МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя

Кафедра комп’ютерних систем та мереж

МЕТОДИЧНІ ВКАЗІВКИ ДЛЯ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни «Системне програмування»
для студентів денної та заочної форми навчання
спеціальності 123 «Комп’ютерна інженерія»

Тернопіль 2020

Методичні вказівки для виконання лабораторних робіт з дисципліни «Системне програмування» для студентів денної та заочної форми навчання розроблені у відповідності з навчальним планом спеціальності 123 «Комп'ютерна інженерія» / Уклад. Паламар А.М., Паламар М.І. – Тернопіль: ТНТУ, 2020. – 70 с.

Укладачі: Паламар А.М., проф., д.т.н. Паламар М.І.

Рецензент:

Відповідальний за випуск: зав. каф. комп'ютерних систем та мереж, к.т.н., доц. Осухівська Г.М.

Затверджено на засіданні кафедри комп'ютерних систем та мереж, протокол № ___ від «___» _____ 2020 р.

Методичні вказівки складені з врахуванням методичних розробок інших вищих закладів освіти, а також матеріалів літературних джерел, перелічених в списку.

ЗМІСТ

ВСТУП

Дисципліна «Системне програмування» є складовою частиною підготовки студентів за спеціальністю 123 «Комп'ютерна інженерія».

Метою викладання дисципліни «Системне програмування» є формування знань у студентів з основ системного програмування, формування навиків і вмінь в області організації та функціонування ЕОМ, структури операційних систем та їх взаємодії з прикладними програмами, набуття студентами навичок і вмінь для самостійної розробки системних програмних засобів та прикладних програм.

Завданням вивчення дисципліни є ознайомлення студентів з мовою програмування Асемблер – мовою низького рівня, яка максимально наближена до апаратних засобів ЕОМ, формування знань та навичок для створення компонентів операційних систем та системного програмного забезпечення.

Для закріплення знань та навичків студентам необхідно виконати низку лабораторних робіт. Лабораторна робота – невеликий науковий звіт, що узагальнює проведену студентом роботу, яку представляють для захисту викладачу. До лабораторних робіт пред’являється низка вимог, основною з яких є повний, вичерпний опис всієї проведеної роботи, що дозволяє оцінити отримані результати, міру виконання задання та професійної підготовки студентів.

Звіт по лабораторній роботі друкується або пишеться студентом на одній стороні аркуша паперу формату 210х297 мм (А4). При цьому необхідно залишати поля: зліва – 25 мм, справа – 10 мм, поверх – 20 мм, знизу – 15 мм.

Звіт повинний включати наступні пункти:

1. Мета лабораторної роботи.
2. Теоретичні відомості.
3. Індивідуальне завдання (згідно варіанту) лабораторної роботи.
4. Основні етапи та результати виконання роботи.
5. Висновки.

ЛАБОРАТОРНА РОБОТА №1

ТЕМА: Написання простої програми на мові Assembler.

МЕТА: Ознайомитись з процесом написання та налагодження програми на мові Асемблер. Вивчити основи оформлення типових програм на асемблері.

1. Теоретичні відомості

1.1 Структура програми на мові асемблер

При вивченні будь-якої мови програмування традиційним є написання програми, що виводить на екран повідомлення “Hello, world!”. Текст такої програми на мові Assembler:

```
.MODEL small
.STACK 4096
.DATA
   Hellostr DB 'Hello, world!', 13, 10, '$'
.CODE
main PROC
   mov ax,@data
   mov ds,ax
   mov dx,offset Hellostr
   mov ah,9h
   int 21h
   mov ah, 04Ch
   int 21h
main ENDP
END main
```

На перший погляд може здатися, що дана програма довша, ніж аналогічна програма, написана на іншій мові. Програми, створені з використанням мови Assembler дійсно довші, оскільки кожна інструкція виконує менше дій, ніж інструкція на мові високого рівня. З іншого боку, мова Assembler дозволяє створювати максимально ефективні програми, які повністю контролюють ресурси обчислювальної системи.

У програмі є місце, де зберігаються дані і де зберігається код. І ці місця потрібно розділяти. Директива .DATA вказує на те що далі будуть йти дані а директива .CODE що тепер почнуться команди процесора. Це важливий поділ даних і команд процесора. Коли програма завантажується в пам'ять то операційній системі потрібно знати куди поставити вказівник для виконання команди. Саме директива .CODE і вказує при складанні, де це місце буде знаходитись. Те ж саме і для .DATA.

1.2 Переривання

В програмі ми використовували команду INT:

```
mov ah,9h
int 21h
mov ah, 04Ch
int 21h
```

Int - це скорочення слова Interrupt що з англійської на українську переводитися як переривання. Виглядає ця команда так:

```
Int номер
```

Тобто у кожного переривання є номер. З'явилося поняття переривання разом із створенням ЕОМ. Тоді стояло завдання про спільну роботу процесора і повільних зовнішніх пристроїв. Хорошим прикладом може служити клавіатура. Коли користувач натисне клавішу не відомо. Це може трапитися в будь-який момент, от коли він натискає клавішу – процесору повідомляється що потрібно обробити цю дію і отримати код клавіші, яка натиснута.

Можна зробити висновок, що переривання породжують зовнішні пристрої. Звичайно, крім отримання інформації від пристрою, цими пристроями потрібно ще й керувати. Пристрої повільні і крім іншого ще потрібно буде дочекатися закінчення виконання операції. Це теж реалізується за допомогою переривань тільки викликаються вони з програми. Отже, переривання бувають двох типів:

- програмні;
- апаратні.

Пристроїв існує багато - клавіатура, монітор, принтер, миша і так далі. Якщо не користуватися перериваннями, то операційна система повинна постійно опитувати пристрої, чи натиснута клавіша, чи потрібно вивести дані на монітор і так далі. Набагато простіше обумовити деякий механізм на який і буде звертатися увага операційної системи і процесора на необхідність проведення деяких дій.

Отже, програма щось виконує. У цей момент натискається клавіша. Програма повинна бути перервана. І як тільки це буде зроблено, управління буде передано спеціальним кодом (процедура обробки переривання), а потім програма буде виконуватися далі.

Те ж саме коли викликаються переривання для виведення символів на екран монітора (int 21h) то переривання самі генеруються. У цей момент може відбуватися зчитування з зовнішнього носія інформації або вивід інших символів на екран. Через те що переривання можуть наступати одночасно, існує пріоритет їх обробки. Є переривання, які будуть виконуватися в будь-якому випадку навіть якщо йде обробка іншого переривання.

Процедура обробки переривання це програма. Питання в тому тільки де вона зберігається. Базова обробка переривання зберігається в BIOS і в самих мікросхемах. Але використовувати їх досить важко. Для того, що б записати файл потрібно завести двигун дисковода, встановити головку в потрібному місці, дати команду перейти в той сектор прочитати таблицю файлів, перевірити чи там немає файлу і так далі. Всі ці завдання полегшує операційна система, яка надає Вам можливість використовувати переривання більш високо рівня. Використовуючи ці переривання, Ви можете одним етапом створити файл, наприклад. Розрізняють переривання за номерами:

21h - переривання DOS

13h - переривання BIOS

От коли викликається переривання (INT) вказується ще його номер (21h) тобто, призначається той пристрій, який буде виконувати цю дію.

Повернемось до коду програми "HelloWord":

```
mov ah, 9h
int 21h
mov ah, 04Ch
int 21h
```

Як видно одне і теж переривання викликаються два рази. У першому випадку виводиться рядок на екрані монітора, у другому закривається програма. Номер переривання один і той самий, так як же вдалося операційній системі розібратися що до чого? Справа в тому, що одне переривання може виконувати багато функцій. Наприклад виведення на екран і завершення програми. Просто викликати INT 21h мало, ще потрібно вказати що Ви конкретно хочете зробити. Ось це вказується в регістрі AX. В регістрі AX вказується що потрібно зробити, тобто яку дію. До речі цей регістр ділитися на дві частини (AX: AH + AL).

```
mov ah, 9h      ; функція виведення рядка
mov ah, 04Ch    ; функція завершення програми
```

А тепер все разом:

```
mov ah, 9h      ; будемо виводити рядок
int 21h         ; вивести
mov ah, 04Ch    ; будемо закривати програму
int 21h         ; закрити
```

Функцію потрібно вказувати для усіх переривань (10, 13, 21 і так далі).

1.3 Регістри

Регістри це спеціальні комірки пам'яті. Вся їх перевага у тому, що звернення до регістрів здійснюється значно швидше ніж до оперативної пам'яті ПК. Саме з цієї причини регістри використовуються для команд процесора. Звідти процесору зручно і швидко отримувати інформацію. Якщо говорити про ПК з типом процесорів 286 то розмір регістру 16 біт. Кожен регістр має ім'я і своє призначення. Вони бувають наступні за типами:

- регістри загального призначення: AX, BX, CX, DX, BP, SI, DI, SP
- сегментні регістри: CS, DS, SS, ES, GS, FS
- лічильник команд IP
- регістр прапорів FLAGS

В ПК з 32-розрядними процесорами є такі основні типи регістрів:

- регістри загального призначення: EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP
- сегментні регістри: CS, DS, SS, ES, GS, FS
- лічильник команд EIP
- регістр прапорів EFLAGS

В ПК з 64-розрядними процесорами є такі основні типи регістрів:

- регістри загального призначення: RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP
- сегментні регістри: CS, DS, SS, ES, GS, FS
- лічильник команд RIP
- регістр прапорів RFLAGS

Кожне ім'я регістра має певне значення:

AX (EAX, RAX) - accumulator акумулятор;

BX (EBX, RBX) - base база;

CX (ECX, RCX) - counter лічильник;

DX (EDX, RDX) - data дані;

BP (EBP, RBP) - base pointer вказівник бази;

SI (ESI, RSI) - source index індекс джерела;

DI (EDI, RDI) - destination index індекс приймача;

SP (ESP, RSP) - stack pointer вказівник стека;

CS - code segment сегмент команд;

DS - data segment сегмент даних;

SS - stack segment сегмент стеку;

ES - extra segment додатковий сегмент;

GS - extra segment додатковий сегмент;

FS - extra segment додатковий сегмент;

IP - instruction pointer лічильник команд.

