

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

**"ЗАТВЕРДЖУЮ"**

Декан ФІОТ О.А.Павлов

«\_\_\_\_\_»

---

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ СТУДЕНТАМ ЩОДО ЗАСВОЄННЯ  
КРЕДИТНОГО МОДУЛЯ

«ПРОГРАМУВАННЯ, ч.ІІІ»

«СИСТЕМНЕ ПРОГРАМУВАННЯ»

Рекомендовано кафедрою  
технічної кібернетики  
протокол № 1  
від "29" серпня 2017 р.

Завідувач кафедри

Пархомей І.Р.

Київ 2017

Програмування, ч.3. Системне програмування: Метод. вказівки до комп'ютерного практикуму для студентів 2-го курсу напряму “Програмна інженерія” Уклад. О.І. Лісовиченко. – К.: КПІ ім. Ігоря Сікорського, 2017 – 37с.

*Рекомендовано кафедрою  
технічної кібернетики ФІОТ  
КПІ ім. Ігоря Сікорського  
(протокол №1 від 29.08.2017 р.)*

Навчальне видання

**Програмування, ч.3  
Системне програмування**

МЕТОДИЧНІ ВКАЗІВКИ  
ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ  
для студентів 2-го курсу напряму “Програмна інженерія”

Укладачі *Лісовиченко Олег Іванович*, к.т. н., доцент

Відповідальний редактор *Ткач Михайло Мартинович*, к.т.н., доцент

Рецензенти *Остапченко Костянтин Борисович*, к.т.н., доцент

## Зміст

<i>Вступ</i> .....	4
<i>Комп'ютерний практикум № 1</i> .....	5
1.1 Загальні положення.....	5
1.2 Завдання .....	8
1.3 Контрольні запитання .....	8
<i>Комп'ютерний практикум № 2</i> .....	9
2.1 Загальні положення.....	9
2.2 Завдання .....	11
2.3 Контрольні питання.....	12
<i>Комп'ютерний практикум № 3</i> .....	13
3.1 Загальні положення.....	13
3.2 Завдання .....	14
3.3 Контрольні питання.....	16
<i>Комп'ютерний практикум № 4</i> .....	17
4.1 Загальні положення.....	17
4.2 Завдання .....	20
4.3 Контрольні питання.....	20
<i>Комп'ютерний практикум № 5</i> .....	21
5.1 Загальні положення.....	21
5.2 Завдання .....	23
5.3 Контрольні питання.....	23
<i>Література</i> .....	24
<i>Додаток 1</i> .....	26
<i>Додаток 2</i> .....	32

## ***Вступ***

Ці методичні вказівки призначені студентам напряму «Програмна інженерія» для виконання комп'ютерних практикумів з курсу “Програмування”, частини “Системне програмування”.

В основу дисципліни покладено вивчення мови асемблер персональних комп'ютерів (ПК) на базі мікропроцесора i8086/88. Наступні покоління процесорів фірми Intel відзначаються спадковістю на рівні машинних команд: програми, написані для попередніх моделей можуть виконуватися і на наступних. Тому знання та володіння навичками програмування мовою асемблера для зазначеного типу процесора є передумовою для опанування додатковими можливостями сучасних моделей.

Мова асемблер є машинно-орієнтованою, передбачає знання архітектури конкретного ПК, тому програмування більш трудомістке і вимагає ґрунтовнішої початкової підготовки, порівнюючи з мовами високого рівня. Але вивчення цієї мови дозволяє зрозуміти принципи функціонування ПК, операційних систем і трансляторів з мов високого рівня, повністю використати можливості мікропроцесора, створити високоефективні програми.

Для виконання лабораторних робіт необхідно засвоїти теоретичний матеріал зазначеної теми, дати відповіді на контрольні питання. Після цього треба написати програму та занести її до протоколу. В лабораторії студент подає протокол викладачеві, відповідає на поставлені ним питання. Після виконання програми і одержання правильних результатів студент записує їх до протоколу, робить висновки і захищає звіт.

## *Комп'ютерний практикум № 1*

**Тема:** створення програм на асемблері.

### 1.1 Загальні положення

Будь-який сучасний мікропроцесор, за допомогою шести сегментних реєстрів, може одночасно працювати:

- ✓ з одним сегментом коду;
- ✓ одним сегментом стеку;
- ✓ одним сегментом даних;
- ✓ трьома додатковими сегментами даних.

Фізично кожен сегмент являє собою сектор пам'яті, що зайнята командами або даними, адреси яких вираховуються відносно значення у відповідному сегментному реєстрі.

У найпростішому випадку програма написана на асемблері складається з опису трьох сегментів: стеку, даних та коду. Такий розподіл пам'яті дуже важливий, оскільки інакше компілятор, що не може відрізнити данні від команд, буде все сприймати як команди.

Синтаксично сегмент описується наступним чином:

*Ім'я\_сегменту*        **SEGMENT**    *тип\_вирівнювання*    *тип\_комбінування*  
                                 *клас\_сегмента*        *тип\_розміру\_сегмента*

- *директиви асемблера*
- *команди асемблера*
- *макрокоманди асемблера*
- *коментарі*

*Ім'я сегменту*        **ENDS**

Для процесорів i80386 та вище сегменти можуть бути 16- або 32-разрядними. Це впливає перш за все на розмір сегмента і на формування фізичної адреси у ньому. Атрибут розміру сегмента може приймати значення:

- ✓ USE16 – ознака того, що сегмент дозволяє 16-разрядну адресацію. При формуванні фізичної адреси може бути використано лише 16-разрядне зміщення. Такий сегмент може містити до 64 Кбайт коду або даних.
- ✓ USE32 – сегмент буде 32-разрядним. При формуванні фізичної адреси може бути використано 32-разрядне зміщення. Отже, такий сегмент може містити до 4 Гбайт коду або даних.

Всі сегменти рівноправні. Тому, щоб використовувати їх як сегменти коду, стеку і даних, необхідно попередньо повідомити про це транслятору, для чого використовують спеціальну директиву ASSUME. Ця директива повідомляє транслятору про те, який сегмент до якого сегментного реєстру прив'язаний. Це дозволить транслятору коректно зв'язати символічні імена, що визначені у сегментах.

Сегмент коду складається з процедур. Однієї головної, з якої починається виконання програми, та допоміжних. Описується процедура наступним чином:

*Ім'я\_процедури*    **PROC**    *[відстань]*

**[ARG**        *список аргументів]*

**[RETURNS**    *список аргументів]*

**[LOCAL**        *список аргументів]*

**[USES**        *список реєстрів]*

- *група команд;*
- *директиви асемблера;*
- *коментарі тощо;*

**[ret]** *[число]*

### *Ім'я\_процедури*    **ENDP**

Отже обов'язковим є задання імені процедури. Атрибут *відстань* може приймати значення **near** або **far** і характеризує можливість звернення до процедури з іншого сегмента коду. Головна процедура завжди має бути **far**. Виклик процедури здійснюється командою **call ім'я\_процедури**. Команда **ret [число]** повертає управління програмі, що здійснює виклик. **[число]** – необов'язковий параметр, що зазначає кількість елементів, які видаляються зі стеку при поверненні з процедури.

Наведемо приклад програми, яка буде переписувати масив з чотирьох елементів у зворотному порядку.

#### *Програма 1.1*

```
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
```

Виконання програми мовою асемблер на комп'ютері складається з таких етапів:

- 1) створення текстового файлу типу .asm у будь-якому текстовому редакторі;
- 2) компіляція створеного файлу, в результаті чого одержуємо об'єктний файл типу .obj. За бажанням користувача, можна створити допоміжний файл лістингу типу .lst, де наводяться зміщення кожної команди або даних, машинні коди команд та діагностуються помилки трансляції;
- 3) компоновання об'єктного файлу, в результаті чого отримуємо або багатосегментний .exe-файл, або односегментний .com-файл. Додатково можна одержати карту пам'яті або .map-файл, де зазначено, як скомпоновано програму – початкові та кінцеві адреси кожного сегменту;
- 4) завантаження та виконання програми.

Компіляція .asm-файла здійснюється програмою `tasm.exe`, а компоновання .obj-файлу – компоновником `tlink.exe`.

Для задоволення різних потреб користувача компіляція та компоновання повинні здійснюватись з відповідними особливостями, що задаються опціями, які позначаються певними символами.

