101100_2

Для вісімкової - розбиваємо на тетради, перетворимо по таблиці

101 100 → 5 4 $_8$

Для шістнадцятковій - розбиваємо на квартети, перетворимо по таблиці

0010 1100 → 2 C $_{16}$

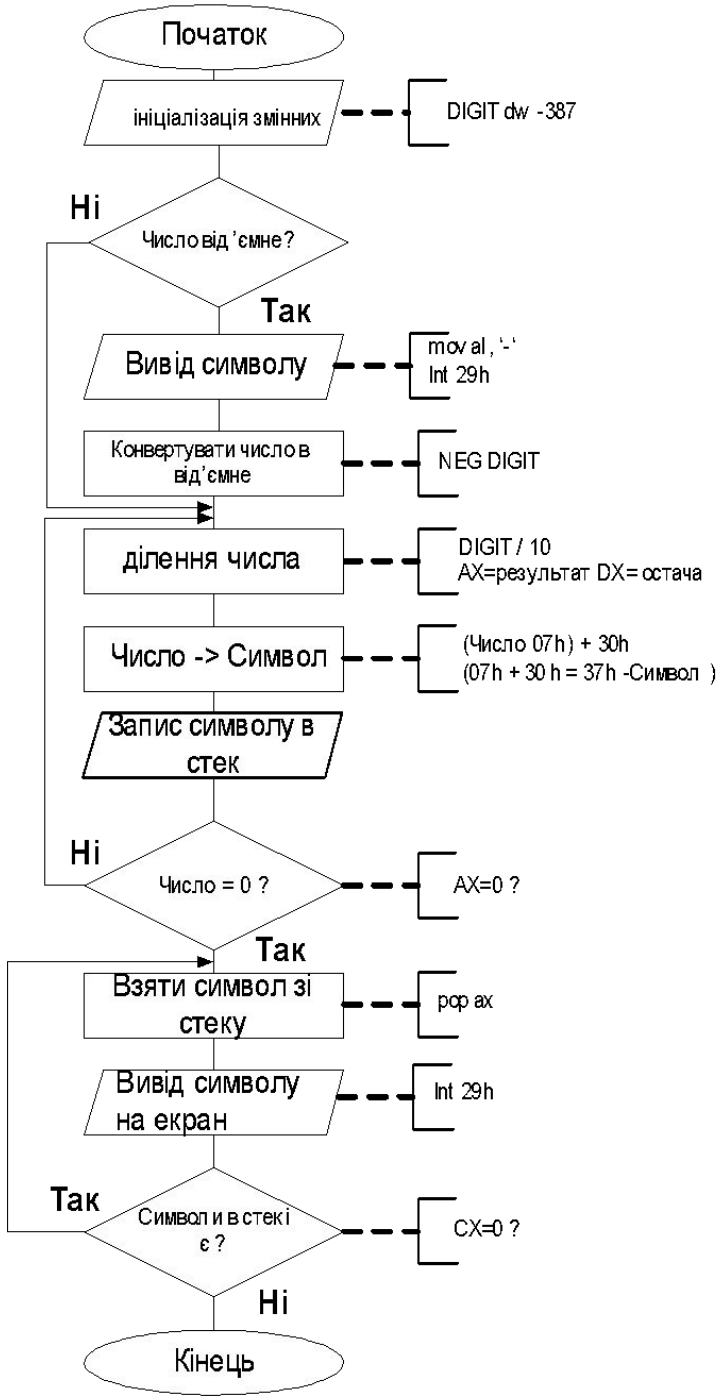
16 (hex)	10 (dec)	8 (oct)	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0
1	1	1	0	0	0	1
2	2	2	0	0	1	0
3	3	3	0	0	1	1
4	4	4	0	1	0	0
5	5	5	0	1	0	1
6	6	6	0	1	1	0
7	7	7	0	1	1	1
8	8	10	1	0	0	0
9	9	11	1	0	0	1
A	10	12	1	0	1	0
B	11	13	1	0	1	1
C	12	14	1	1	0	0
D	13	15	1	1	0	1
E	14	16	1	1	1	0
F	15	17	1	1	1	1