Регістри AX, BX, CX і DX дозволяють нам звертатися не лише до всього регістра а також до старшого і молодшого байта:

AX: AH, AL

BX: BH, DL

DX: DH, DL

CX: CH, CL

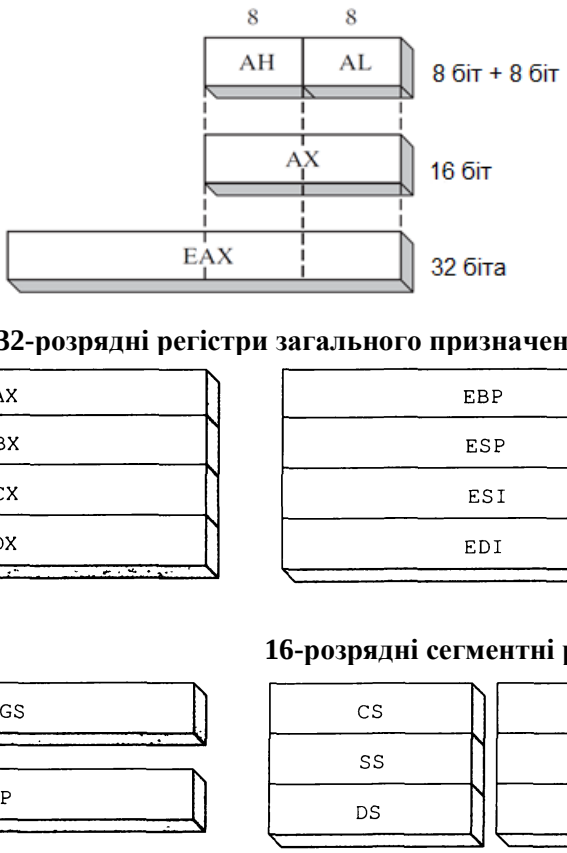


Рисунок 8 – Структура основних програмних регістрів процесора IA-32

В програмі «HelloWord» використовувався регістр AX для задання функції переривання:

mov ah, 9h

int 21h

mov ah, 04Ch

int 21h

При цьому використовувалась тільки частина регістра, а точніше старший байт:

- H (high) старший
- L (low) молодший

Чому не можна було використати, наприклад, комірки пам'яті? Тому що в асемблері існують правила про те, що і в якому регістрі повинно знаходитися. Правила, ці описані в документації. Ну, наприклад остання функція описана так:

Int 21H Функція 4CH

AH = 4CH

AL = код повернення

Повернення немає.

Припиняє процес і передає операційній системі код повернення.

Для більшості операцій використовуються регістри, а для переривань конкретно є специфікації, в яких написано що і в якому регістрі повинно знаходиться.

1.4 Команда MOV

В програмі кілька разів зустрічалася команда MOV:

```
mov dx, offset Hellostr
mov ah, 9h
```

Сенс команди MOV полягає в приміщенні з одного місця в інше:

```
MOV одержувач, передавач
```

Наприклад:

```
mov dx, offset Hellostr   ; помістити в dx зміщення рядка
mov ah, 9h                ; помістити в ah число 9h
```

Для вказівки сегменту даних використовується регістр DS. Тобто цей регістр повинен вказувати на початок даних в нашій програмі або на сегмент .DATA. В програмі було так:

```
mov ax, @data
mov ds, ax
```

По-перше, @data - це ідентифікатор сегменту даних .DATA при компіляції і збірці програми на місце цього слова буде поставлено реальне зміщення сегмента, в якому знаходяться дані. Адже до компонування програми ці інформація не відома. І реально це число стане відоме тільки при компонуванні програми.

Чому інформація поміщається спочатку в регістр AX? Існує правило - не можна безпосередньо змінювати вміст регістрів CS, DS, SS. Тобто ми не можемо написати так:

```
mov ds, 12345
```

Ми можемо змінити цей регістр тільки використовуючи інші регістри. Отже, все разом:

```
mov ax, @data    ; в регістр помістити AX зміщення для сегмента даних
mov ds, ax       ; встановити вміст регістра DS рівним AX тобто тепер там значення зміщення сегмента даних.
```

Отже, у регістр DS не можливо безпосередньо отримати доступ, а тільки використовуючи інші регістри. Цей регістр вказує на дані.

Отже, регістр DS вказує на сегмент даних, але ж даних може бути багато, наприклад, багато рядків, ось для цього і використовується адресація сегменту - зміщення. Взагалі подібна адресація була створена у зв'язку з необхідністю адресації до більшого діапазону ніж дозволяє число 16 біт. У нашому випадку нам потрібно ще й отримати вказівник на дані. І це було зроблено:

```
.DATASEG
Hellostr DB 'Hello, world!', 13, 10, '$'
.CODESEG
main PROC
```

.....

mov dx, offset Hellostr

Символи з кодами 13 і 10 позначають перехід на наступний рядок (символ 13 (0Dh) називається CR - Carriage Return - повернення каретки (курсора) на початок поточного рядка, а символ 10 (0Ah) LF - Line Feed - переведення каретки (курсора) на наступний рядок.

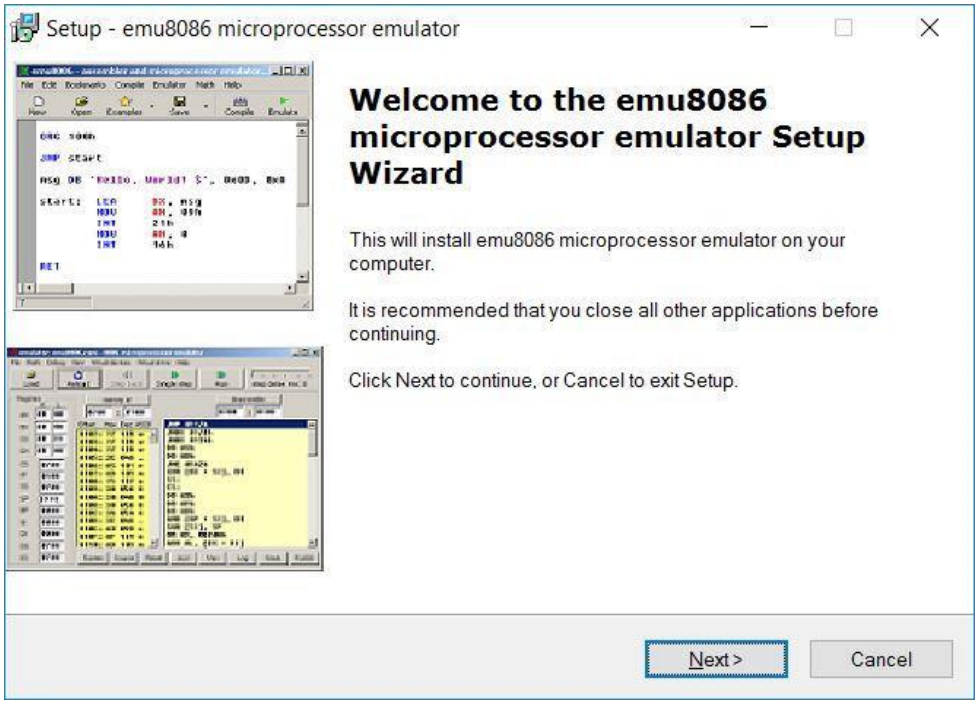
В цьому коді використовувалась директива offset – вона дає змогу обчислювати те як зміщені певні дані відносно початку сегмента даних. У приведеному прикладі це змінна *Hellostr*. При компонуванні програми в це місце буде вставлено конкретне число.

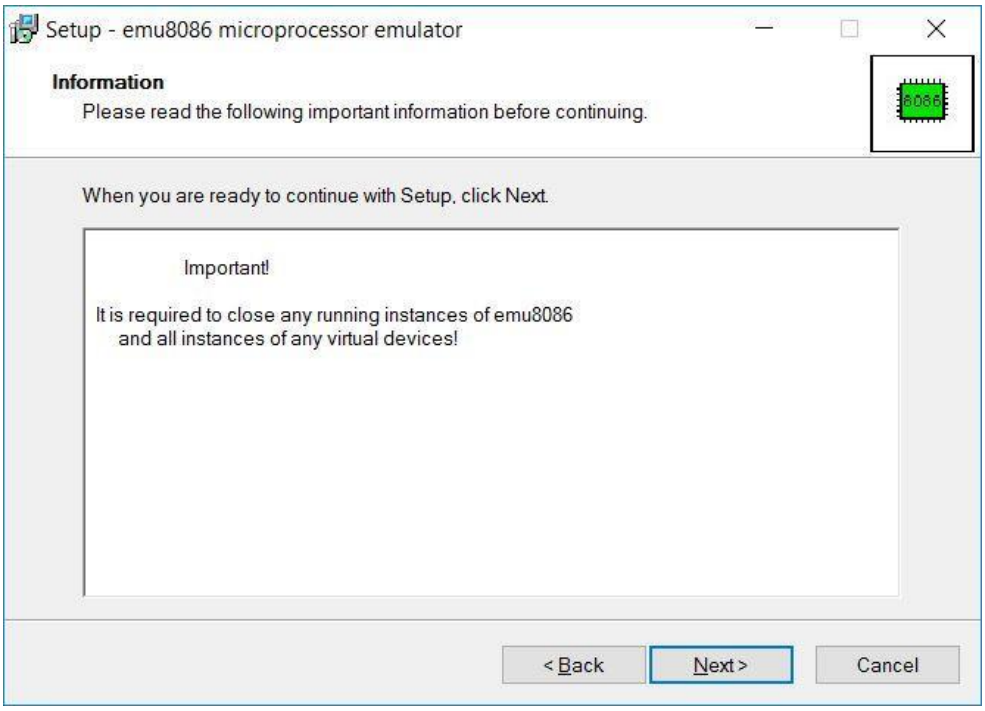
Інформація в регістрах DS:DX буде вказувати на місце розташування змінної *Hellostr* в сегменті даних. Після того як регістр DS ініціалізований, програма отримає першу літеру змінної *Hellostr*. Ось саме з'ясуванням цього числа і займається директива offset.

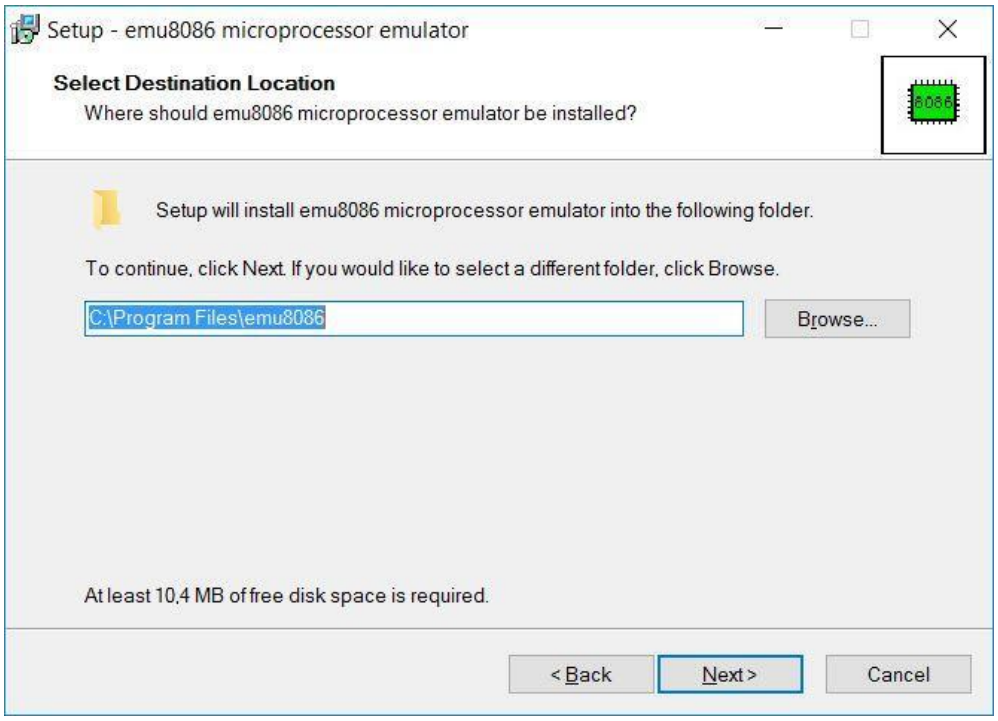
Отже, у даних є поняття зсув, яке вираховується щодо регістра DS. Для цього є директива offset:

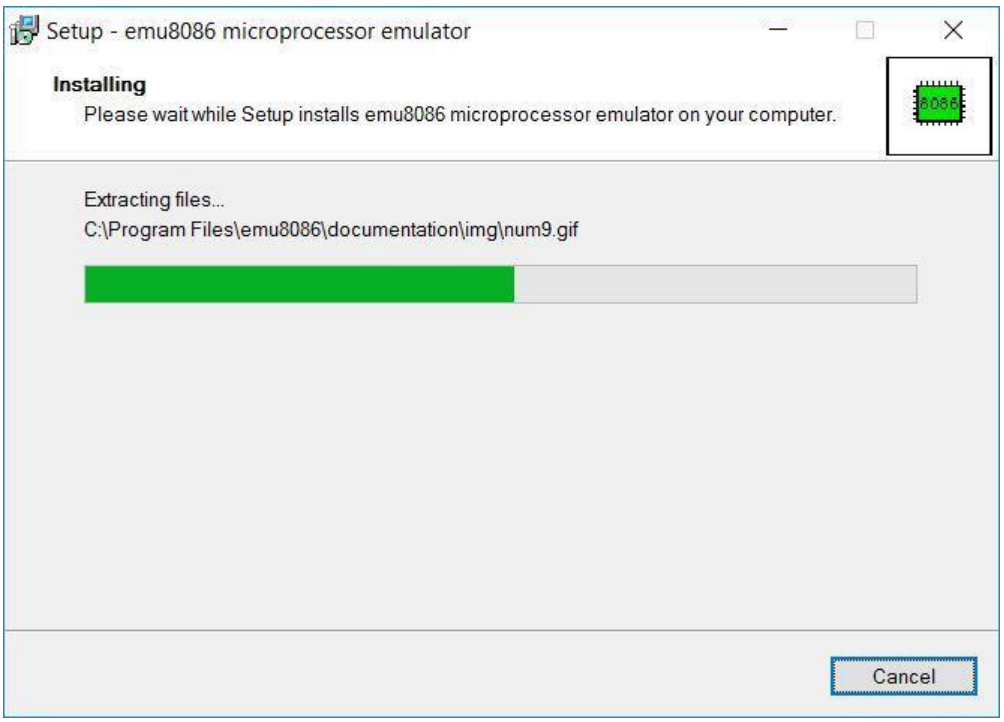
Offset ім'я_змінної

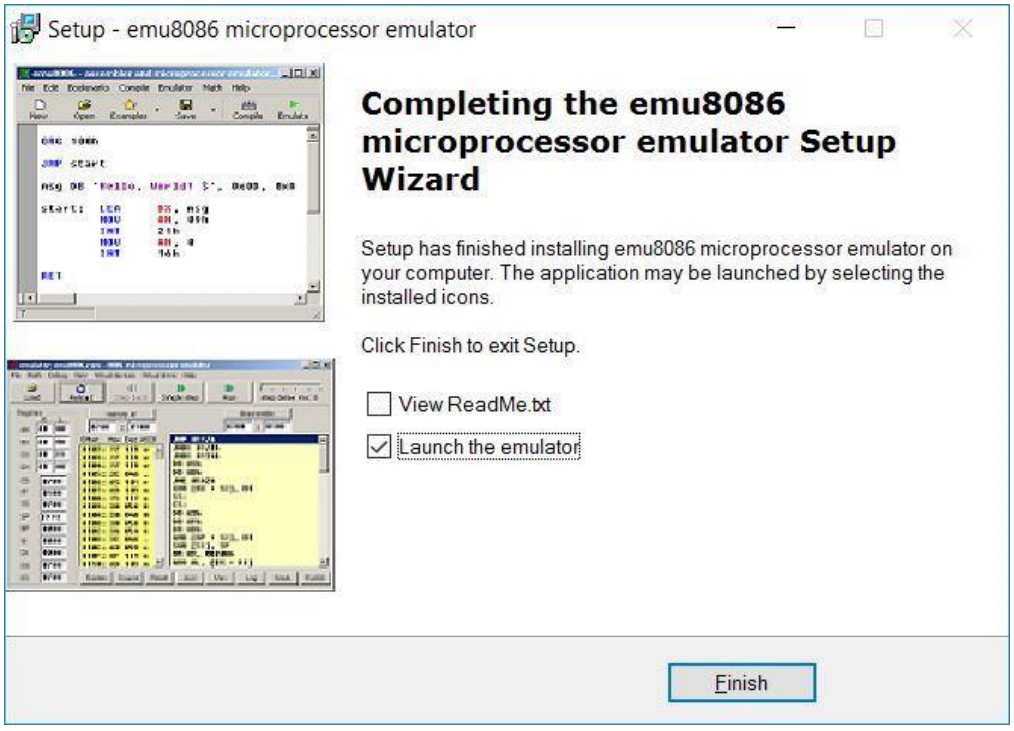
1.5 Процес встановлення програми emu8086




Setup - emu8086 microprocessor emulator
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
Important!
It is required to close any running instances of emu8086
and all instances of any virtual devices!
< Back
Next >
Cancel

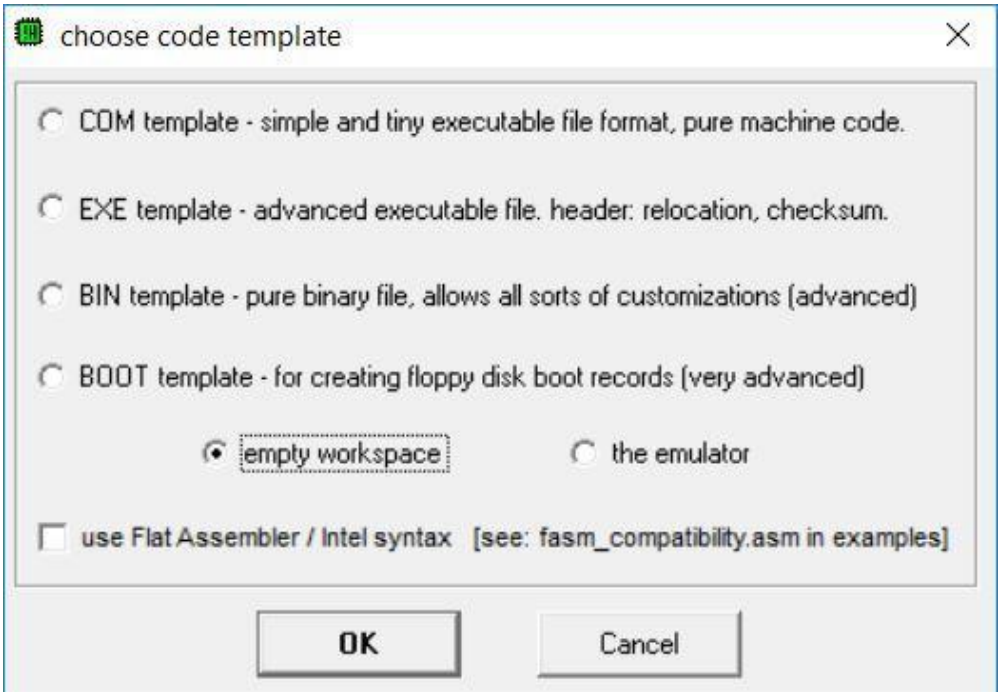

Setup - emu8086 microprocessor emulator
Select Destination Location
Where should emu8086 microprocessor emulator be installed?
Setup will install emu8086 microprocessor emulator into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\emu8086
Browse...
At least 10,4 MB of free disk space is required.
< Back
Next >
Cancel