Налагодження програми можна здійснювати за допомогою універсального налагоджувача `td.exe`, який обробляє її .exe-файл. Щоб встановити відповідність між командами .exe-файлу та початковим .asm, потрібно створити спеціальні таблиці. Тому до .obj- та .exe-файлів програми треба включати додаткову інформацію для налагоджувача.

Команда компілятора має таку структуру

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

де **source** – ім'я .asm-файлу;

**object** – ім'я .obj-файлу;

**listing** - ім'я .lst-файлу;

елементи в дужках є необов'язковими.

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

Для створення .lst-файлу до команди треба включити опцію /l, а для налагоджувача – опцію /zi.

Команда компоновника має таку структуру

**tlink objfiles [,exefile][,mapfile],**

де **objfiles** – імена об'єктних файлів;

**exefile** – ім'я .exe-файлу;

**mapfile** - ім'я файлу карти пам'яті.

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

Для створення .map-файлу до команди треба включити опцію /m, для налагоджувача – опцію /v, для створення не .exe-файлу, а .com-файлу до команди включається опція /t.

Після виправлення всіх помилок компіляції та компоновання одержану .exe програму можна налагоджувати за допомогою `td.exe`.

Ця програма використовує графічний інтерфейс, який дуже схожий на середовище Turbo Pascal або Borland C. Головне меню складається з кількох розділів. Завантаження програми до налагоджувача можна здійснити через розділ File.

З розділу перегляду View доцільно використовувати режим DUMP (розміщення програми в пам'яті), розділ CPU, в якому фактично одночасно є п'ять вікон. Переміщуватися по вікнах у напрямі годинникової стрілки слід за допомогою клавіші Tab, а в протилежному напрямку – Shift-Tab.

Кожне вікно має локальне меню, яке можна зробити доступним клавішами Alt+F10 або Ctrl+F10.

Покрокове виконання програми можна здійснити клавішами F7 або F8, при цьому текстом програми пересувається стрілочка, а у інших вікнах відображаються зміни у регістрах, стеку, пам'яті, стані регістру прапорців.

У режимі Module відображається лише послідовність виконання команд .asm-файлу. Детальніше середовище td.exe описано в [2] с.66 –82 та [3].

### 1.2 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 .map-файлу.
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.

### 1.3 Контрольні запитання

1. Структура програми мовою асемблер.
2. Початкове завантаження сегментних регістрів.
3. Забезпечення передачі управління операційній системі після завершення програми.
4. Методи адресації [2] с.123 – 128.
5. Структура двоадресної машинної команди [1] с.17 – 20.
6. Чому значення початкової адреси сегменту даних у .map-файлі не збігається з її значенням після завантаження на виконання?



## Комп'ютерний практикум № 2

**Тема:** засоби обміну даними.

### 2.1 Загальні положення

Обмін даними здійснюється зовнішніми пристроями ПК. Тому програми обміну є процедурами управління цими пристроями. Для їх реалізації треба знати особливості роботи певного пристрою, структуру портів управління тощо. Оскільки обмін даними використовується у багатьох програмах, то він реалізований у вигляді стандартних процедур, які знаходяться у пам'яті ПК. Для їх виклику можна було б використовувати команду **call** із зазначенням адреси початку певної процедури обміну. Але у різних версіях операційних систем (ОС) ці початкові адреси різні, тому програми обміну реалізовані як процедури обробки переривань.

Ці процедури викликаються за допомогою команди **int** (interrupt) із зазначенням вектора відповідного переривання, наприклад:

```
int 25h
```

Оскільки процедур обміну багато, то відводити кожній свій вектор переривання недоцільно, а можна пов'язати їх одним вектором 21h, кожен процедуру визначити як функцію з відповідним номером. Цей номер треба попередньо записати до регістру АН. В залежності від особливостей функція може мати певні параметри, які треба занести до певного регістру. Отже виклик функції DOS матиме вигляд:

```
mov ah, number ;номер функції  
<пересилання параметрів>  
int 21h
```

Функції BIOS пов'язані з вектором переривання 10h.

#### 2.1.1 Виведення символу

Виведення будь-яких даних базується на процедурі виведення одного символу. Для цього призначена функція 2 вектора переривання DOS 21h, сам символ повинен бути записаним до регістру DL.

Отже, виклик зазначеної функції матиме вигляд:

```
mov ah, 2  
mov dl, '*'  
int 21h
```

Швидше працює виведення символу за допомогою переривання 29h. Символ, який потрібно ввести, повинен знаходитись у регістрі AL:

```
mov al, '*'  
int 29h
```

#### 2.1.2 Виведення рядка символів

Виведення рядка символів здійснюється за допомогою функції 9. Попередньо до регістру DS треба занести номер того сегменту пам'яті, де знаходиться рядок, а до регістру DX – початкову адресу рядка (зміщення). Рядок має закінчуватись символом '\$'. Наприклад:

```
mas db 13, 10, 'string $'  
...  
lea dx, mas  
mov ah, 9  
int 21h
```

### 2.1.3 Введення рядка символів

Введення рядка символів здійснюється за допомогою функції 10 (0A h) переривання 21h DOS. Символи вводяться до буферу, який треба передбачити у програмі. Особливості такого буферу:

- ✓ до першого байту треба записати максимальну кількість символів  $max$ , яку можна ввести;
- ✓ якщо введено  $max-1$  символів, то при спробі ввести ще якісь символи лунає звуковий сигнал і ці символи до буферу не записуються;
- ✓ після завершення вводу сама функція запише до другого байта кількість фактично введених символів  $n$ , послідовно коди всіх символів; за останнім символом буде записано символ “кінця рядка” (CR = 13), який до числа  $n$  не зараховується.

Перед викликанням функції до регістру DS має бути записаний номер сегменту, де знаходиться буфер, а до регістру DX треба записати початкову адресу буфера.

Приклад введення 20 символів:

```
...  
dump db 6, '?', 6 dup('?') ; в сегменті даних  
...  
lea dx, dump  
mov ah, 10  
int 21h
```

Введені символи висвічуються на екрані і їх можна редагувати ( клавішою Backspace видаляє останній, а ESC – рядок ), після чого належить натиснути клавішу Enter.

### 2.1.4 Виведення цілого числа

Ціле число зі знаком записується в пам'яті ПК у додатковому двійковому коді. Тому для виведення його необхідно перетворити на послідовність десяткових цифр-символів. Отже, спочатку треба одержати значення десяткових цифр, починаючи з нульового розряду, а потім кожну цифру перетворити на символ. Перше можна здійснити послідовним діленням числа на 10, остача від ділення визначатиме відповідну десяткову цифру.

Як відомо, код нуля –  $48_{10}$ , а коди всіх наступних символів цифр збільшуються на одиницю, так що символ “9” має код  $57_{10}$ . Отже, код символу-цифри можна одержати додаванням до неї 48, або символу “0”.

Для від'ємного числа попередньо достатньо вивести символ мінус і перетворити його на додатне.

Виведення цілого числа реалізує процедура `digit`, яка подається нижче. Число `num` записується до регістру BX. Спочатку перевіряється знак числа, і для від'ємного здійснюються відповідні перетворення.

Далі число записується до акумулятора для послідовного ділення на 10. Регістр CX використовується як лічильник кількості десяткових розрядів числа. Починаючи з мітки `m2` число ділиться на 10, до остачі додається код символу нуль і одержаний код символу цифри записується до стеку. Це повторюється, доки ціла частина від ділення числа на 10 не стане дорівнювати нулю.

Після цього, починаючи з мітки `m3`, здійснюється виведення символів-цифр числа, починаючи зі старшого розряду.

Наведемо приклад процедури, яка виводить на екран ціле число.

## *Програма 2.1*

```
...
dseg segment para public 'data'
    num dw -23567
dseg ends

...
digit proc
    mov bx, num
    or  bx, bx
    jns m1
    mov al, '-'
    int 29h
    neg bx
m1:
    mov ax, bx
    xor cx, cx
    mov bx, 10
m2:
    xor dx, dx
    div bx
    add dl, '0'
    push dx
    inc cx
    test ax, ax
    jnz m2
m3:
    pop ax
    int 29h
    loop m3
    ret
digit endp
...
```

### *2.1.5 Введення цілого числа*

Можна здійснити за допомогою функції 0Ah переривання DOS 21h, після чого воно знаходиться в буфері у вигляді послідовності символів. Цей рядок необхідно перетворити у додатковий двійковий код.