Комп’ютерний практикум №2

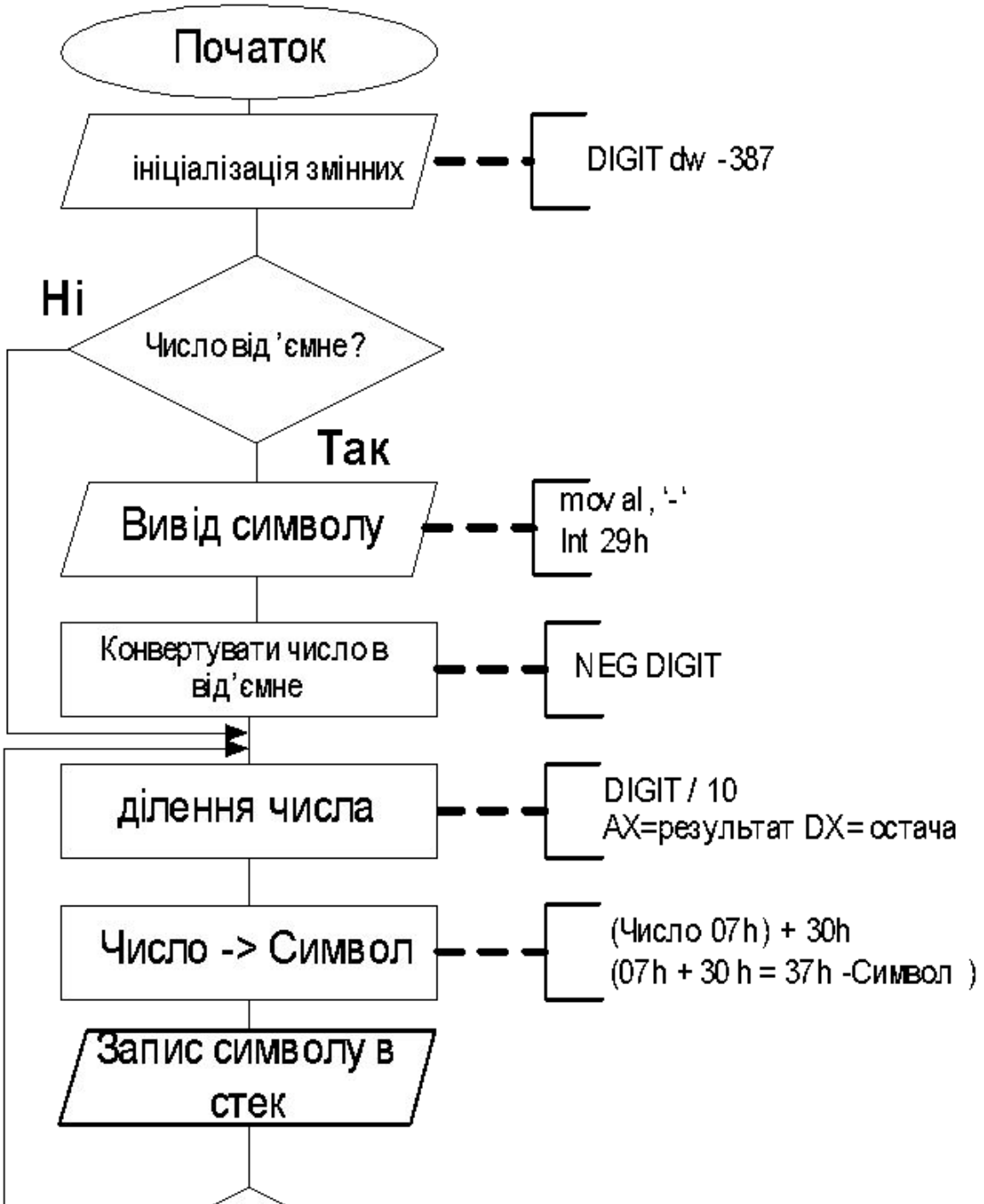
Завдання

1. Скласти процедуру введення і перетворення цілого числа.
2. Скласти і реалізувати програму введення і виведення цілого числа зі знаком і виведення рядка символів.