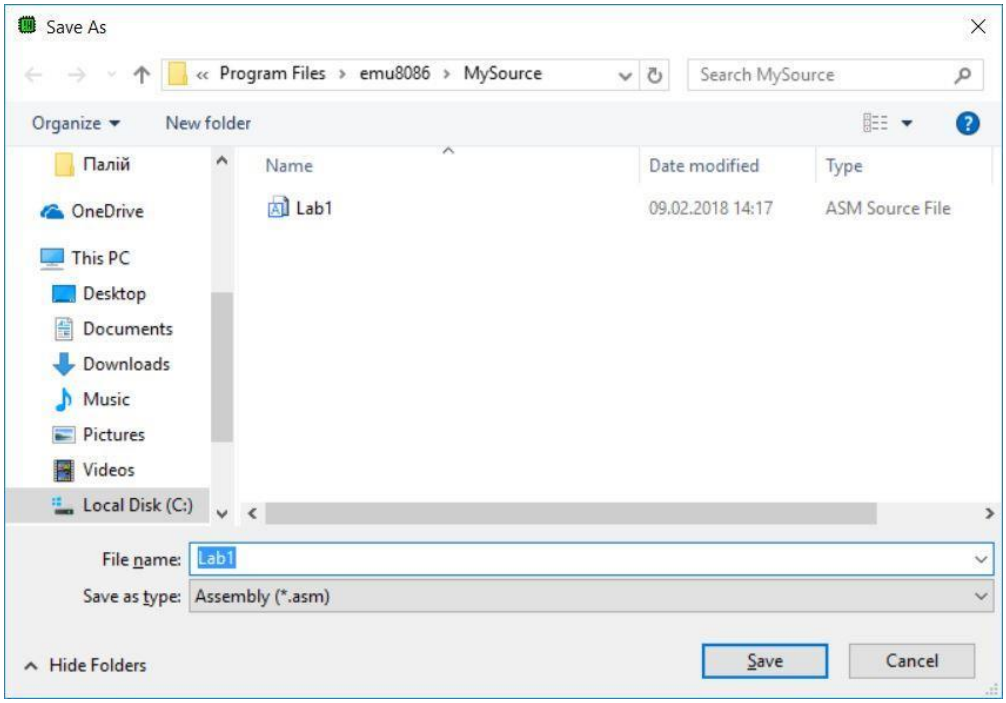


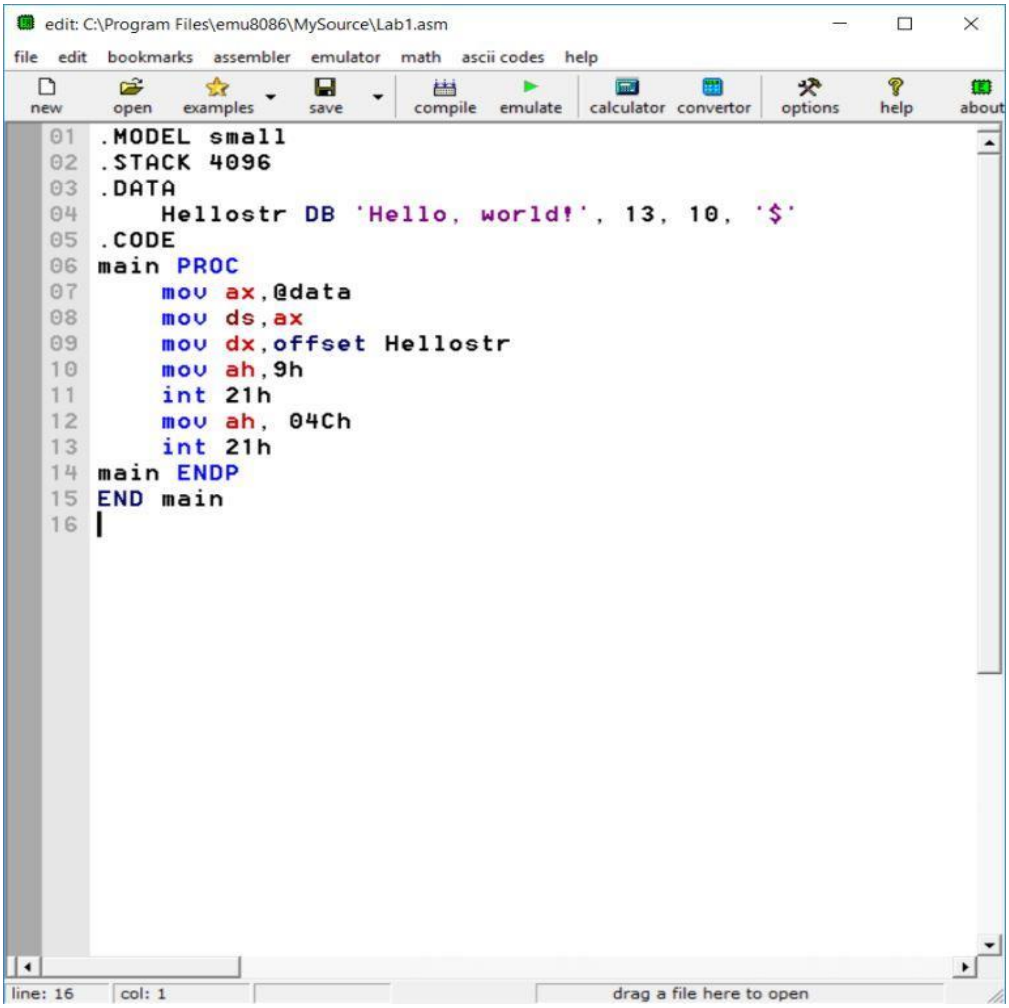


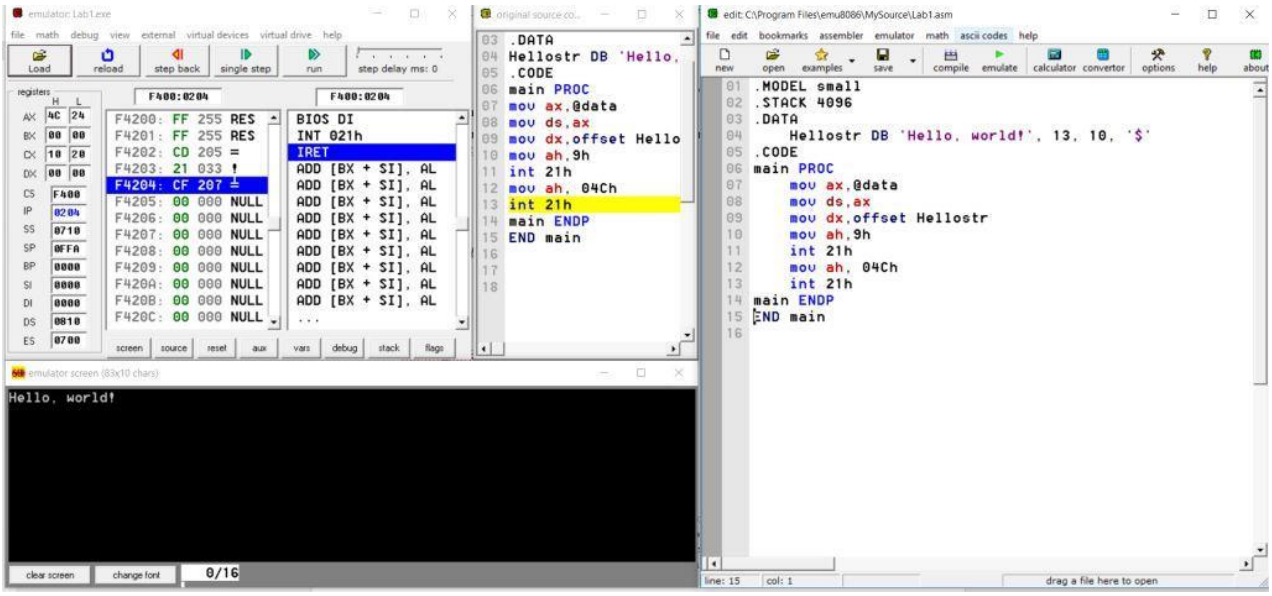
1.6 Процес створення проекту в програмі emu8086


welcome...
emu8086
microprocessor emulator with integrated assembler
version 4.08
new
code examples
quick start tutor
recent files
this product is registered to:
ic0n0statse
www.emu8086.com

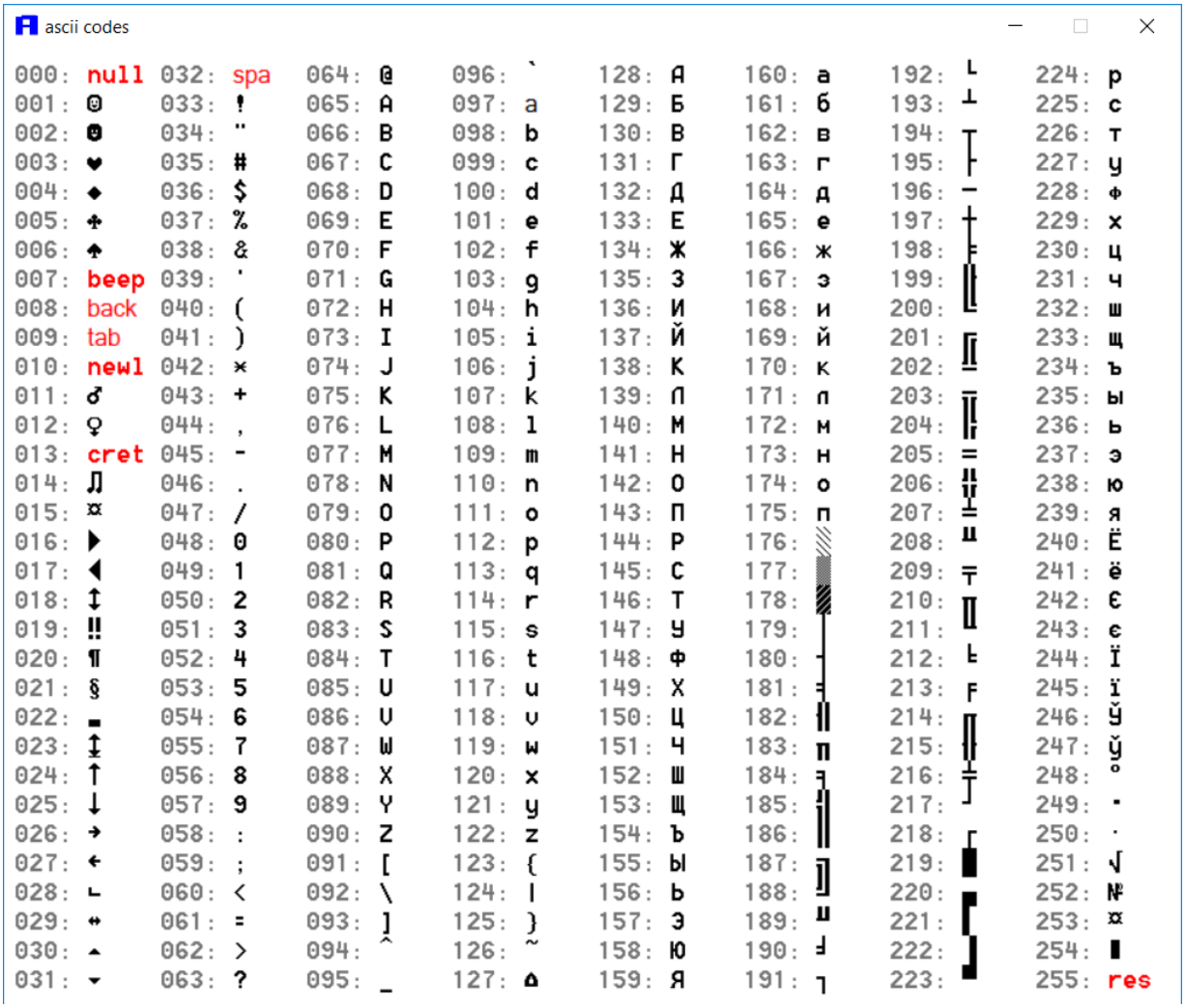

choose code template
COM template - simple and tiny executable file format, pure machine code.
EXE template - advanced executable file. header: relocation, checksum.
BIN template - pure binary file, allows all sorts of customizations (advanced)
BOOT template - for creating floppy disk boot records (very advanced)
empty workspace
the emulator
use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]
OK
Cancel


Save As
Program Files › emu8086 › MySource
Search MySource
Organize
New folder
Name
Date modified
Type
Lab1
09.02.2018 14:17
ASM Source File
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
File name: Lab1
Save as type: Assembly (*.asm)
Hide Folders
Save
Cancel


edit: C:\Program Files\emu8086\MySource\Lab1.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01 .MODEL small
02 .STACK 4096
03 .DATA
04 Hellostr DB 'Hello, world!', 13, 10, '$'
05 .CODE
06 main PROC
07 mov ax,@data
08 mov ds,ax
09 mov dx,offset Hellostr
10 mov ah,9h
11 int 21h
12 mov ah, 04Ch
13 int 21h
14 main ENDP
15 END main
16
line: 16
col: 1
drag a file here to open


Hello, world!
clear screen
change font
0/16
.MODEL small
.STACK 4096
.DATA
Hellostr DB 'Hello, world!', 13, 10, '$'
.CODE
main PROC
mov ax,@data
mov ds,ax
mov dx,offset Hellostr
mov ah,9h
int 21h
mov ah, 04Ch
int 21h
main ENDP
END main

1.7 Коди ASCII



2. Хід роботи

1. Вивчити структуру програми на мові програмування Асемблер та основні оператори, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. З метою кращого розуміння порядку створення та асемблювання програми, розробленої з використанням мови Assembler, написати та виконати програму яка відображає на екрані монітора прізвище, ім’я і по-батькові студента (латинськими літерами). Кожне слово повинне виводитись в новому рядку.
4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці латинськими літерами прізвище, ім’я і по-батькові студента.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Охарактеризуйте основні сегменти програм та їх призначення.
2. Наведіть особливості синтаксису мови Асемблер.
3. Який спосіб запису коментарів до програми?
4. Поясніть призначення та спосіб визначення міток в програмах на мові Асемблер.
5. Опишіть формат кодування команд на мові Асемблер.
6. Які відмінності між стандартними та спрощеними директивами керування сегментами?
7. Які типи переривань ви знаєте?
8. Як здійснюється визначення даних, необхідних для використання в програмі?
9. Перелічіть основні команди переміщення даних та обмеження на їх використання.

ЛАБОРАТОРНА РОБОТА №2

ТЕМА: Зчитування даних з клавіатури. Команди порівняння. Умовні та безумовні переходи.

МЕТА: Вивчення можливостей зчитування клавіатури, використання команд порівняння і умовних та безумовних переходів в мові програмування Асемблер.