Спочатку треба перевірити, чи є перший символ знаком “+” чи “-”. Далі кожному цифру-символ треба перетворити на десяткову цифру, віднявши код символу “0”. Одержану цифру треба помножити на відповідну ступінь 10 і добуток пододати.

Для від’ємного числа одержаний код необхідно перевести у додатковий двійковий код командою neg.

## **2.2 Завдання**

1. Скласти процедуру введення і перетворення цілого числа.
2. Скласти і реалізувати програму введення та виведення цілого числа зі знаком та виведення рядка символів.
3. Введення та виведення цілого числа з запрошенням до користувача.

### 2.3 Контрольні питання

1. Подання цілих і дійсних чисел та символів у пам'яті ПК.
2. Вектори переривання, їх розташування у пам'яті.
3. Особливості виконання команд множення MUL та IMUL:
  - a) місце знаходження множників;
  - b) місце знаходження добутку [1]с.53 – 55.
4. Особливості виконання команд ділення DIV та IDIV:
  - a) місце розташування діленого для ділення байтів та слів;
  - b) місце розташування частки (ціле від ділення) та остачі при діленні байтів та слів.
5. Що таке стек і як він працює? [1]с.55 – 56.
6. Команда організації циклів loop та особливості її виконання.

### Комп'ютерний практикум № 3

**Тема:** програмування розгалужених алгоритмів.

#### 3.1 Загальні положення

Існує велика група команд, що вміють приймати рішення про те, яка команда має виконуватися наступною. Рішення приймається в залежності від визначених умов. Умова визначається вибором команди переходу. Існують команди умовного переходу, які дозволяють перевірити:

- ✓ відношення між операндами зі знаком (“більше - менше”);
- ✓ відношення між операндами без знаку (“вище-нижче”);
- ✓ стан арифметичних прапорців *zf*, *sf*, *cf*, *of*, *pf* (але не *af*).

Команди умовного переходу мають однаковий синтаксис:

#### **жсс мітка\_переходу**

Перша літера **ж** походить від англійського *jump* (стрибок), **сс** визначає умову, яку аналізує команда, вираз **мітка\_переходу** може містити лише ту мітку, що знаходиться у поточному сегменті, міжсегментної передачі управління в умовних переходах не дозволяється.

Для того щоб прийняти рішення, куди буде передано управління командою умовного переходу, треба сформулювати умову, на основі якої і буде здійснена подальша передача управління. Джерелами такої умови можуть бути:

- ✓ будь-яка команда, що змінює стан арифметичних прапорців;
- ✓ команда порівняння **cmp**;
- ✓ стан регістра **сх**.

Команда порівняння **cmp** має принцип дії схожий на принцип дії команди віднімання **sub**, але при цьому вона на відміну від останньої не записує результат віднімання на місце першого операнда. Команда **cmp** виконує порівняння і встановлює відповідним чином прапорці. Синтаксис цієї команди:

#### **cmp операнд\_1, операнд\_2.**

Наведемо перелік команд умовного переходу для команди **cmp операнд\_1, операнд\_2.**

**Таблиця 3.1**

Типи операндів	Мнемокод Команд умовного Переходу	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
Зі знаком	JL або JNGE	Операнд_1<операнд_2	sf<>of
Зі знаком	JLE або JNG	Операнд_1<=операнд_2	sf<>of або zf=1
Зі знаком	JG або JNLE	Операнд_1>операнд_2	sf=of та zf=0
Зі знаком	JGE або JNL	Операнд_1=>операнд_2	sf=of
Без знаку	JB або JNAE	Операнд_1<операнд_2	cf=1
Без знаку	JBE або JNA	Операнд_1<=операнд_2	cf=1 або zf=1
Без знаку	JA або JNBE	Операнд_1>операнд_2	cf=0 та zf=0
Без знаку	JAЕ або JNB	Операнд_1=>операнд_2	cf=0

Мнемонічні позначення деяких команд умовного переходу відображають назву прапорця, з яким вони працюють. Мнемокоди команд, назв прапорців і умов переходів наведені у табл.3.2.

Таблиця 3.2

Назва прапорця	Команда умовного переходу	Значення прапорця для здійснення переходу
Прапорець переносу cf	JC	cf=1
Прапорець парності pf	JP	pf=1
Прапорець нуля zf	JZ	zf=1
Прапорець знаку sf	JS	sf=1
Прапорець переповнення of	JO	of=1
Прапорець переносу cf	JNC	cf=0
Прапорець парності pf	JNP	pf=0
Прапорець нуля zf	JNZ	zf=0
Прапорець знаку sf	JNS	sf=0
Прапорець переповнення of	JNO	of=0

До команд умовного переходу належить також наступна команда

**jcxz мітка\_переходу (Jump if cx is Zero).**

Враховуючи, що регістр cx виконує роль лічильника у командах керування циклами та при роботі з ланцюжками символів, команда **jcxz** також використовується при організації циклів. На відміну від інших команд умовного переходу ця команда може адресувати переходи лише на – 128 байт або на +127 байт від наступної за нею команди.

### 3.2 Завдання

Написати програму, яка буде обчислювати значення функції. Номер завдання вибирати за останніми двома числами номеру в заліковій книжці.

Таблиця 3.3

1. $Z = \begin{cases} 8x^2/y & \text{якщо } y \neq 0; x = -5 \\ 6x & \text{якщо } y = 0; x > 3 \\ 1 & \text{в інших випадках} \end{cases}$	2. $Z = \begin{cases} 6x^3/y & \text{якщо } y > 0; x = 5 \\ 38x/5y^2 & \text{якщо } y < 0 \\ 25x^2 & \text{якщо } y = 0 \end{cases}$
3. $Z = \begin{cases} (5x-y^2)/7(t-y) & \text{якщо } x > y; t \neq y \\ 13x+7y+5t & \text{якщо } x \leq y \\ 2xy & \text{в інших випадках} \end{cases}$	4. $Z = \begin{cases} y^2/(10-x+y) & \text{якщо } x+y \neq 10 \\ 35x^2/y & \text{якщо } xy = 10 \\ 1 & \text{в інших випадках} \end{cases}$
5. $Z = \begin{cases} (x+y)/xy & \text{якщо } x > 0; y > 0 \\ 25y & \text{якщо } x < 0; y < 0 \\ 6x & \text{якщо } x > 10; y = 0 \\ 1 & \text{в інших випадках} \end{cases}$	6. $Z = \begin{cases} 34x^2/y(x-y) & \text{якщо } y > 0; x \neq y \\ (1-x)/(1+x) & \text{якщо } y = 0 \\ x^2y^2 & \text{якщо } y < 0 \end{cases}$
7. $Z = \begin{cases} (15x-1)/y(x-y) & \text{якщо } 0 \leq x \leq 10, \\ & y \neq x \\ 35x^2+8x & \text{якщо } x < 0 \\ (10-x)^2 & \text{якщо } x > 10 \end{cases}$	8. $Z = \begin{cases} 8x^2+36/x & \text{якщо } x > 0 \\ (1+x)/(1-x) & \text{якщо } -5 \leq x \leq 0 \\ 10x^2 & \text{якщо } x < -5 \end{cases}$
9. $Z = \begin{cases} 35x^2-15 & \text{якщо } x > 5 \\ 10/x & \text{якщо } 0 < x \leq 5 \\ 215-x & \text{якщо } x \leq 0 \end{cases}$	10. $Z = \begin{cases} (4+x^2)/yx & \text{якщо } x \neq 0, y \neq 0 \\ 25y & \text{якщо } x = 0, y \leq 0 \\ 4x & \text{якщо } y = 0, x \leq 0 \\ x+y & \text{в інших випадках} \end{cases}$