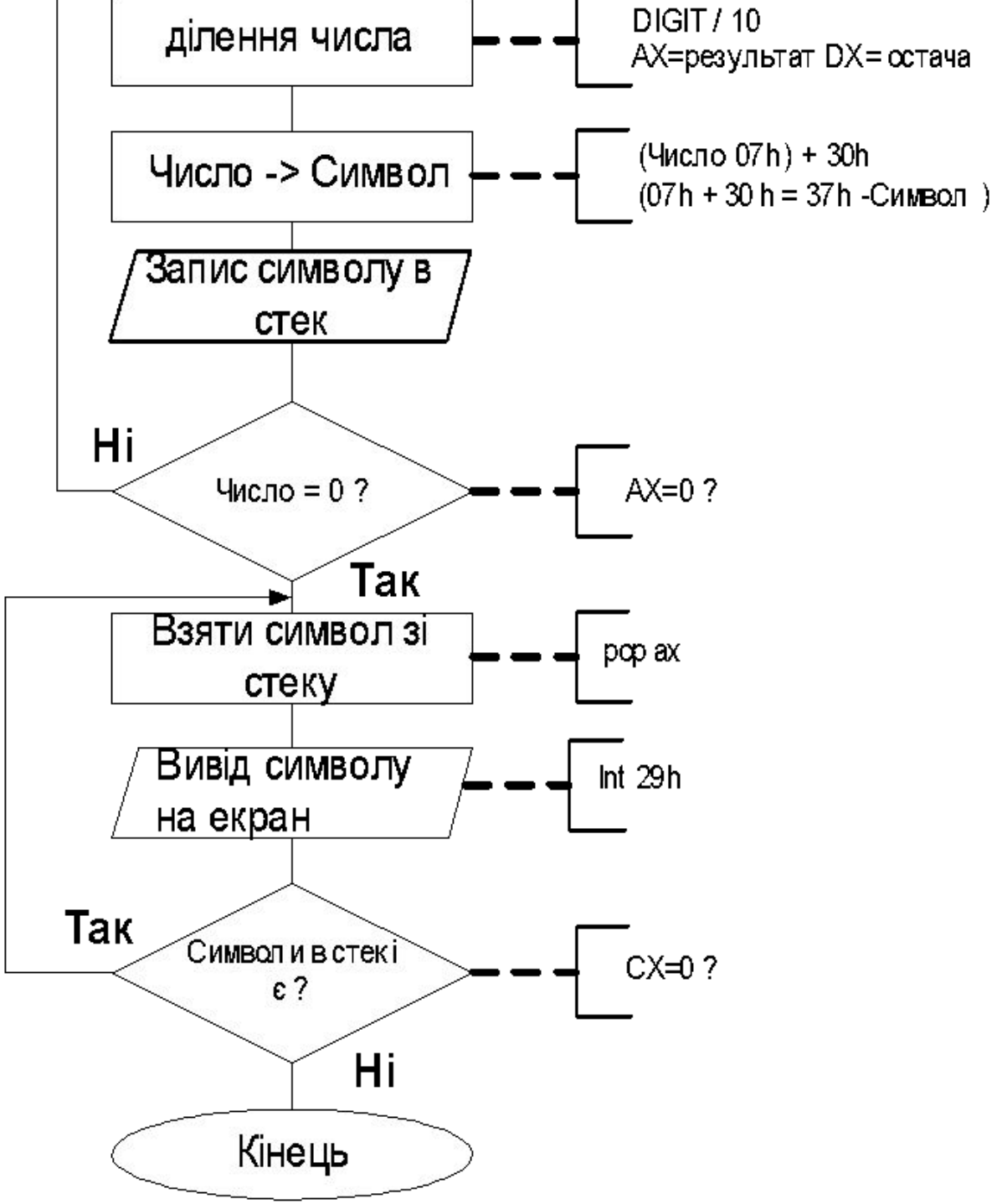
Процедура перетворення **числа** в **символ** **(є в методичці)**



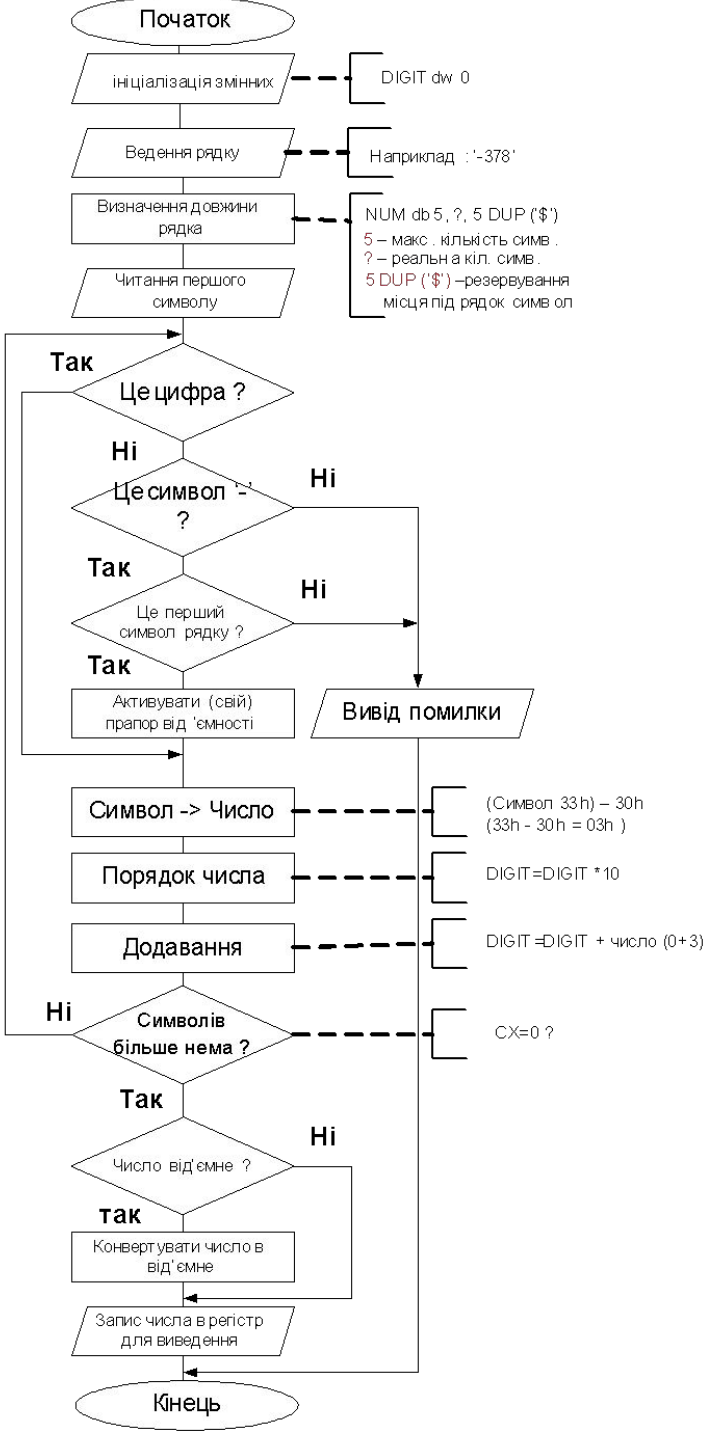
Процедура перетворення **числа в символ (є в методичці)**