1. Теоретичні відомості

1.1 Приклад програми для зчитування символів з клавіатури

```
.MODEL small
.STACK 4096
.DATA
   Question DB 'ARE YOU A STUDENT? - [Y/N] $'
   GreetingStudent DB  10, 13, 'HELLO STUDENT!$'
   GreetingTeacher DB  10, 13, 'HELLO TEACHER!$'
.CODE
main PROC
mov ax,@Data
mov ds,ax                    ; встановити регістр DS так, щоб він вказував на сегмент даних
mov dx,OFFSET Question       ; посилання на повідомлення-запитання
mov ah,09h                   ; функція DOS виводу стрічки
int 21h
mov ah,01h                   ; отримати дані вводу одного символа
int 21h
cmp al,'Y'              ; велика буква Y
jz IsStudent                 ; так, студент
cmp al,'y'              ; маленька буква y
jnz IsTeacher           ; ні, не студент
IsStudent:
mov dx,OFFSET GreetingStudent ; вказує на привітання "HELLO STUDENT"
jmp DisplayGreeting
IsTeacher:
mov dx,OFFSET GreetingTeacher ; вказує на привітання "HELLO TEACHER"
DisplayGreeting:
mov ah,09h                   ; функція DOS виводу повідомлення
int 21h                 ; вивести відповідне повідомлення
```

```
mov ah,04ch                    ; функція DOS завершення програми
int 21h                        ; завершити програму
main ENDP
END main
```

Таким чином в програму було додано два дуже важливих нових інструменти: можливість введення і прийняття рішень. Ця програма запитує в користувача «*ARE YOU A STUDENT?»*, зчитуючи відповідь (один символ) з клавіатури. Якщо такою відповіддю буде буква «Y» у верхньому або нижньому регістрі, що означає відповідь ТАК («Yes»), то програма виводить повідомлення «*HELLO STUDENT!*», в іншому випадку виводиться повідомлення «*HELLO TEACHER*!». У даній програмі є всі основні елементи корисної програми: введення інформації зовнішнього середовища, обробка даних і прийняття рішення.

1.2 Функція отримання символу (INT 21h 01H)

Для отримання інформації про код натиснутого на клавіатурі символу потрібно використати функцію переривання з кодом 01h:

```
AH = 01H
Виведення символу
AL = 8 бітний код числа
```

1.3 Регістр прапорів

Фізичний зміст регістра прапорів такий же, як і у інших регістрів (32 біта).

Eflags Flags

А от призначення - спеціальне. У цей регістр не можна просто записати значення командою MOV. У нього навіть немає імені. Він змінюється по-іншому. І він сильно відрізняється від першого спеціального регістра - EIP (вказівника поточної інструкції).

Регістри загального призначення програміст може сприймати як сховище:

- Числа (адреси або значення).
- Чисел (по тетрадам, байтам, словам тощо).
- Бітової інформації.

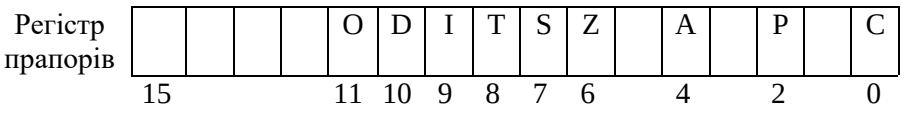
Регістр EIP (або IP) можна вважати лише як сховище числа, і воно завжди означає адресу.

А ось регістр прапорів – тільки як сховище бітової інформації!

Зрозуміло, що з точки зору комп'ютера будь-яка інформація бітова. Однак з точки зору програміста є числа, які можна ділити, множити і проводити з ними складні обчислення. У цих числах важливіше оперувати цілим байтом, а не кожним окремим бітом.

Існують такі байти, кожен біт яких має самостійне значення. Стан процесора, результат порівняння і багато іншого. Звичайно, ці байти теж можуть виглядати як число в шіснадцятковій системі числення, але воно, навпаки, в цілому вигляді не має явного сенсу.

Регістр прапорів - це якраз 16 біт, які розглядаються окремо один від одного.



Регістр прапорів містить дев'ять найбільш використовуваних прапорів:

CF - прапор перенесення;
PF - прапор парності;
AF - прапор додаткового перенесення;
ZF - прапор нуля;
SF - прапор знаку;
TF - прапор трасування (пастки);
IF - прапор дозволу переривання;
DF - прапор напрямку;
OF - прапор переповнення.

Прапор перенесення (Carry flag, або **CF**) встановлюється у випадку, якщо при виконанні беззнакової арифметичної операції результатом є число, розрядність якого перевищує розрядність виділеного для нього поля результату. Існують команди безпосереднього встановлення (STC) і скидання (CLC) прапора переносу:

1 - арифметична операція здійснила перенесення з старшого біта результату, старшим є 7-й, 15-й або 31-й біт в залежності від розмірності операнда;

0 - перенесення не було.

Прапор парності (Parity flag, або **PF**) встановлюється у випадку, якщо в результаті виконання арифметичної або логічної операції результатом є число, що містить парну кількість одиничних бітів у 8-розрядному числі або в молодшому байті 16-розрядного числа. Цей прапор корисний при тестуванні пам'яті і при контролі правильності передачі даних:

1 - 8 молодших розрядів (цей прапор застосовується тільки для 8 молодших розрядів операнда будь-якого розміру) результату містять парне число одиниць;

0 - 8 молодших розрядів результату містять непарне число одиниць.

Прапор додаткового перенесення (Auxiliary Carry, або **AF**) встановлюється, якщо при виконанні арифметичної операції з 8-розрядним операндом відбувається перенесення з третього біта в четвертий. Застосовується тільки для команд, що працюють з BCD-числами. Фіксує факт позичання з молодшої тетради результату.

Прапор нуля (Zero flag, або **ZF**) встановлюється, якщо при виконанні арифметичної або логічної операції результатом є число, рівне нулю (тобто всі біти результату рівні 0):

1 - результат нульовий;

0 - результат ненульовий.

Прапор знаку (Sign flag, або **SF**) встановлюється, якщо при виконанні арифметичної або логічної операції результатом є негативне число (тобто старший біт результату рівний 1). У стандартному додатковому коді одиниця в старшому розряді результату означає отримання негативного числа. Відображає стан старшого біта результату (біти 7, 15 або 31 для 8-, 16- або 32-розрядних операндів відповідно):

1 - старший біт результату дорівнює 1;

0 - старший біт результату дорівнює 0.

Прапор трасування (пастки) (trace flag, або **TF**) призначений для організації покрокової роботи процесора:

1 - процесор генерує переривання з номером 1 після виконання кожної машинної команди (може використовуватися при відладці програм);

0 - звичайна робота.

Прапор дозволу переривання (interrupt enable flag, або **IF**) призначений для дозволу або заборони (маскування) апаратних переривань (зовнішніх переривань по входу INTR). Існують команди безпосереднього встановлення (STI) і скидання (CLI) прапора переривання:

1 - апаратні переривання дозволені;

0 - апаратні переривання заборонені.

Прапор напрямку (direction flag, або **DF**) використовується рядковими командами. Значення прапора DF визначає напрямок поелементної обробки в цих операціях: від початку рядка до кінця (DF = 0) або, навпаки, від кінця рядка до його початку (DF = 1). Для роботи з прапором DF існують спеціальні команди CLD (зняти прапор DF) і CTD (встановити прапор DF). Застосування цих команд дозволяє привести прапор DF у відповідність з алгоритмом і забезпечити автоматичне збільшення або зменшення лічильників при виконанні операцій з рядками.

Прапор переповнення (Overflow flag, або **OF**) встановлюється у випадку, якщо при виконанні арифметичної операції із знаком результатом є число, розрядність якого перевищує розрядність виділеного для нього поля результату. Використовується для фіксації факту втрати значущого біта при арифметичних операціях:

1 - в результаті операції відбувається перенесення в старший знаковий біт результату або позичання з старшого знакового біта результату (біти 7,15 або 31 для 8-, 16- або 32-розрядних операндів відповідно);

0 - в результаті операції не відбувається перенесення в старший знаковий біт результату або позичання з старшого знакового біта результату.

1.4 Команда CMP

Команда CMP походить від англ. слова compare – порівнювати.

Формат: *CMP (операнд призначення), (операнд-джерело)*

Команда CMP (порівняння) віднімає операнд-джерело з операнда призначення, не змінюючи при цьому значення операндів. Операнди можуть бути байтовими або двобайтовими числами. Хоча значення операндів не змінюються, значення прапорів оновлюються, що може бути враховано в наступних командах умовного переходу. Команда CMP впливає на прапори AF, CF, OF, PF, SF і ZF. При збігу значень операндів встановлюється прапор ZF. Прапор перенесення встановлюється, якщо операнд призначення менший ніж операнд-джерело.

В приведеному прикладі важливий лише прапор ZF.

Наприклад, щодо прапора нуля (ZF) дія команди виглядає так: MOV AH, 10h; для наочності присвоїмо AH значення 10h

```
CMP AH, 8       ; ZF = 0
CMP AH, 10h     ; ZF = 1
CMP AH, 0A0h    ; ZF = 0
```

1.5 Команда умовного переходу JNZ (JNE)

Команда умовного переходу JNZ (JNE) походить від англ. слів Jmp if Not Zero - стрибнути якщо Zero Flag (ZF) не встановлений, тобто не рівний одиниці (тобто, тоді коли ZF = 0).

Формат:

JNZ мітка

Дія Якщо ZF = 0, то дія схожа на команду JMP (зміна EIP на вказану адресу, тобто перехід куди сказали).

Якщо ZF = 1, то управління переходить до наступної команди (зміна EIP на адресу наступної команди, тобто нічого не відбувається).

JZ (JE) - команда протилежної дії.

1.6 Умовні та безумовні переходи

Безумовний перехід - це перехід, який виконується завжди. Безумовний перехід здійснюється за допомогою команди **JMP**. У цієї команди є один операнд, який може бути безпосередньою адресою (міткою), регістром або осередком пам'яті, яка містить адресу. Існують також «далекі» переходи - між сегментами.

Мітка – це набір з символів латинського алфавіту, цифр та/або символу нижнього підкреслення («_») який починається з символу і закінчується двокрапкою («:»).

```
JMP     mitka       ; Перехід на мітку
JMP     bx          ; Перехід за адресою в BX
JMP     word [bx]   ; Перехід за адресою, що міститься в пам'яті за адресою в BX
```

Умовний перехід здійснюється, якщо виконується певна умова, задана прапорами процесора (крім однієї команди, яка перевіряє CX на рівність нулю). Стан прапорів змінюється після виконання арифметичних, логічних і деяких інших команд. Якщо умова не виконується, то управління переходить до наступної команди.