11. $Z = \begin{cases} 54+x^2 / (1+x) & \text{якщо } 1 < x \leq 20 \\ 75x^2-17x & \text{якщо } x \leq 1 \\ 85x / (1+x) & \text{якщо } x > 20 \end{cases}$	12. $Z = \begin{cases} 35x / (1-x^2) & \text{якщо } 1 < x \leq 6 \\ x^3-75 & \text{якщо } x > 6 \\ x^2 & \text{якщо } x \leq 1 \end{cases}$
13. $Z = \begin{cases} (40x^2-23) / x & \text{якщо } 0 < x \leq 7 \\ 38x^3+5 & \text{якщо } x \leq 0 \\ 126 / x & \text{якщо } x > 7 \end{cases}$	14. $Z = \begin{cases} 35 / x + x^3 & \text{якщо } 1 < x \leq 3 \\ x / (1+x^2) & \text{якщо } -1 < x \leq 1 \\ 2x & \text{якщо } x \leq -1 \\ x+y & \text{в інших випадках} \end{cases}$
15. $Z = \begin{cases} (36x^2-17x+1) / x & \text{якщо } 0 < x \leq 6 \\ 35x^2-2x+1 & \text{якщо } x \leq 0 \\ 1250 / x & \text{якщо } x > 6 \end{cases}$	16. $Z = \begin{cases} (1+x^2) / (1-x) & \text{якщо } x \leq -5 \\ x^2+375 & \text{якщо } -5 < x \leq 5 \\ x^2 / 10 & \text{якщо } x > 5 \end{cases}$
17. $Z = \begin{cases} x^3 / y & \text{якщо } x > 0, y > 0 \\ x / 2y & \text{якщо } y < 0, x > 0 \\ 3x^2 & \text{якщо } y = 0 \\ 1 & \text{в інших випадках} \end{cases}$	18. $Z = \begin{cases} (12x^3-9x^2+16x)/(x+1) & \text{якщо } 0 < x \leq 9 \\ 1 & \text{якщо } x \leq 0 \\ x^2 / 10 & \text{якщо } x > 9 \end{cases}$
19. $Z = \begin{cases} ax^2+b / x & \text{якщо } x > 0 \\ a+2b & \text{якщо } x = 0 \\ ax^2-bx & \text{якщо } x < 0 \end{cases}$	20. $Z = \begin{cases} (2x^2-y) / (x-y) & \text{якщо } x > y \\ 10x^2-y & \text{якщо } x = y \\ (x^2-y) / (x+y) & \text{якщо } x < y \end{cases}$
21. $Z = \begin{cases} 5[(2+x)^{-1}+(3(1+x))^{-1}] & \text{якщо } x > 0 \\ 5 & \text{якщо } x = 0 \\ 5x^2 / (1-x) & \text{якщо } x < 0 \end{cases}$	22. $Z = \begin{cases} (x-1) & \text{якщо } x < -1 \\ 0 & \text{якщо } x = -1 \\ (x^3+2x^2+11) / (2x+1) & \text{якщо } x > -1 \end{cases}$
23. $Z = \begin{cases} x-1 & \text{якщо } x < 10 \\ (3x^2+4) / (x-2) & \text{якщо } x = 10 \\ (7x^2-56) / (2x-5) & \text{якщо } x > 10 \end{cases}$	24. $Z = \begin{cases} x+3 & \text{якщо } x \leq 0 \\ 4x^2 / (x+1) & \text{якщо } 0 < x < 2 \\ (x^2-1) / (2x+5) & \text{якщо } 2 \leq x \leq 4 \\ (x^3-1) / (x^2+1) & \text{якщо } x > 4 \end{cases}$
25. $Z = \begin{cases} (x^3-2x^2+1) / (x^2+1) & \text{якщо } x < 9 \\ 1 & \text{якщо } x = 9 \\ (x^2-8x+6) / (x^2+1) & \text{якщо } x > 9 \end{cases}$	26. $Z = \begin{cases} (x^3+2x^2) / x & \text{якщо } x < 0 \\ x^4 & \text{якщо } 0 \leq x < 5 \\ (x^2-8x+6) / x^2 & \text{якщо } x \geq 5 \end{cases}$
27. $Z = \begin{cases} (x^3-7x) / (x^2+17) & \text{якщо } x < 0 \\ 127 & \text{якщо } x = 0 \\ (x^2+8x-7) / x^3 & \text{якщо } x > 0 \end{cases}$	28. $Z = \begin{cases} (x^3-18x) / x^3 & \text{якщо } x < 8 \\ 549x & \text{якщо } x = 8 \\ (x^2-18x-16) / (x^5-89) & \text{якщо } x > 8 \end{cases}$
29. $Z = \begin{cases} 9[(12+x)^{-1}+3x^{-1}] & \text{якщо } x < 0 \\ 1 & \text{якщо } x = 0 \\ (x^3+6) / (x^2+19) & \text{якщо } x > 0 \end{cases}$	30. $Z = \begin{cases} (x^3+1) / (x^3-1) & \text{якщо } x < 7 \\ 32800 & \text{якщо } x = 7 \\ (x^7 / (x^6+19)) & \text{якщо } x > 7 \end{cases}$
31. $Z = \begin{cases} x-3 & \text{якщо } x < 15 \\ (3x^3+4) / (x-4) & \text{якщо } x = 15 \\ (9x^2-58) / (2x-5) & \text{якщо } x > 15 \end{cases}$	32. $Z = \begin{cases} x+3 & \text{якщо } x \leq 0 \\ 9x^2 / (x+1) & \text{якщо } 0 < x < 10 \\ (x^2-1) / (6x+5) & \text{якщо } 10 \leq x \leq 24 \\ (x^3-9) / (x^2+1) & \text{якщо } x > 24 \end{cases}$

33. $Z = \begin{cases} x^2 / y & \text{якщо } x > 0, y > 0 \\ x / 9y & \text{якщо } y < 0, x > 0 \\ 3+x^2 & \text{якщо } y = 0 \\ 1 & \text{в інших випадках} \end{cases}$	34. $Z = \begin{cases} (19x^3+9x^2-16x)/(x+1) & \text{якщо } 0 < x \leq 19 \\ 1 & \text{якщо } x \leq 0 \\ x^2 / 17 & \text{якщо } x > 19 \end{cases}$
35. $Z = \begin{cases} (a+x^2+b) / x & \text{якщо } x > 1 \\ (a+2)/b & \text{якщо } x = 1 \\ a/x^2-bx & \text{якщо } x < 1 \end{cases}$	36. $Z = \begin{cases} (2x^3-y) / (x+y) & \text{якщо } x > y \\ 17x^2-y & \text{якщо } x = y \\ (x^2+y) / (x-y) & \text{якщо } x < y \end{cases}$

### 3.3 Контрольні питання

1. Команда безумовного переходу та її особливості.
2. Команди умовного переходу.
3. Команда порівняння СМР.
4. Як здійснити умовний перехід на відстань, більшу за 128 байт?



## Комп'ютерний практикум № 4

Тема: масиви.

### 4.1 Загальні положення

Дамо формальне визначення: *масив* – структурований тип даних, який складається з деякого числа елементів одного типу.

Спеціальних засобів опису масивів у програмах асемблера не існує. В разі необхідності використання масиву потрібно моделювати його одним із нижче наведених способів:

- ✓ перелік елементів масиву у полі операндів однієї з директив опису даних:  
`mas dd 1,2,3,4,5`
- ✓ використовуючи оператор повторення `dup`:  
`mas dw 5 dup (0)`  
такий спосіб визначення застосовується для резервування пам'яті з метою розміщення та ініціалізації масиву;

При роботі з масивами необхідно пам'ятати, що їх елементи розміщені у пам'яті комп'ютера послідовно. Для процесора байдуже з чим він у даний момент працює: чи це є елемент масиву, чи структури, чи якась інша змінна. Теж саме можна сказати і про індекси елементів масиву. Асемблер і не підозрює про їх існування. Щоб локалізувати окремий елемент масиву, треба до його імені додати індекс. Отже в асемблері індекси символів – це звичайні адреси, працюють з якими, щоправда, інакше. В загальному випадку для отримання адреси елемента в масиві необхідно до початкової (базової) адреси масиву додати добуток індексу (номер елемента мінус одиниця) цього елемента на розмір елемента масиву:

**адреса + ((номер\_елемента - 1) · розмір\_елемента).**

Розглянемо індексний та базовий індексний типи адресації, що реалізуються за допомогою відповідних реєстрів і дозволяють ефективно працювати з масивами у пам'яті.

- Індексна адресація зі зміщенням. Ефективна адреса формується з двох компонентів:
  - ✓ постійного (базового) – вказівкою прямої адреси масиву у вигляді імені ідентифікатора, що позначає початок масиву;
  - ✓ змінного (індексного) – вказівкою імені індексного реєстру.