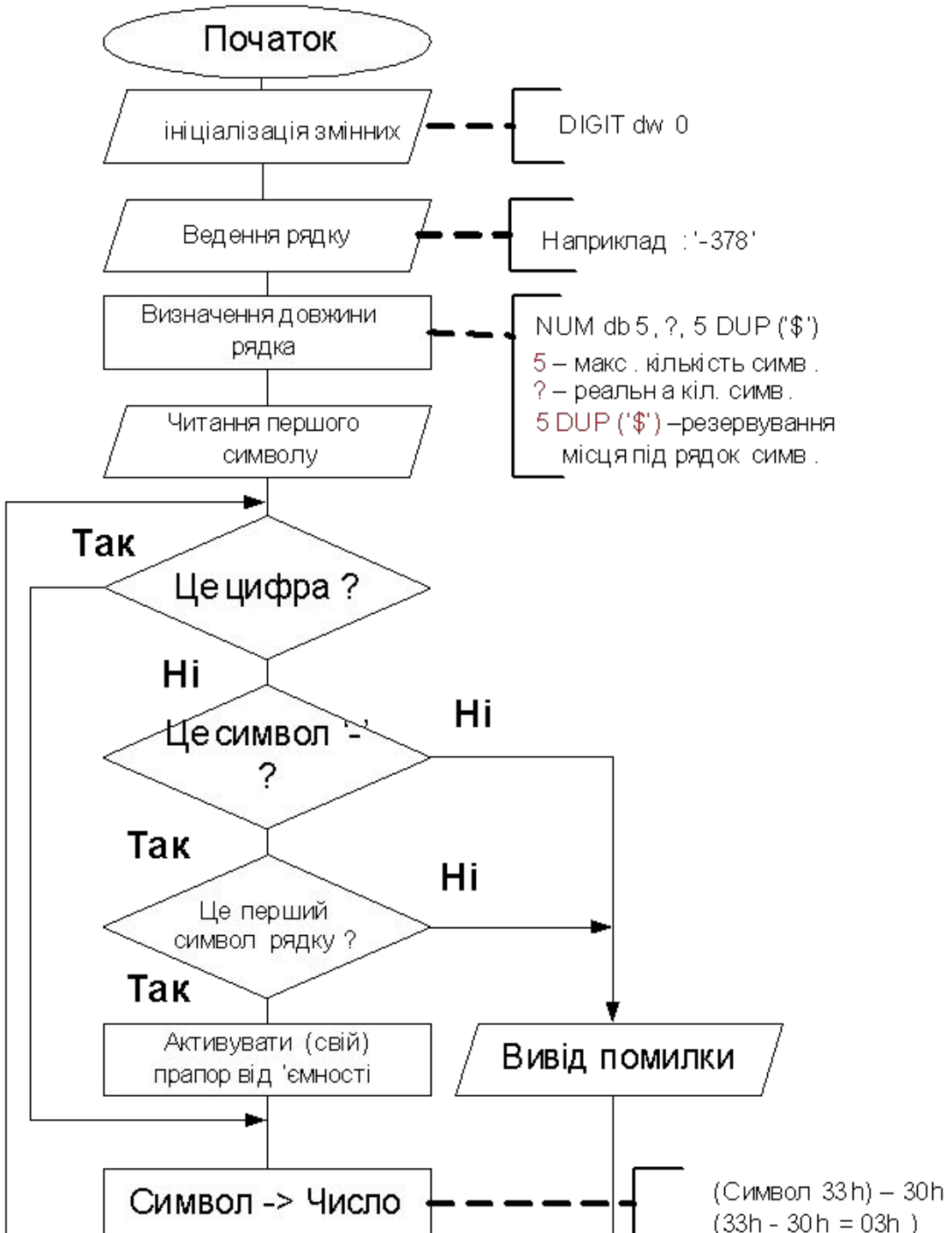
Процедура перетворення **числа** в **символ** **(є в методичці)**