Існує багато команд для різних умовних переходів. Також для деяких команд є синоніми (наприклад, JZ і JE - це одне й те саме). Для наочності всі команди умовних переходів наведені в таблиці:

Типи операндів	Команда	Критерій умовного переходу	Значення прапорів для переходу
Будь-які	JZ / JE	операнд_1 = операнд_2	ZF = 1
Будь-які	JNZ / JNE	операнд_1 ≠ операнд_2	ZF = 0
Числа зі знаком	JL / JNGE	операнд_1 < операнд_2	SF ≠ OF
Числа зі знаком	JLE / JNG	операнд_1 <= операнд_2	SF ≠ OF або ZF = 1
Числа зі знаком	JG / JNLE	операнд_1 > операнд_2	SF = OF і ZF = 0
Числа зі знаком	JGE / JNL	операнд_1 => операнд_2	SF = OF
Числа без знаку	JB / JNAE / JC	операнд_1 < операнд_2	CF = 1
Числа без знаку	JBE / JNA	операнд_1 <= операнд_2	CF = 1 або ZF = 1
Числа без знаку	JA / JNBE	операнд_1 > операнд_2	CF = 0 і ZF = 0
Числа без знаку	JAE / JNB / JNC	операнд_1 => операнд_2	CF = 0
Будь-які	JP	число одиничних бітів парне	PF = 1
Будь-які	JNP	число одиничних бітів непарне	PF = 0
Числа зі знаком	JS	знак дорівнює 1	SF = 1
Числа зі знаком	JNS	знак дорівнює 0	SF = 0

Числа без знаку	JO	є переповнення	OF = 1
Числа без знаку	JNO	немає переповнення	OF = 0
Будь-які	JCXZ	вміст CX дорівнює нулю	CX = 0

Значення абревіатур в назві команди jxx:

Позначення	**Оригінальний термін**	**Переклад**	**Тип операндів**
e	Equal	Рівно	Будь-які
n	Not	Ні	Будь-які
g	Greater	Більше	Числа зі знаком
l	Less	Менше	Числа зі знаком
a	Above	Вище (в значенні більше)	Числа без знаку
b	Below	Нижче (в значенні менше)	Числа без знаку

У всіх цих команд один операнд - ім'я мітки для переходу. Зверніть увагу, що деякі команди застосовуються для беззнакових чисел, а інші - для чисел із знаком. Порівняння «вище» і «нижче» відносяться до беззнакових чисел, а «більше» і «менше» - до чисел із знаком. Для беззнакових чисел ознакою переповнення буде прапор CF, а відповідними командами переходу JC і JNC. Для чисел із знаком про переповнення можна судити за станом прапора OF, тому їм відповідають команди переходу JO і JNO. Команди переходів не змінюють значення прапорів.

2. Хід роботи

1. Вивчити основні поняття та оператори, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму яка виконує завдання відповідно до варіанту. В програмі використовувати символи латинського алфавіту. При зчитуванні з клавіатури використовувати як маленькі так і великі символи.

№ варіанту	**Завдання**
1	Програма пропонує вам ввести символи ‘N’ або ‘S’. При введенні символу ‘N’ – виводить на екран ваше ім’я. При введені символу ‘S’ – виводить ваше прізвище. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
2	Програма пропонує вам ввести порядковий номер дня тижня. Після його введення виводить на екран повну назву відповідного дня і завершує програму. У випадку введення символу, який не відповідає жодному з днів тижня – виводить повідомлення про помилку і пропонує ввести порядковий номер ще раз.

3	Програма пропонує вам ввести символи ‘N’ або ‘F’. При введенні символу ‘N’ – виводить на екран ваше ім’я. При введені символу ‘F’ – виводить по-батькові. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
4	Програма пропонує вам ввести порядковий номер календарного місяця (від 1 до 9). Після його введення виводить на екран повну назву відповідного місяця і завершує програму. У випадку введення символу, який не відповідає жодному з місяців – виводить повідомлення про помилку і пропонує ввести порядковий номер ще раз.
5	Програма запитує вас чи ви чоловічої статі чи жіночої. При введенні символу ‘M’ - виводить повідомлення ‘MAN’ і ваше ім’я, при введені символу ‘W’ – виводить повідомлення ‘WOMAN’ і ваше ім’я. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
6	Програма запитує вас який зараз час доби. При введенні символу ‘M’ - виводить привітання ‘GOOD MORNING’ і ваше ім’я, при введені символу ‘D’ – виводить повідомлення ‘GOOD DAY’ і ваше ім’я, при введенні символу ‘E’ - виводить привітання ‘GOOD EVENING’ і ваше ім’я, при введені символу ‘N’ – виводить повідомлення ‘GOOD NIGHT’ і ваше ім’я. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
7	Програма пропонує вам ввести символи ‘N’, ‘S’ або ‘F’. При введенні символу ‘N’ – виводить на екран ваше ім’я. При введені символу ‘S’ – виводить ваше прізвище. При введені символу ‘F’ – виводить по-батькові. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
8	Програма запитує вас чи ви чоловічої статі чи жіночої. При введенні символу ‘M’ - виводить повідомлення ‘MAN’ і ваше по-батькові, при введені символу ‘W’ – виводить повідомлення ‘WOMAN’ і ваше по-батькові. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
9	Програма пропонує вам ввести символи ‘I’ або ‘P’. При введенні символу ‘I’ – виводить на екран ваше ім’я. При введені символу ‘P’ – виводить ваше прізвище. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
10	Програма пропонує вам ввести символи ‘N’ або ‘F’. При введенні символу ‘N’ – виводить на екран ваше ім’я. При введені символу ‘F’ – виводить по-батькові. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
11	Програма запитує вас який зараз час доби. При введенні символу ‘M’ - виводить привітання ‘GOOD MORNING’ і ваше ім’я, при введені символу ‘D’ – виводить повідомлення ‘GOOD DAY’ і ваше ім’я, при введенні символу ‘E’ - виводить привітання ‘GOOD EVENING’ і ваше ім’я, при введені символу ‘N’ – виводить повідомлення ‘GOOD NIGHT’ і ваше ім’я. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого

	символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
12	Програма пропонує вам ввести символи ‘I’ або ‘P’. При введенні символу ‘I’ – виводить на екран ваше ім’я. При введені символу ‘P’ – виводить ваше прізвище. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
13	Програма запитує вас чи ви чоловічої статі чи жіночої. При введенні символу ‘M’ - виводить повідомлення ‘MAN’ і ваше прізвище, при введені символу ‘W’ – виводить повідомлення ‘WOMAN’ і ваше прізвище. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
14	Програма запитує вас який зараз час доби. При введенні символу ‘M’ - виводить привітання ‘GOOD MORNING’ і ваше прізвище, при введені символу ‘D’ – виводить повідомлення ‘GOOD DAY’ і ваше прізвище, при введенні символу ‘E’ - виводить привітання ‘GOOD EVENING’ і ваше прізвище, при введені символу ‘N’ – виводить повідомлення ‘GOOD NIGHT’ і ваше прізвище. Після цього програма завершується. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
15	Програма пропонує вам ввести символи ‘I’ або ‘P’. При введенні символу ‘I’ – виводить на екран ваше ім’я. При введені символу ‘P’ – виводить ваше прізвище. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
16	Програма пропонує вам ввести порядковий номер дня тижня. Після його введення виводить на екран повну назву відповідного дня. Після цього програма знову пропонує ввести порядковий номер дня тижня. У випадку введення символу, який не відповідає жодному з днів тижня – виводить повідомлення про помилку і пропонує ввести порядковий номер ще раз.
17	Програма пропонує вам ввести символи ‘N’ або ‘S’. При введенні символу ‘N’ – виводить на екран ваше ім’я. При введені символу ‘S’ – виводить ваше прізвище. Після цього програма знову пропонує ввести один з цих символів. У випадку введення іншого символу – виводить повідомлення про помилку і пропонує ввести символ ще раз.
18	Програма пропонує вам ввести порядковий номер календарного місяця (від 1 до 9). Після його введення виводить на екран повну назву відповідного місяця. Після цього програма знову пропонує ввести порядковий номер календарного місяця. У випадку введення символу, який не відповідає жодному з місяців – виводить повідомлення про помилку і пропонує ввести порядковий номер ще раз.

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.

7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці повідомлення відповідно до завдання поданого в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Які ви знаєте команди умовного переходу в мові програмування Асемблер?
2. Що таке безумовний перехід в мові програмування Асемблер?
3. Що таке мітка в мові програмування Асемблер?
4. Як можна зчитувати символ з клавіатури?
5. Як виводити рядок на екран монітора в мові програмування Асемблер?
6. Наведіть приклад використання команди порівняння.

ЛАБОРАТОРНА РОБОТА №3

ТЕМА: Арифметичні операції в мові програмування Assembler.

МЕТА: Вивчити основні команди мови програмування Асемблер для здійснення арифметичних операцій та навчитись використовувати їх в процесі написання програм.

1. Теоретичні відомості

1.1 Приклад програми

```
.MODEL small
.STACK 4096
.DATA
   Set_X  DB      'X = $'
   Result  DB      13,10,'Y = $'
   error_  db "incorrect number$"
   buff    db 6,7 Dup(?)
.CODE
main PROC
   mov     ax,@Data
   mov     ds,ax              ; встановити регістр DS таким чином, щоб він вказував на сегмент даних
   mov     dx,OFFSET Set_X            ; посилання на повідомлення-запитання
   mov     ah,09h                     ; функція DOS виводу стрічки
   int     21h
;*************************************************************************
;***** Зчитування числа з клавіатури і перетворення його в цифрове значення *****
;********                              Результат в регістрі AX                 ********
;*************************************************************************
   mov    cx,0
   mov     ah,0ah
   xor     di,di
   mov     dx,offset buff             ; адреса буфера
   int     21h                        ; зчитуємо стрічку
   mov     dl,0ah
   mov     ah,02
   int     21h                        ; виводимо перевід стрічки
```

```
 ; обробляємо вміст буфера
   mov     si,offset buff+2                ; берем аддрес початку стрічки
   cmp     byte ptr [si],"-"               ; якщо перший символ мінус
   jnz     ii1
   mov     di,1                    ; встановлюємо прапор
   inc     si                      ; і пропускаєм його
ii1:
   xor     ax,ax
   mov     bx,10
ii2:
   mov     cl,[si]                 ; берем символ з буфера
   cmp     cl,0dh                  ; перевіряємо чи він не останній
   jz      endin
; якщо символ не останній, то перевіряємо його на правильність
   cmp     cl,'0'                  ; якщо введений невірний символ <0
   jl      er
   cmp     cl,'9'                  ; якщо введений невірний символ >9
   ja      er
   sub     cl,'0'                  ; робимо з символу число
   mul     bx                      ; перемножуємо на 10
   add     ax,cx                   ; додаємо до інших
   inc     si                      ; вказівник на наступний символ
   jmp     ii2                     ; повторюємо
er:  ; якщо була помилка, то виводимо повідомлення про це і виходимо
   mov     dx, offset error_
   mov     ah,09
   int     21h
   int     20h
    ; все символи з буфера оброблені число знаходиться в ax
endin:
   cmp     di,1                    ; якщо встановлений прапорець, то
   jnz     ii3
   neg     ax                      ; робим число від'ємним
ii3:
;*************************************************************************
```

```
;************ Завершення функції зчитування і розпізнавання числа ***************
;*********************************************************************************
    add     ax,1
    mov     bx,2
    mul     bx
    xchg    cx,ax

    mov     dx,OFFSET Result       ; вказує на
    mov     ah,09h          ; функція DOS виводу повідомлення
    int     21h             ; вивести відповідне повідомлення
    xchg    cx,ax
;*********************************************************************************
;******** Початок функції перетворення числа в стрічку і виводу її на екран ********
;*********************************************************************************
; Перевіряємо число на знак
   test     ax, ax
   jns      oi1
; Якщо воно від'ємне, виведемо мінус і залишимо його модуль.
   mov      cx, ax
   mov      ah, 02h
   mov      dl, '-'
   int      21h
   mov      ax, cx
   neg      ax
; Кількість цифр будемо тримати в CX.
oi1:
    xor     cx, cx
    mov     bx, 10              ; основа сс. 10 для десяткової і т.п.
oi2:
    xor     dx,dx
    div     bx
; Ділимо число на основу сс. В залишку виходить остання цифра.
; Відразу виводити її неможна, тому збережемо її в стек.
    push    dx
    inc     cx
```

```
; А з остачею повторюємо те ж саме, відділяючи від нього чергову
; цифру справа, поки не залишиться нуль, що означає, що далі
; зліва лише нулі.
   test     ax, ax
   jnz      oi2
; Тепер приступимо до виводу.
   mov      ah, 02h
oi3:
   pop       dx
; Беремо чергову цифру, переводимо її в символ и виводимо.
   add      dl, '0'
   int      21h
; Повторимо рівно стільки разів, скільки цифр нарахували.
   loop   oi3
;*************************************************************************
;******* Завершення функції перетворення числа в стрічку і виводу її на екран ******
;*************************************************************************
   mov      ah,04ch        ; функція DOS завершення програми
   int      21h            ; завершити програму
main ENDP
END main
```

Дана програма зчитує число X з клавіатури, виконує обчислення за формулою: *Y = (X+1)*2*. Після цього виводить результат обчислення – число Y на екран.