Наприклад:

```
mas dw 0,1,2,3,4,5
```

...

```
mov si, 4
```

**; перенести 3-ій елемент масиву в реєстр ax:**

```
mov ax, mas[ si ]
```

- Базова індексна адресація зі зміщенням. Ефективна адреса формується максимум з трьох компонентів:
  - ✓ постійного (необов'язкового компонента), яким може бути пряма адреса масиву виражена ім'ям ідентифікатора, що визначає початок масиву;
  - ✓ змінного (базового) – визначенням імені базового реєстру;
  - ✓ змінного (індексного) – визначенням імені індексного реєстру.

```
mov mas[ bx ][ si ]
```

Слід зауважити, що цей вид адресації зручно використовувати, працюючи з двомірними масивами.

Базовим реєстром може виступати будь-який реєстр загального призначення. У ролі індексного реєстра може також використовуватися будь-який реєстр загального призначення окрім `sp`.

Для демонстрації основних прийомів роботи з масивами розглянемо програму, яка виконує сортування масиву у порядку зростання.

### Програма 4.1

```
STK SEGMENT STACK
DB 64 DUP ( "?" )
STK ENDS
```

```
DATA SEGMENT PARA PUBLIC "DATA"
MES1 DB 0ah, 0dh 'Початковий масив - $', 0ah, 0ah
; деякі повідомлення
MES2 DB 0ah, 0dh 'Відсортований масив - $', 0ah, 0ah
N EQU 9 ; кількість елементів у масиві, рахуючи від 0
MAS DW 2, 7, 0, 1, 9, 3, 6, 5, 8 ; задання елементів масиву
TMP DW 0 ; змінні для роботи з масивом
I DW 0
J DW 0
DATA ENDS
```

```
CODE SEGMENT PARA PUBLIC "CODE"
MAIN PROC FAR
ASSUME CS: CODE, DS: DATA, SS: STK
PUSH DS
XOR AX, AX
PUSH AX
MOV AX, DATA
MOV DS, AX
; вивід на екран початкового масиву
MOV AH, 09h
LEA DX, MES1
INT 21h ; вивід повідомлення mes1
MOV CX, 10
MOV SI, 0

SHOW_PRIMARY: ; вивід значень елементів початкового масиву на екран
MOV DX, MAS[SI]
ADD DL, 30h
MOV AH, 02h
INT 21h
ADD SI, 2
LOOP SHOW_PRIMARY

MOV I, 0 ; ініціалізація i, внутрішній цикл по j

INTERNAL:
MOV J, 9 ; ініціалізація j, перехід на тіло циклу
JMP CYCL_J
EXCHANGE:
MOV BX, I
```

```

SHL BX, 1
MOV AX, MAS[BX]
MOV BX, J
SHL BX, 1
CMP AX, MAS[BX]
JLE LESS

MOV BX, I
SHL BX, 1 ; помножуємо на 2, оскільки елементи – слова
MOV TMP, AX

MOV BX, J
SHL BX, 1
MOV AX, MAS[BX]
MOV BX, I
SHL BX, 1
MOV AX, MAS[BX]

MOV BX, J
SHL BX, 1
MOV AX, TMP
MOV AX, MAS[BX]
LESS: ; пересування далі по масиву у внутрішньому циклі
    DEC J
CYCL_Y: ; тіло циклу по j
    MOV AX, J
    CMP AX, I
    JG EXCHANGE

    INC I
    CMP I, N
    JL INTERNAL

; вивід відсортованого масиву
MOV AH, 09h
LEA DX, MES2
INT 21h

MOV CX, 10
MOV SI, 0
SHOW: ; вивід значень елемента на екран
    MOV DX, MAS[SI]
    ADD DL, 30h
    MOV AH, 02h
    INT 21h
    ADD SI, 2
    LOOP SHOW
MAIN ENDP
CODE ENDS
END MAIN

```

## **4.2 Завдання**

Скласти програму на нижче наведені завдання:

1. Написати програму знаходження суми елементів масиву.
2. Написати програму пошуку максимального (або мінімального) елемента масиву.
3. Написати програму пошуку координат всіх входжень заданого елемента в двовірному масиві.
4. Написати програму сортування одномірного масиву цілих чисел загального вигляду.

## **4.3 Контрольні питання**

1. Команди організації циклів.
2. Рядкові команди та особливості їх використання.
3. Методи адресації за базою, з індексуванням, з подвійним індексуванням.

## Комп'ютерний практикум № 5

**Тема:** макрозасоби мови асемблер.

### 5.1 Загальні положення

Для розв'язання локальних задач та полегшення роботи у певній проблемній галузі використовують апарат *макроасемблера*. Цей апарат є дуже потужним і важливим при написанні асемблерівських програм інструментом, який дозволяє усунути або, принаймні, мінімізувати нижче перераховані недоліки:

- ✓ повторення деяких ідентичних або таких, що незначно відрізняються фрагментів програми;
  - ✓ необхідність включення в кожен програму фрагментів коду, які були використані в інших програмах;
  - ✓ обмеженість набору команд;
- тощо.

До найпростіших макрозасобів мови асемблер можна віднести псевдооператори **equ** та “**=**”. Різниця між цими операторами у тому, що за допомогою **equ** є можливість ставити у відповідність ідентифікатору не лише числові вирази, а й текстові рядки, у той час, як “**=**” використовується тільки з числовими виразами.

Існують також засоби для роботи з текстовими макросами, оголошеними за допомогою псевдооператора **equ**:

- **ідентифікатор catstr рядок\_1, рядок\_2, ...** – результатом буде рядок, що складається з *рядок\_1* та *рядок\_2*;
- **ідентифікатор substr рядок, номер\_позиції, розмір** – результатом буде частина заданого рядку;
- **ідентифікатор instr номер\_поч\_позиції, рядок\_1, рядок\_2** – *ідентифікатор* приймає значення номера позиції, з якої *рядок\_1* та *рядок\_2* починають співпадати, якщо такого співпадання не має, то *ідентифікатор* приймає значення 0;
- **ідентифікатор sizestr рядок** – результатом є довжина *рядка*.

Наведені директиви зручно використовувати при розробці макрокоманд. Макрокоманда описується наступним чином:

```
ім'я макрокоманди macro список_формальних_аргументів  
тіло макровизначення  
endm
```

Макровизначення можуть бути розташовані:

- ✓ до сегментів коду і даних;
- ✓ в окремому файлі, який необхідно підключити на початку вихідного тексту програми директивою **include ім'я\_файлу**;

за допомогою директиви **purge ім'я\_макросу1, ім'я\_макросу2, ...** можна не підключати деякі макроси розміщені у файлі, якщо у даній програмі вони непотрібні.

Наведемо приклад програми, що використовує макрозасоби. Програма буде перетворювати двозначкове число, задане у шістнадцятковій системі, у двійкову систему .

### Програма 5.1

```
INIT_DS MACRO
```

```
; Макрос налаштування ds на сегмент даних.
```

```
MOV AX, DATA
```

```
MOV DS, AX
```

ENDM

OUT\_STR MACRO STR

; макрос виводу рядка на екран.  
; на вході – рядок, що виводиться.  
; на виході – повідомлення на екрані.

```
PUSH AX
MOV AH, 09h
MOV DX, OFFSET STR
INT 21h
POP AX
```

ENDM

CLEAR\_R MACRO RG

; очищення регістра rg.

```
XOR RG, RG
```

ENDM

GET\_CHAR MACRO

; введення символу.  
; введений символ в al.

```
MOV AH, 1h
INT 21h
```

ENDM

CONV\_16to2 MACRO

; макрос перетворення символу шістнадцяткової цифри  
; у її двійковий еквівалент в al.

```
SUB DL, 30h
CMP DL, 9h
JLE $+5
SUB DL, 7h
```

ENDM

EXIT MACRO

; макрос завершення програми.

```
MOV AX, 4c00h
INT 21h
```

ENDM

DATA SEGMENT PARA PUBLIC "DATA"

MESSAGE DB "Введіть дві шістнадцяткові цифри: \$"

DATA ENDS

STK SEGMENT STACK

DB 256 DUP ("?")

STK ENDS

CODE SEGMENT PARA PUBLIC "CODE"

```
ASSUME CS : CODE, DS : DATA, SS : STK
```

```
MAIN PROC
  INIT_DS
  OUT_STR MESSAGE
  CLEAR_R AX
  GET_CHAR
  MOV DL, AL
  CONV_16to2
  MOV CL, 4h
  SHL DL, CL
  GET_CHAR
  CONV_16to2
  ADD DL, AL
  XCHG DL, AL
  EXIT
MAIN ENDP
CODE ENDS
END MAIN
```

## 5.2 Завдання

Скласти програму на нижче наведені завдання:

- 1) переписати *Програму 2.1* з використанням макросів;
- 2) переписати *Програму 3.1*, з використанням макросів;
- 3) переписати *Програму 4.1* з використанням макросів.