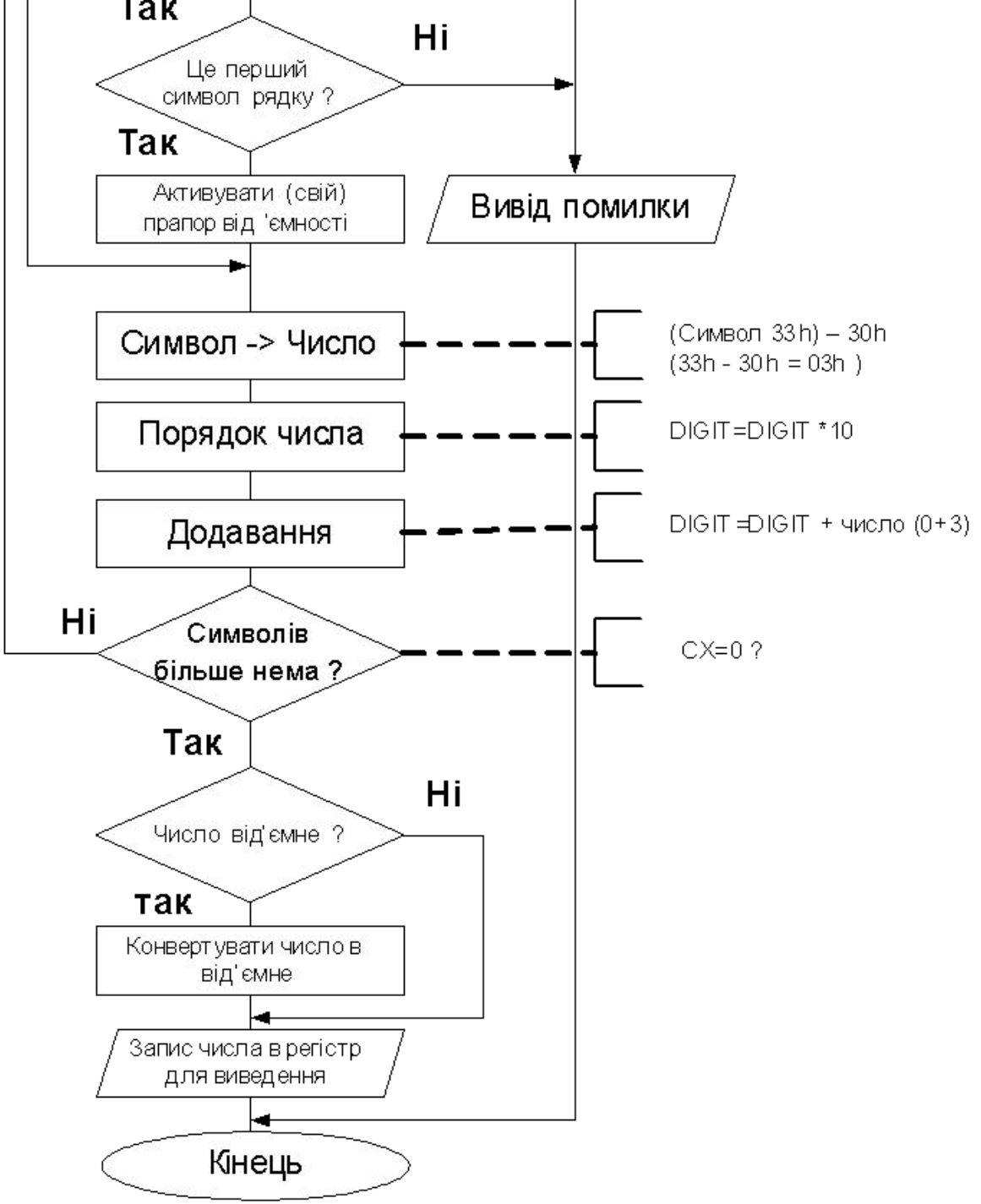
Процедура перетворення **символа** в **число**



Процедура перетворення **символа** **в** **число**



Процедура перетворення **символа** в **число**



Комп'ютерний практикум №3

Завдання

Написати програму, яка буде обчислювати значення функції.
Номер завдання за вказівкою викладача.

Наприклад:

$$Z = \begin{cases} (x+y)/xy & \text{якщо } y > 0;\quad x > 0 \\ 25x & \text{якщо } x = 0 \\ 6y & \text{якщо } y = 0 \\ 1 & \text{в інших випадках} \end{cases}$$

Таблиця
Значення абревіатур в назві команди **JCC**

Мнемонічне позначення	Англійська	Українська	Тип операндів
E чи e	equal	Рівні	Будь-які
N чи n	not	Не	Будь-які
G чи g	greater	Більше	Числа із знаком
L чи l	less	Менше	Числа із знаком
A чи a	above	Вищий в значенні «більший»	Числа без знаку
B чи b	below	Нижчий в значенні «менший»	Числа без знаку