Таблиця 1 – Команди арифметичних операцій мови Асемблер

Додавання	
ADD	Додавання байта або слова
ADC	Додавання байта або слова з розрядом переносу
INC	Збільшення байта або слова на 1
AAA	Корекція додавання неупакованих десяткових чисел
DAA	Корекція додавання упакованих десяткових чисел
Віднімання	
SUB	Віднімання байта або слова
SBB	Віднімання байта або слова з розрядом переносу
DEC	Зменшення байта або слова на 1
NEG	Інверсія байта або слова
CMP	Порівняння байта або слова
AAS	Корекція віднімання неупакованих десяткових чисел
DAS	Корекція віднімання упакованих десяткових чисел
Множення	

MUL	Множення беззнакового байта або слова
IMUL	Цілочисельне множення байта або слова
AAM	Корекція множення неупакованих десяткових чисел
Ділення	
DIV	Ділення беззнакового байта або слова
IDIV	Цілочисельне ділення байта або слова
AAD	Корекція ділення неупакованих десяткових чисел
CWB	Перетворення байта в слово
CWD	Перетворення слова в подвійне слово

1.2 Команди додавання

ADD (операнд призначення), (операнд-джерело)

Сума двох операндів, які можуть бути байтами або словами, поміщається в операнд призначення. Обидва операнда можуть бути знаковими або беззнаковими числами. Команда ADD змінює значення прапорів AF, CF, OF, PF, SF і ZF.

ADC (операнд призначення), (операнд-джерело)

Команда ADC (додавання з врахуванням розряду переносу) додає операнди, які можуть бути байтами або словами, і додає 1, якщо встановлено розряд переносу, результат поміщається в операнд призначення. Обидва операнда можуть бути знаковими або беззнаковими числами. Команда ADD змінює значення прапорів AF, CF, OF, PF, SF і ZF. Команда ADC враховує значення розряду переносу від попередньої операції, це може бути використано для організації додавання чисел довільної розрядності.

INC (операнд призначення)

Команда INC (інкремент) додає одиницю до операнду призначення. Операнд може бути байтом або словом і трактується як беззнакове двійкове число. Команда INC змінює значення прапорів AF, OF, PF, SF і ZF значення прапора CF ця команда не змінює.

AAA

Команда AAA (корекція додавання неупакованих десяткових чисел) приводить вміст регістра AL до виду правильного неупакованого десяткового числа, старший напівбайт при цьому обнуляється. Команда AAA змінює значення прапорів FC і AC вміст прапорів OF, PF, SF і ZF після виконання команди AAA невизначено.

DAA

Команда DAA (десяткова корекція додавання) приводить вміст регістра AL до виду правильного упакованого десяткового числа після попередньої команди додавання. Команда DAA змінює значення прапорів AF, CF, PF, SF і ZF вміст прапора OF після виконання команди DAA не визначено.

Для додавання двох чисел призначена команда ADD. Вона працює як з числами зі знаком, так і з числами без знаку (це особливість додаткового коду).

Операнди повинні мати однаковий розмір (не можна додавати 16- і 8-бітове значення). Результат поміщається на місце першого операнда. Загалом, ці правила справедливі для більшості команд.

Після виконання команди змінюються прапори, за якими можна визначити характеристики результату:

Прапор CF встановлюється, якщо при додаванні відбулося перенесення із старшого розряду. Для беззнакових чисел це означатиме, що відбулося переповнення і результат вийшов некоректним.

Прапор OF позначає переповнення для чисел із знаком.

Прапор SF дорівнює знаковому біту результату (для чисел зі знаком, а для беззнакових він дорівнює старшому біту і особливо сенсу не має).

Прапор ZF встановлюється, якщо результат дорівнює 0.

Прапор PF - ознака парності, дорівнює 1, якщо результат містить непарне число одиниць.

Приклади:

```
add ax, 5      ; AX = AX + 5
add dx, cx     ; DX = DX + CX
add dx, cl     ; Помилка: різний розмір операндів.
```

1.3 Команди віднімання

SUB (операнд призначення), (операнд-джерело)

Вміст операнда-джерела віднімається з вмісту операнда призначення, і результат поміщається в операнд призначення. Операнди можуть бути знаковими або беззнаковими, двійковими або десятковими (див. команди AAS і DAS), однобайтовими або двобайтовими числами. Команда SUB змінює значення прапорів AF, CF, OF, PF, SF і ZF.

Насправді віднімання в процесорі реалізовано за допомогою додавання. Процесор змінює знак другого операнда на протилежний, а потім додає два числа. Якщо вам необхідно в програмі поміняти знак числа на протилежний, можна використовувати команду NEG. У цієї команди всього один операнд.

Приклад:

```
sub si, dx      ; SI = SI - DX
neg ax                  ; AX = -AX
```

SBB (операнд призначення), (операнд-джерело)

Команда SBB (віднімання з урахуванням позичання) віднімає вміст операнда-джерела від вмісту операнда призначення, потім віднімає від результату 1, якщо був встановлений прапор перенесення CF. Результат поміщається на місце операнда призначення.

Операнди можуть бути знаковими або беззнаковими, двійковими або десятковими (див. команди AAS і DAS), однобайтовими або двобайтовими числами. Команда SBB змінює значення прапорів AF, CF, OF, PF, SF і ZF. Команда SBB може бути використана для здійснення віднімання мультибайтних чисел.

DEC (операнд призначення)

Команда DEC (декремент) віднімає одиницю з операнда призначення, який може бути одно- або двобайтовим. Команда DEC змінює вміст прапорів AF, OF, PF, SF і ZF. Вміст прапора CF при цьому не змінюється.

NEG (операнд призначення)

Команда NEG (інверсія) віднімає операнд призначення, який може бути байтом або словом і поміщає результат в операнд призначення. Така форма двійкового доповнення числа придатна для інверсії знакових цілих чисел. Якщо операнд нульовий, його знак не змінюється. Спроба застосувати команду NEG до байтового числа - 128 або до двобайтового числа - 32768 не призводить до зміни значення операнда, але встановлює прапор OF.

Команда NEG впливає на прапори AF, CF, OF, PF, SF і ZF. Прапор CF завжди встановлений за винятком випадку, коли операнд дорівнює нулю, коли цей прапор скинутий в стан логічного «0».

Приклад:

```
neg bx                ; BX = -BX
```

CMP (операнд призначення), (операнд-джерело)

Команда CMP (порівняння) віднімає операнд-джерело з операнда призначення, не змінюючи при цьому значення операндів. Операнди можуть бути байтовими або двобайтовими числами. Хоча значення операндів на змінюються, значення прапорів оновлюються, що може бути враховано в наступних командах умовного переходу. Команда CMP впливає на прапори AF, CF, OF, PF, SF і ZF. При збігу значень операндів встановлюється прапор ZF. Прапор перенесення встановлюється, якщо операнд призначення менший за операнд-джерело.

AAS

Команда AAS (корекція віднімання неупакованих десяткових чисел) коригує результат попереднього вирахування двох правильних упакованих десяткових чисел. Операндом призначення в команді віднімання повинен бути регістр AL. Команда AAS призводить значення в AL до виду правильного неупакованого десяткового числа; старший напівбайт при цьому обнуляється. AAS впливає на прапори AF і CF. Значення прапорів OF, PF, SF і ZF після виконання команди AAS невизначено.

DAS

Команда DAS (десяткова корекція вирахування) коригує результат попереднього вирахування двох правильних упакованих десяткових чисел. Операндом призначення в команді віднімання повинен бути регістр AL. Команда DAS призводить значення в AL до виду двох правильних упакованих десяткових чисел. Команда DAS впливає на прапори AF і CF. Значення прапорів OF, PF, SF і ZF після виконання команди DAS невизначено.

1.4 Команди множення

1.4.1 Множення чисел без знаку

MUL (операнд-джерело)

Команда MUL (множення) виконує беззнакове множення операнда-джерела і вмісту регістра AX (AL). У цієї команди тільки один операнд - другий множник, який повинен знаходитися в регістрі або в пам'яті. Розташування першого множника і результату задається неявно і залежить від розміру операнда. Якщо операнд-джерело однобайтовий, здійснюється множення на вміст регістра AL, а двобайтовий результат повертається регістри AH і AL. Якщо операнд-джерело двобайтовий, здійснюється множення на вміст регістра AX, а чотирьохбайтовий результат повертається в регістри DX і AX:

Розмір операнда	Множник	Результат
Байт	AL	AX
Слово	AX	DX:AX

Відмінність множення від додавання і віднімання в тому, що розрядність результату виходить в 2 рази більшою, ніж розрядність співмножників. Також і в десятковій системі - наприклад, перемножуючи двозначне число на двозначне, ми можемо отримати в результаті

максимум чотиризначне. Запис «DX:AX» означає, що старше слово результату буде знаходитися в DX, а молодше - в AX. Приклади:

*mul bl; AX = AL * BL*

*mul ax; DX: AX = AX * AX*

Операнди розглядаються як беззнакові двійкові числа. Якщо старша половина результату (регістр AH при однобайтовому множенні і DX при двобайтовому множенні) встановлюються прапори CF і OF, в іншому випадку ці прапори скидаються (тобто встановлюються в нульове значення). У цьому випадку старшу частину результату можна відкинути. Цю властивість можна використовувати в програмі, якщо результат повинен бути такого ж розміру, що і множники.

Якщо після виконання множення встановлені прапори CF і OF, це говорить про наявність значущих цифр результату в регістрі AH або DX. Вміст прапорів AF, PF SF і ZF після виконання команди множення невизначено.

1.4.2 Множення чисел зі знаком

IMUL (операнд-джерело)

Команда IMUL (цілочисельне множення зі знаком) виконує знакове множення операнда-джерела і вмісту регістра AX (AL). Якщо операнд-джерело однобайтовий, здійснюється множення на вміст регістра AL, а двобайтовий результат повертається в регістри AH і AL. Якщо операнд-джерело двобайтовий, здійснюється множення на вміст регістра AX, а чотирьохбайтовий результат повертається регістри DX і AX. Операнди розглядаються як беззнакові двійкові числа. Якщо старша половина результату (регістр AH при однобайтовому множенні і DX при двобайтовому множенні) встановлюються прапори CF і OF, в іншому випадку ці прапори додаються. Якщо після виконання множення встановлені прапори CF і OF, це говорить про наявність значущих цифр результату в регістрі AH або DX. Вміст прапорів AF, PF SF і ZF після виконання команди цілочисельного множення невизначено.