## 5.3 Контрольні питання

1. Як виконуються макрокоманди?
2. Чим відрізняється виконання процедур від виконання макрокоманд?
3. Особливості використання міток у макровизначеннях.

## *Література*

1. Абель Питер Ассемблер. Язык и программирование для IBM PC IBM PC Assembly Language and Programming Издательства: Корона-Век, Энтроп, 2007 г., 736 стр.
2. Абель Питер Язык Ассемблера для IBM PC и программирования : Пер. с англ. / П. Абель ; пер. Ю. В. Сальников. - М. : Высшая школа, 1992. - 447[1] с. : ил. - ISBN 5-06-001518
3. Ассемблер Z-80 / Ред. У.Тант. - М. : ВА принт, 1993. - 123 с. - (ZX Spectrum)
4. Баазе Сара Ассемблер мини-ЭВМ VAX-11 : Пер. с англ. / Сара Баазе; Пер. В. Л. Григорьев. - М. : Финансы и статистика, 1988. - 416 с. - ISBN 5-279-00062-0
5. Брэдли Д. Программирование на языке ассемблере для персональных ЭВМ фирмы IBM – Москва: Радио и связь, 1988. – 447с.
6. Галисеев Г.В. Ассемблер IBM PC : самоучитель / Г. В. Галисеев. - М. : Вильямс, 2004. - 303[1] с. : ил., -ISBN 5-8459-0708-X
7. Жуков Андрей Ассемблер : [Руководство по программированию] / А. В. Жуков, А. А. Авдюхин. - СПб. : БХВ-Петербург, 2002. - 444[4] с. : ил., табл. - (Самоучитель) (Программирование на машинном уровне). - ISBN 5-94157-133-X
8. Зубков С.В. Assembler. Для DOS, Windows и Unix : / С. В. Зубков. - М. : ДМК, 1999. - 637[3] с. : ил. - ISBN 5-89818-019-2
9. Использование Turbo Assembler при разработке программ : учебное пособие. - Киев : Диалектика, 1994. - 288 с. : ил. - ISBN 5-7707-5043
10. Кац Е. Я. Системное программирование на ПЭВМ типа IBM PC: Системные ресурсы IBM PC и язык системного программирования Ассемблер : Учебное пособие / Ефим Яковлевич Кац - Саратов : СГТУ, 1993. - 99 с. : ил. - ISBN 5-230-07263-6
11. Лин Вэн. PDP-11 и VAX-11. Архитектура ЭВМ и программирование на языке ассемблера : Пер. с англ. / Вэн Лин. - М. : Радио и связь, 1989. - 320 с. : ил. - ISBN 5-256-00299-6
12. Майко Г. В. Ассемблер для IBM PC / Г.В. Майко. - М. : Бизнес-Информ, 1997 ; М. : Сирин, 1997. - 212 с.
13. Нортон Питер Язык Ассемблера для IBM PC : Пер. с англ. / П. Нортон, Дж. Соухэ. - М. : Компьютер, 1993. - 351 с. - ISBN 5-88201-008-X
14. Пильщиков В.И. Программирование на языке ассемблере на IBM PC. – Москва: Диалог-МИФИ, 1999. – 288с. ISBN 5-86404-051-7
15. Пирогов В.Ю. Ассемблер для Windows Серия: Профессиональное программирование Издательство: БХВ-Петербург, 2007 г., 896 стр.
16. Пустоваров В. И. Язык ассемблера в программировании информационных и управляющих систем : Учебное пособие / В.И. Пустоваров. - Киев : ВЕК, 1996 ; М. : ЭНТРОП, 1996 ; М. : БИНОМ-УНИВЕРСАЛ, 1996. - 304 с.
17. Скэнлон Лео Персональные ЭВМ IBM PC XT. Программирование на языке ассемблера : Пер. с англ. / Лео Скэнлон. - 2-е изд., стереотип. - М. : Радио и связь, 1991. - 336 с. : ил. - ISBN 5-256-00300-3 (в пер.)
18. Старостин Н., Старостин О.В. Язык Assembler для программирования Издательство: Познавательная книга (ЗАО), Познавательная книга +, 2000 г. 416 стр. ISBN: 5-8321-0107-3
19. Шнайдер А. Язык ассемблера для персонального компьютера фирмы IBM : Пер. с англ. / А. Шнайдер. - М. : Мир, 1988. - 405с. : ил. - ISBN 5-03-000394-0

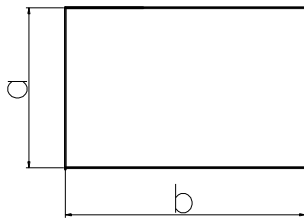


20. Юров В., Хорошенко С. *Assembler: учебный курс* – Санкт-Петербург: ПитерКом, 1999. – 672с. ISBN 5-314-00047-4
21. Белецкий Ян. *Энциклопедия языка Си*. М., Мир, 1992. – с.687.

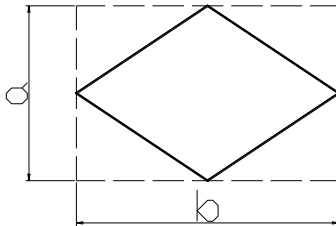
Виконання схем алгоритмів [4]

*Основні символи, що використовуються при побудові схем*

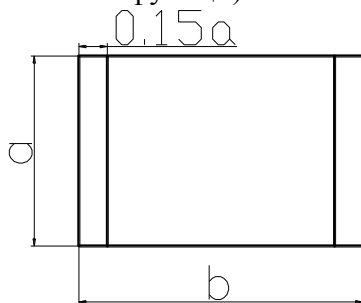
1. Процес – виконання операції чи групи операцій, у результаті якого змінюється значення, форма представлення чи розташування даних



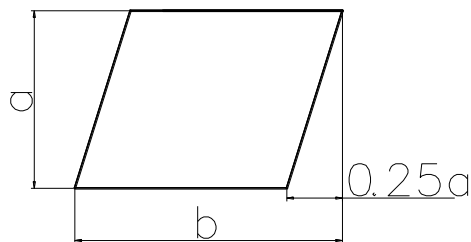
2. Рішення – вибір напрямку виконання алгоритму в залежності від деяких змінних умов



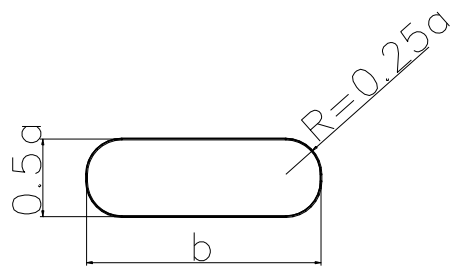
3. Визначений процес – використання окремо описаних алгоритмів чи програм (бібліотечні функції)



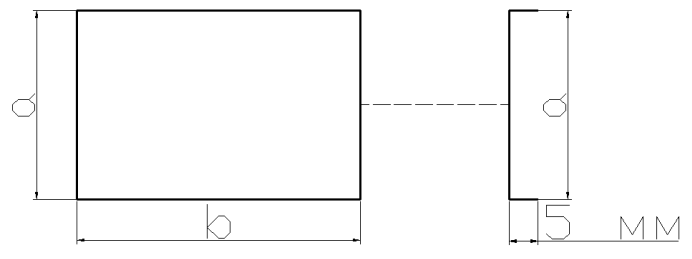
4. Символ вводу-виводу даних:



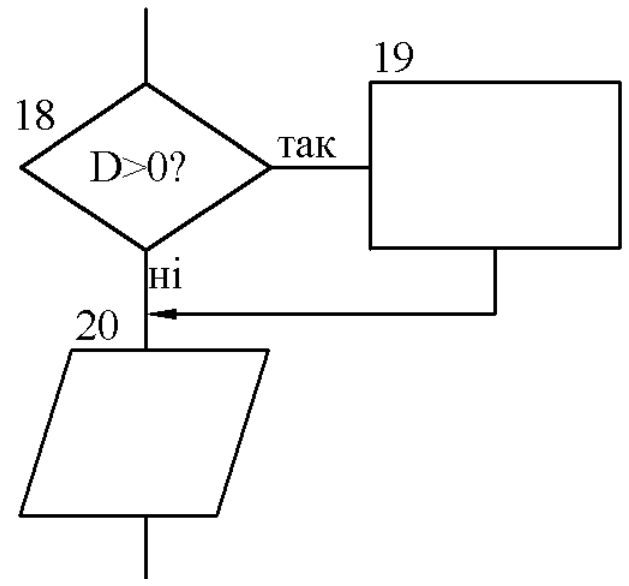
5. Символ початку, кінця, призупинення програми:



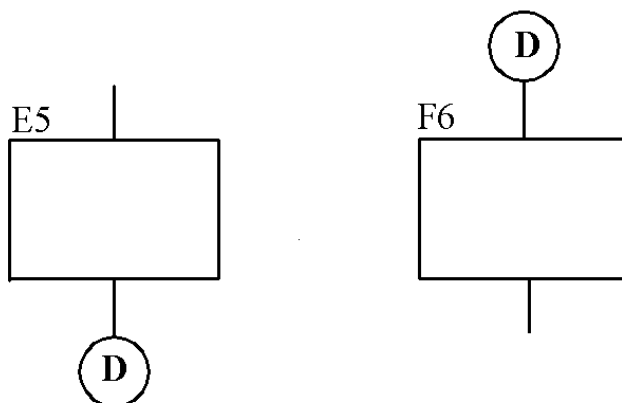
6. Коментарі до схеми:



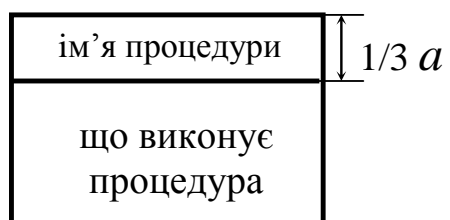
6. Фрагмент схеми з розгалуженням



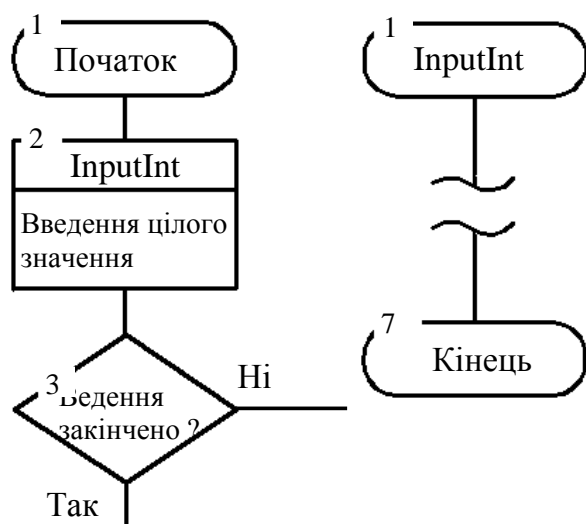
7. Використання з'єднувача



8. Визначений процес – використання раніше створених розробником процедур та макросів.



9. Схеми на кількох рівнях



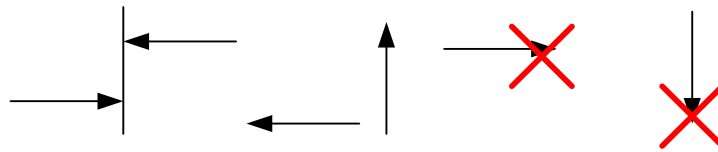
При виконанні схем алгоритмів основний розмір  $a$  повинний бути обраний з ряду 10, 15, 20 мм. Допускається збільшити це значення на число, кратне 5. Розмір  $b$  складає  $1,5a$ , при ручному виконанні схем допускається  $b = 2a$ .

Символи нумеруються порядковими номерами чи координатами зон (арабські цифри або великі літери латинського алфавіту).

При ручному виконанні схеми в її межах допускається застосовувати не більш двох суміжних типорозмірів символів.

Лінії потоків повинні бути паралельні лініям зовнішньої рамки схеми.

Напрямок лінії потоку згори донизу і зліва направо стрілкою можна не позначати. В інших випадках стрілки обов'язкові. (10)



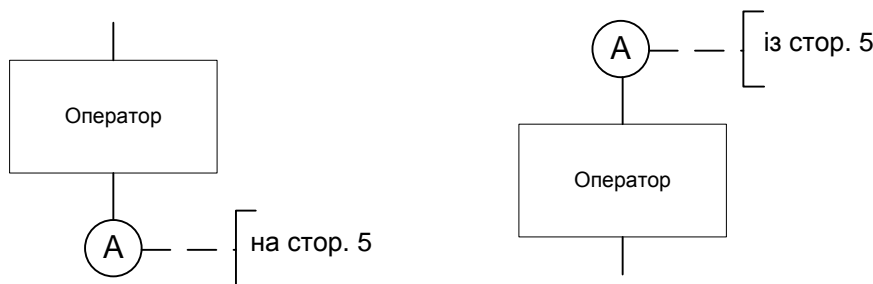
Відстань між окремими символами схеми повинна бути не менше 10 мм.

Для пояснення процесів всередині символів розміщують короткі записи. Для зручності читання схем використовують коментарі до символів чи ліній потоку.

У межах одного аркуша лінії потоку можна розривати із застосуванням з'єднувачів, усередині яких міститься їх ідентифікатор. (11)

Можливі варіанти відображення ходу рішення при кількості можливих виходів два і більше показані на схемах 8,9.

Якщо ж лініями потоку з'єднують символи на різних аркушах, то до кожного з'єднувача додається коментар – з якої сторінки, або до якої сторінки.



Схеми можуть бути побудовані на кількох рівнях.

Функції, які деталізуються, на крупних схемах позначаються прямокутником (12), в середині якого проведена горизонтальна лінія на  $1/3$  висоти символу. Посередині над цією лінією пишеться назва деталізованої функції.

В символі початку деталізованої функції пишеться її назва.

Приклади оформлення схем на більш поширенні комбінації при описі алгоритмів програм:

## Виконання по умові

Умовний оператор

*JCC мітка\_переходу*

*оператор2*

Якщо *результат*  $\neq 0$  (true)  $\Rightarrow$  виконується  
*переходить на мітку*

Якщо *результат* = 0 (false)  $\Rightarrow$  виконується



**Приклад коду:**

СМР АХ, ВХ

JE *суcl* ; якщо АХ = ВХ, то перейти по мітці *суcl*

JCХZ *m1* ; обійти цикл, якщо СХ=0

*суcl*: ; деякий цикл

LOOP *суcl*

*m1*: ...

### *Циклічне виконання*

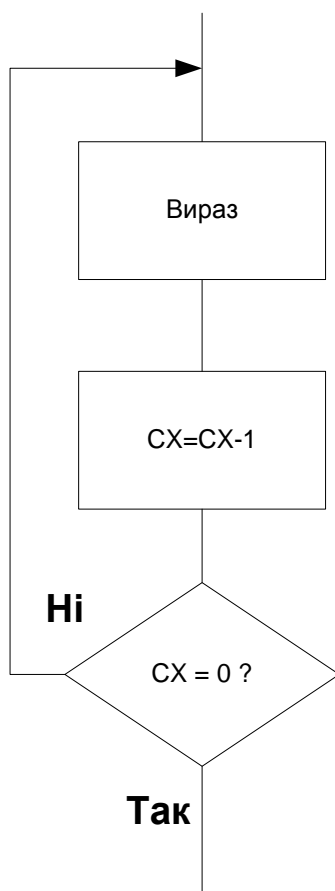
**LOOP** мітка - зменшує регістр **CX** на 1 і здійснює перехід на певну мітку, якщо зміст  $CX \neq 0$ , якщо  $CX = 0$ , здійснюється перехід на наступну за **LOOP** команду.