Команди реакції на арифметичні порівняння із знаком

Для таких порівнянь використовуються слова «менше» (Less) і “більше” (Greater):

Типи операндів	Мнемокод команди	Критерій умовного переходу	Значення прапорців для здійснення переходу
Будь-які	**JE**	операнд_1 = операнд_2	zf = 1
Будь-які	**JNE**	операнд_1<>операнд_2	zf = 0
Із знаком	**JL / JNGE**	операнд_1 < операнд_2	sf <> of
Із знаком	**JLE / JNG**	операнд_1 <= операнд_2	sf <> of ЧИ zf = 1
Із знаком	**JG / JNLE**	операнд_1 > операнд_2	sf = of І zf = 0
Із знаком	**JGE / JNL**	операнд_1 => операнд_2	sf = of

Наприклад:

CMP AX, BX
JL LABEL2; перехід, якщо AX<BX

Команди реакції на арифметичні прівняння без знаку

Для таких порівнянь використовуються слова «вище» (Above) і «нижче» (Below), після порівняння (**CMP**) адрес:

Типи операндів	Мнемокод команди	Критерій умовного переходу	Значення прапорців для здійснення переходу
Будь-які	**JE**	операнд_1 = операнд_2	zf = 1
Будь-які	**JNE**	операнд_1<>операнд_2	zf = 0
Без знаку	**JB / JNAE**	операнд_1 < операнд_2	cf = 1
Без знаку	**JBE / JNA**	операнд_1 <= операнд_2	cf = 1 ЧИ zf=1
Без знаку	**JA / JNBE**	операнд_1 > операнд_2	cf = 0 I zf = 0
Без знаку	**JAE / JNB**	операнд_1 => операнд_2	cf = 0

Команди перевірки окремих прапорців і регістрів

Структура: першим іде символ: “**J**” (jump, перехід), другий – або позначення прапорця, або символ заперечення “**N**”, після якого стоїть назва прапорця.

Назва прапорця	№ біта в eflags/flag	Команда	Значення прапорця
Прапорець переносу cf	1	JC	CF = 1
Прапорець парності pf	2	JP	PF = 1
Прапореь нуля zf	6	JZ	ZF = 1
Прапорець знаку sf	7	JS	SF = 1
Прапорець переповнення of	11	JO	OF = 1
Прапорець переносу cf	1	JNC	CF = 0
Прапорець парності pf	2	JNP	PF = 0
Прапорець нуля zf	6	JNZ	ZF = 0
Прапорець знаку sf	7	JNS	SF = 0
Прапорець переповнення of	11	JNO	OF = 0

Типи операндів	Команда умовного переходу	Критерій умовного переходу	Значен. регістрів для переходу
Будь-які	JCXZ	Jump if **CX** is Zero	CX = 0
Будь-які	JECXZ	Jump Equal **ECX** Zero	ECX = 0

Наприклад:

```
CMP AX, BX
JE cycl
JCXZ    m1  ; обійти цикл, якщо CX=0
cycl:;деякий цикл
LOOP  cycl
m1:     ...
```

2.2 Арифметичні команди

Цілочисельний обчислювальний пристрій підтримує трохи більше десятка арифметичних команд.

Комп’ютерний практикум №4

Завдання

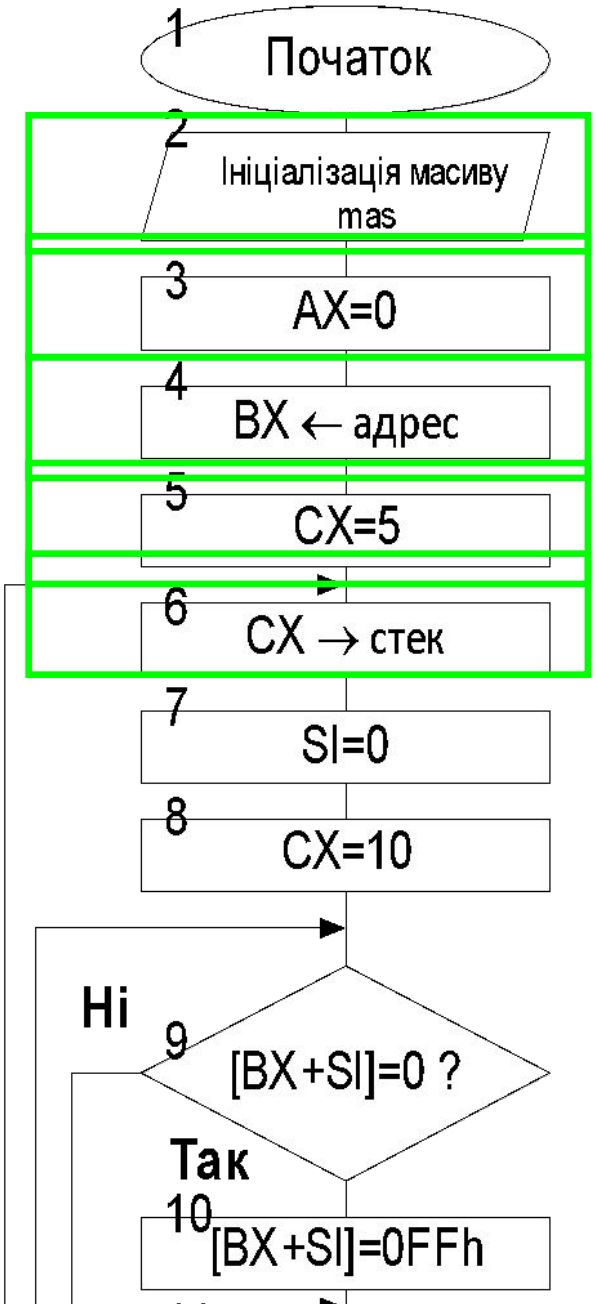
1. Написати програму додавання елементів масиву.

2. Написати програму пошуку максимального
(або мінімального) елемента масиву.

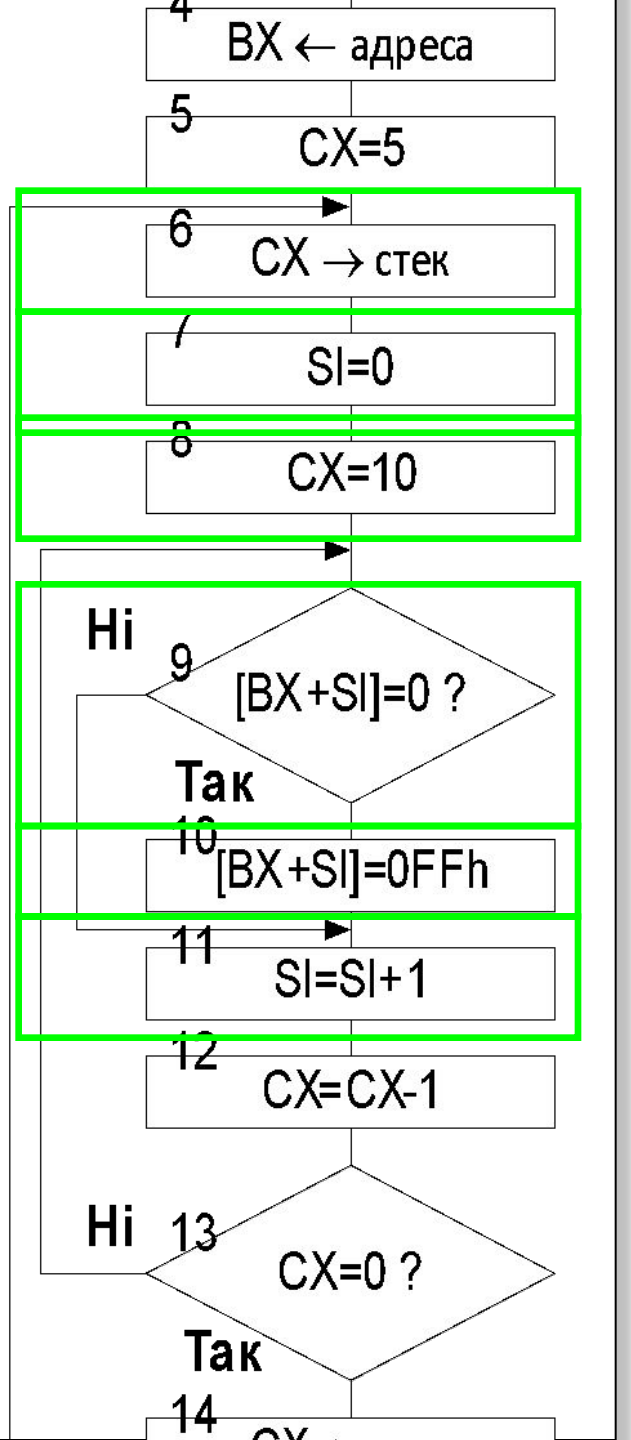
3. Написати програму пошуку всіх вкладень
заданого елемента в двомірному масиві.

4. Написати програму сортування масиву цілих чисел загального вигляду.