**MOV CX, LOOP\_COUNT**

**BEGIN\_LOOP:**

<тіло цикла>

**LOOP BEGIN\_LOOP**



*Приклад виконання звіту*

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки  
Кафедра технічної кібернетики

*Звіти до комп'ютерних практикумів з модуля  
«Системне програмування»*

**Прийняв**  
доцент кафедри ТК  
Лісовиченко О.І.  
“...” ..... 20xx р.

**Виконав**  
Студент групи ІК-ХХ  
Іванов І.І.

Київ – 20XX



## 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. Скопіювати 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 .map-файлу.
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

; описание сегмента стека

```
STSEG SEGMENT PARA STACK "STACK"
```

```
DB 64 DUP ("STACK")
```

```
STSEG ENDS
```

; описание сегмента данных

```
DSEG SEGMENT PARA PUBLIC "DATA"
```

```
SOURCE DB 10, 20, 30, 40
```

```
DEST DB 4 DUP ( "?")
```

```
DSEG ENDS
```

; описание сегмента кода

```
CSEG SEGMENT PARA PUBLIC "CODE"
```

; код основной функции

```
MAIN PROC FAR
```

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

; адрес возврата

```
PUSH DS
```

```
MOV AX, 0 ; або XOR AX, AX
```

```
PUSH AX
```

; инициализация DS

```
MOV AX, DSEG
```

```
MOV DS, AX
```

; обнуление массива

```
MOV DEST, 0
```

```
MOV DEST+1, 0
```

```
MOV DEST+2, 0
```

```
MOV DEST+3, 0
```

; пересылка

```
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 4.1 09/10/12 21:47:30 Page 1  
lab1.asm

```

1          ; описание сегмента стека
2 0000     STSEG SEGMENT PARA STACK "STACK"
3 0000 40*(53 54 41 43 4B) DB 64 DUP ("STACK")
4 0140     STSEG ENDS
5          ; описание сегмента данных
6 0000     DSEG SEGMENT PARA PUBLIC "DATA"
7 0000 0A 14 1E 28 SOURCE DB 10, 20, 30, 40
8 0004 04*(3F) DEST DB 4 DUP ( "?")
9 0008     DSEG ENDS
10         ; описание сегмента кода
11 0000     CSEG SEGMENT PARA PUBLIC "CODE"
12         ; код основной функции
13 0000     MAIN PROC FAR
14         ASSUME CS: CSEG, DS: DSEG, SS: STSEG
15         ; адрес возврата
16 0000 1E   PUSH DS
17 0001 B8 0000     MOV AX, 0 ; або XOR AX, AX
18 0004 50   PUSH AX
19         ; инициализация DS
20 0005 B8 0000s     MOV AX, DSEG
21 0008 8E D8     MOV DS, AX
22         ; обнуление массива
23 000A C6 06 0004r 00     MOV DEST, 0
24 000F C6 06 0005r 00     MOV DEST+1, 0
25 0014 C6 06 0006r 00     MOV DEST+2, 0
26 0019 C6 06 0007r 00     MOV DEST+3, 0
27         ; пересылка
28 001E A0 0000r     MOV AL, SOURCE
29 0021 A2 0007r     MOV DEST+3, AL
30 0024 A0 0001r     MOV AL, SOURCE+1
31 0027 A2 0006r     MOV DEST+2, AL

```

```

32 002A A0 0002r      MOV AL, SOURCE+2
33 002D A2 0005r      MOV DEST+1, AL
34 0030 A0 0003r      MOV AL, SOURCE+3
35 0033 A2 0004r      MOV DEST, AL
36
37 0036 CB           RET
38
39 0037           MAIN ENDP
40
41 0037           CSEG ENDS
42
43           END MAIN
_Turbo Assembler  Version 4.1    09/10/12 21:47:30    Page 2
Symbol Table

```

Symbol Name	Type	Value
??DATE	Text	"09/10/12"
??FILENAME	Text	"lab1 "
??TIME	Text	"21:47:30"
??VERSION	Number	040A
@CPU	Text	0101H
@CURSEG	Text	CSEG
@FILENAME	Text	LAB1
@WORDSIZE	Text	2
DEST	Byte	DSEG:0004
MAIN	Far	CSEG:0000
SOURCE	Byte	DSEG:0000

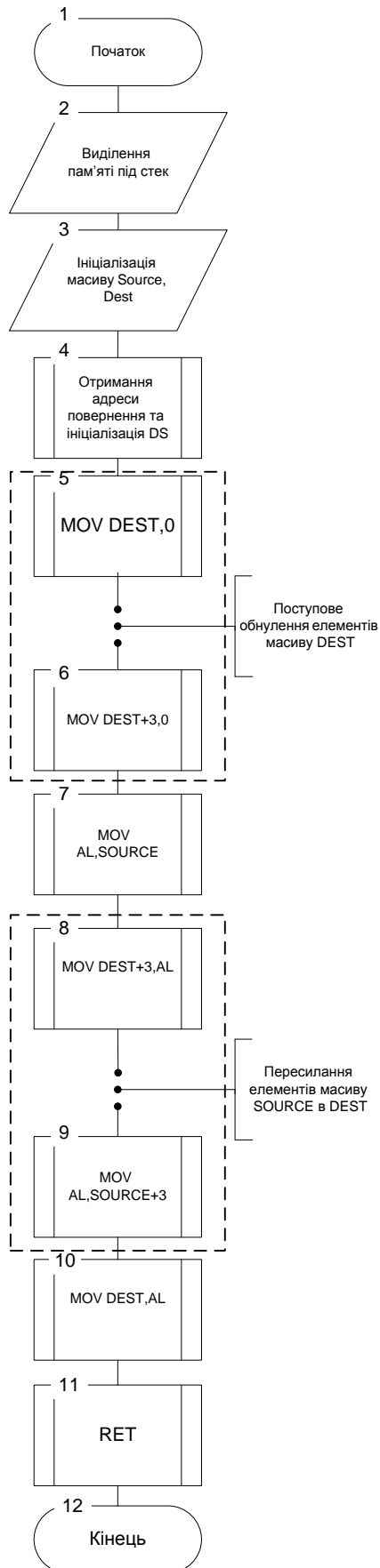
Groups & Segments	Bit	Size	Align	Combine	Class
CSEG	16	0037	Para		Public CODE
DSEG	16	0008	Para		Public DATA
STSEG	16	0140	Para		Stack STACK

**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



## Вікно DUMP

```
ds:0000 0A 14 1E 28 3F 3F 3F 3F 00 00 (????
ds:0008 00 00 00 00 00 00 00 00
ds:0010 1E B8 00 00 50 B8 8D 08 00 00 P 7 i
ds:0018 8E D8 C6 06 04 00 00 C6 00 00 00
ds:0020 06 05 00 00 C6 06 06 00 00 00 00
```

## До виконання програми:

### Масив SOURCE:

```
ds:0000 0A 14 1E 28 3F 3F 3F 3F 00 00 (????
ds:0008 00 00 00 00 00 00 00 00
ds:0010 1E B8 00 00 50 B8 8D 08 00 00 P 7 i
ds:0018 8E D8 C6 06 04 00 00 C6 00 00 00
ds:0020 06 05 00 00 C6 06 06 00 00 00 00
```

### Масив DEST

```
ds:0000 0A 14 1E 28 3F 3F 3F 3F 00 00 (????
ds:0008 00 00 00 00 00 00 00 00
ds:0010 1E B8 00 00 50 B8 8D 08 00 00 P 7 i
ds:0018 8E D8 C6 06 04 00 00 C6 00 00 00
ds:0020 06 05 00 00 C6 06 06 00 00 00 00
```

## Після виконання програми

### Масив DEST

```
ds:0000 0A 14 1E 28 28 1E 14 0A 00 00 ((A)
ds:0008 00 00 00 00 00 00 00 00
ds:0010 1E B8 00 00 50 B8 8D 08 00 00 P 7 i
ds:0018 8E D8 C6 06 04 00 00 C6 00 00 00
ds:0020 06 05 00 00 C6 06 06 00 00 00 00
```

### Висновок:

1. В текстовому редакторі біло створено файл типу .asm.
2. Скомпілював програму, включивши потрібні опції для налагоджувача та створення файлу лістингу типу .lst.
3. Ознайомився зі структурою файлу .lst. Розглянув структури машинних команд.
4. Після усунення помилок, скомпонував .obj-файл програми, включивши опції для налагодження та створення .map-файлу.
5. Відкрив файл карти пам'яті (.map-файл) та подивився на адреси початку та кінця всіх сегментів програми.
6. Завантажив програму налагоджувача td.exe та мій одержаний .exe-файл програми.
7. У вікні CPU у полі DUMP подивився на початкову адресу сегмента даних. В сегменті даних знайшов масиви SOURCE та DEST. Дані у масиві SOURCE подаються у шістнадцятковій системі.
8. У покроковому режимі за допомогою клавіші F7 виконав програму. Програма коректно виконує поставлену задачу.