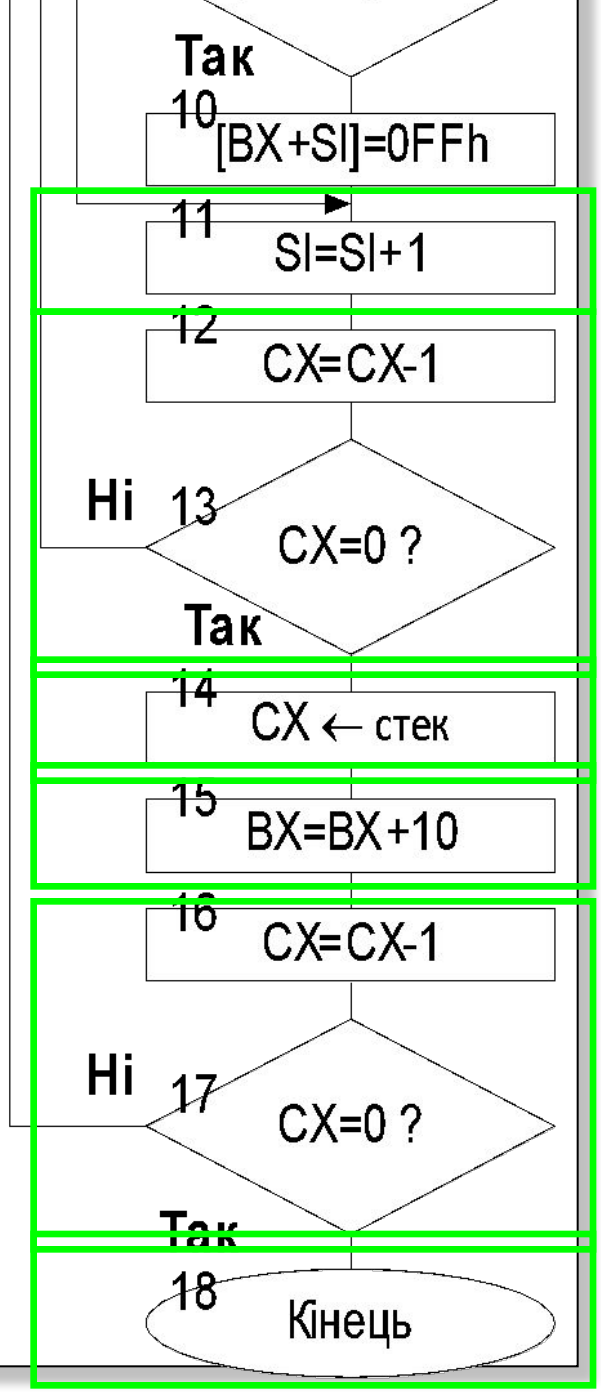
```
model small
.stack 100h
.data
mas    db 1,0,9,8,0,7,8,0,2,0
       db 1,0,9,8,0,7,8,0,2,0
       db 1,0,9,8,0,7,8,?,2,0
       db 1,0,9,8,0,7,6,?,3,0
       db 1,0,9,8,0,7,8,0,2,0
.code
start:
 mov ax,@data
 mov ds,ax
  xor ax, ax
  lea bx, mas
  mov cx, 5
  cycl_I:
   push cx
   xor si, si
```



```
 mov cx, 5
 cycl_l:
  push cx
  xor si, si
  mov cx, 10
  cycl_2:
    cmp byte ptr [bx+si], 0
    jne no_zero
    mov byte ptr [bx+si], 0ffh
   no_zero:
    inc si
   loop cycl_2
  pop cx
  add bx, 10
 loop cycl_1
exit:
mov ax,4c00h
int 21h
end start
```



```
cycl_1:
 push cx
 xor si, si
 mov cx, 10
 cycl_2:
   cmp byte ptr [bx+si], 0
   jne no_zero
   mov byte ptr [bx+si], 0ffh
  no_zero:
   inc si
  loop cycl_2
 pop cx
 add bx, 10
 loop cycl_1
exit:
mov ax,4c00h
int 21h
end start
```



Комп’ютерний практикум №5

Завдання

Скласти програму на нижче наведені завдання:

1. Переписати програму 2.1 з використанням макросів;

2. Переписати програму 3.1 - написати, використовуючи макроси, програму, знаходження значення заданої функції (умови наведені в таблиці 3.3);

3. Переписати програму 4.1 з використанням макросів.

Макроозначення розміщуються в будь-якому місці програми, але ДО виклику макрокоманди.

```
ADD_WORDS MACRO ARG1, ARG2, SUM
 MOV AX, ARG1
 ADD AX, ARG2
 MOV SUM, AX
ENDM
```

Звернення до макрокоманді має вигляд:

```
ADD_WORDS TERM1, TERM2, COST
```

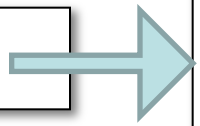
Зрозуміло, що таких звернень в програмі може бути декілька.

Як виконується макрокоманда?

На місце виклику макрокоманди вставляється тіло макроозначення із заміною формалізованих параметрів фактичними.

Тобто, замість одного рядка звернення буде розміщено 3 рядки:

```
ADD_WORDS TERM1, TERM2, COST
```



```
MOV AX, TERM1
ADD AX, TERM2
MOV COST, AX
```

Макрокоманди виконуються швидше процедур, немає потреби в переходах і повернення.
Але використання макрокоманд збільшує об'єм пам'яті: в тілі програми макровизначення дублюються стільки разів, скільки були викликані.

Процедура ж в пам'яті записується один раз.

Макровизначення можна записати в бібліотеку і використовувати при розробці нових програм.