Ця команда має три форми, що розрізняються кількістю операндів:

- *З одним операндом* - форма, аналогічна команді MUL. Як операнд вказується множник. Розташування іншого множника і результату визначається за таблицею.
- *З двома операндами* - вказуються два множника. Результат записується на місце першого множника. Старша частина результату в цьому випадку ігнорується. До речі, ця форма команди не працює з операндами розміром 1 байт.
- *З трьома операндами* - вказується положення результату, першого і другого множника. Другий множник повинен бути безпосереднім значенням. Результат має такий же розмір, як перший множник, старша частина результату ігнорується. Це форма теж не працює з однобайтними множниками.

Приклади:

*imul cl; AX = AL * CL*

*imul si; DX: AX = AX * SI*

*imul bx, ax; BX = BX * AX*

*imul cx, - 5; CX = CX * (-5)*

CF = *OF* = 0, якщо множник поміщається в молодшій половині результату, інакше *CF* = *OF* = 1. Для другої і третьої форми команди *CF* = *OF* = 1 означає, що відбулось переповнення.

AAM

Команда AAM (корекція множення неупакованих десяткових чисел) призводить результат попереднього множення до двох правильно упакованих десяткових чисел. Для отримання правильного результату після виконання корекції старші напівбайти операндів які перемножуються повинні бути нульовими, а молодші повинні бути правильними двійково-десятковими цифрами.

Команда AAM впливає на прапори PF, SF і ZF. Вміст прапорів AF, CF і OF після виконання команди AAM невизначено.

1.5 Команди ділення

1.5.1 Ділення чисел без знаку

DIV (операнд-джерело)

Команда DIV (ділення) виконує беззнакове ділення вмісту акумулятора (і його розширення) на операнд-джерело. Ділення цілих двійкових чисел - це завжди ділення з залишком! Якщо операнд-джерело однобайтовий, здійснюється ділення двобайтового діленого, розташованого в регістрах AH і AL. Однобайтовий результат ділення зберігається в регістрі AL, а однобайтовий залишок - в регістрі AH. Якщо операнд-джерело двобайтовий, здійснюється ділення чотирибайтового діленого, розташованого в регістрах DX і AX. Двобайтовий результат ділення при цьому зберігається в регістрі AX, а двобайтовий залишок - в регістрі DX.

Якщо значення результату ділення перевищує розрядність акумулятора (0FFh для однобайтового ділення і 0FFFFh - для двобайтового) або здійснюється спроба ділення на нуль, генерується переривання типу 0, а результат ділення і залишок залишаються невизначеними. Вміст прапорів AF, CF, OF, PF, SF і ZF після виконання команди DIV невизначено.

Розмір операнда (дільника)	**Ділене**	**Дільник**	**Залишок**
Байт	AX	AL	AH
Слово	DX: AX	AX	DX

Приклади:

```
div cl; AL = AX / CL, залишок у AH
div di; AX = DX: AX / DI, залишок в DX
```

1.5.2 Ділення чисел зі знаком

IDIV (операнд-джерело)

Команда IDIV (цілочисельне ділення зі знаком) виконує знакове ділення вмісту акумулятора (і його розширення) на операнд-джерело. Якщо операнд-джерело однобайтовий, здійснюється ділення двобайтового діленого, розташованого в регістрах AH і AL. Однобайтовий результат ділення розміщується в регістрі AL, а однобайтовий залишок - в регістрі AH. Для однобайтового цілочисельного ділення додатнє ділене не може бути більше за значення +127 (7Fh), а негативне не може бути меншим за -127 (81h). Якщо операнд-джерело двобайтовий, здійснюється ділення чотирибайтового діленого, розміщеного в

регістрах DX і AX. Двобайтовий результат ділення при цьому розміщується в регістрі AX, а двобайтовий залишок - в регістрі DX. Для двобайтового цілочисельного ділення додатнє ділене не може бути більше значення +32767 (7FFFh), а негативне не може бути меншим за значення -32767 (8001H).

Якщо ділене позитивне і перевищує максимум або негативне і менше мінімуму, генерується переривання типу 0, а ділене і залишок залишаються невизначеними. Окремим випадком такої події є спроба ділення на нуль. Вміст прапорів AF, CF, OF, PF, SF і ZF після виконання команди IDIV невизначено.

AAD

Команда AAD (корекція ділення неупакованих десяткових чисел) модифікує вміст регістра AL перед виконанням ділення так, щоб при виконанні ділення в діленому вийшло правильне неупаковане десяткове число. Для отримання правильного результату після виконання ділення вміст регістра AH має бути нульовим. Команда AAD впливає на прапори PF, SF і ZF. Вміст прапорів AF, CF і OF після виконання команди AAD невизначено.

CBW

Команда CBW (перетворення байта в слово) розширює знак байта в регістрі AL на весь регістр AX. Команда CBW не вплине на прапори. Команда CBW може бути використана для отримання двобайтового діленого з однобайтового перед виконанням команди ділення.

CWD

Команда CWD (перетворення слова в подвійне слово) розширює знак слова в регістрі AX на пару регістрів AX і DX. CWD Команда не впливає на прапори. Команда CWD може бути використана для отримання чотирибайтового діленого з двобайтового перед виконанням команди ділення.

1.6 Команда XCHG

Однією із задач в мові програмування Асемблер може бути задача поміняти місцями значення в регістрах. Вирішити це завдання можна використовуючи оператори MOV та додаткову комірку пам'яті, ось так:

```
mov ax, 20
mov cx, 30

mov bx, ax
mov ax, cx
mov cx, bx
```

Виходить, що потрібно виконати три команди. Команда XCHG дозволяє зробити обмін однією командою. Обмін може здійснюватися між регістрами або регістрами і коміркою пам'яті.

Наприклад:

```
mov ax, 20
mov cx, 30
XCHG ax, cx; поміняти місцями
```

Використовувати цю команду процесора дуже зручно, оскільки це дозволяє оптимізувати операції пов'язані з необхідністю великої кількості замін. Наприклад, при сортуванні. Реально зменшується і розмір програми.

2. Хід роботи

1. Вивчити основні поняття та команди, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму яка просить користувача ввести число X. Потім програма обчислює формулу відповідно до варіанту і видає результат обчислення на екран. Після цього програма запитує користувача чи він хоче завершити програму і пропонує ввести символи ‘Y’ (‘y’) або ‘N’ (‘n’). Якщо користувач вводить символ ‘Y’ (‘y’) – програма завершується, а якщо користувач вводить символ N’ (‘n’), то програма переходить на початок, тобто знову пропонує ввести число X, після чого знову видає результат обчислення.

№ варіанту	Завдання
1	$Y = \frac{(X+3)^2}{2} - 1$
2	$Y = \frac{2 \cdot X - 5}{4} - 3$
3	$Y = \frac{(3 \cdot X + 4)^2 - 5}{2}$
4	$Y = \frac{X^3 - 2}{3} + 4$
5	$Y = \left(\frac{X+2}{10}\right)^2 - 3$
6	$Y = \frac{2 \cdot X - 4}{5} + 2$
7	$Y = \left(\frac{2 - X}{5} + 3\right)^2$
8	$Y = \frac{2 \cdot X - 4}{3} + 4$
9	$Y = \left(\frac{4 - X}{5}\right)^3 + 1$
10	$Y = \frac{125}{X+4} - 1$
11	$Y = \frac{101}{5 - X} + 4$
12	$Y = \frac{95}{3 + X^2} - 4$
13	$Y = \frac{88}{(4 - X)^2} + 8$

14	$Y = \frac{200}{X^3 + 1} - 4$
15	$Y = \frac{115}{X^2 - 3} + 1$
16	$Y = \frac{62}{3 - X^3} + 2$
17	$Y = \frac{50}{5 - X^4} + 8$
18	$Y = \left(\frac{4}{1 + X}\right)^2 - 5$

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці результат обчислення формули, яка задана в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Які ви знаєте команди додавання в мові програмування Асемблер?
2. Які ви знаєте команди віднімання в мові програмування Асемблер?
3. Які ви знаєте команди ділення в мові програмування Асемблер?
4. Які ви знаєте команди множення в мові програмування Асемблер?
5. Опишіть принцип дії команди ADD.
6. Опишіть принцип дії команди INC.
7. Опишіть принцип дії команди SUB.
8. Опишіть принцип дії команди DEC.
9. Опишіть принцип дії команди NEG.

10. Опишіть принцип дії команди MUL.
11. Опишіть принцип дії команди IMUL.
12. Опишіть принцип дії команди DIV.
13. Опишіть принцип дії команди IDIV.

ЛАБОРАТОРНА РОБОТА №4

ТЕМА: Використання змінних, стеку та підпрограм в мові програмування Асемблер.

МЕТА: Вивчити правила оголошення та використання змінних в мові програмування Асемблер та навчитись застосовувати стек та підпрограми.

1. Теоретичні відомості

1.1 Змінні в програмі на асемблері

Змінна - це місце в пам'яті яке має ім'я і тип. Створюючи програму на асемблері нам потрібно здійснювати визначення наших змінних та констант. Змінні зазвичай визначаються в сегменті даних. Асемблер підтримує ряд директив які служать для виділення місця в оперативній пам’яті для змінних. Наприклад:

db - 1 байт (8 біт)

dw - 2 байти (16 біт)

dd - 4 байти (32 біт)

dq – 8 байт (64 біти)

```
.MODEL small
.STACK 4096
.DATA
        Data1 DB 31h          ; виділити один байт з вмістом 31h
.CODE
main PROC
        mov   ax, @data       ; встановлення в ds адреси
        mov   ds, ax          ; сегменту даних
        mov   ax, OFFSET Data1  ; де знаходитися змінна
 Exit:
        mov   ah, 04Ch        ; функція DOS виходу з програми
        mov   al, 0h          ; код повернення
        int   21h             ; Виклик DOS зупинка програми
main ENDP
END main
```

Так само цю змінну можна поміщати в регістр ось так наприклад:

```
main PROC
        mov   ax, @data       ; встановлення в ds адреса
        mov   ds, ax          ; сегменту даних
        mov   ax, OFFSET Data1  ; де знаходитися змінна
        mov   al, Data1       ; поміщаємо в регістр AX (AL)
 Exit:
```

Можна і зворотну операцію здійснити - помістити дані з регістра в змінну:

```
main PROC
        mov   ax, @data          ; встановлення в ds адреси
        mov   ds, ax             ; сегменту даних
        mov   ax, OFFSET Data1   ; де знаходитися змінна
        mov   al, Data1          ; поміщаємо в регістр AX (AL)
        inc   al                 ; збільшити на одиницю
        mov   Data1, al          ; в пам'ять
Exit:
```

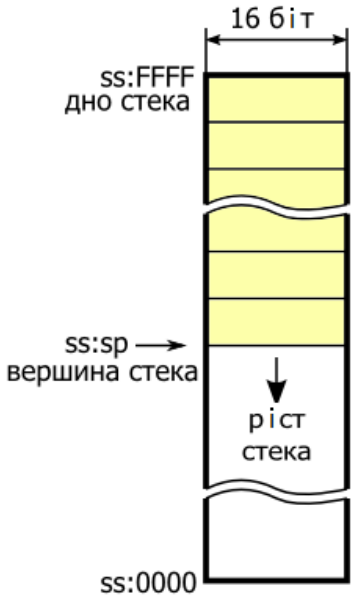
1.2 Стек

Стек - це спеціальна область пам'яті. Адресацією в цій області управляє регістр SP або вказівник стека. Використовується ця пам'ять в основному для тимчасового зберігання вмісту регістрів. Саме тимчасового. Найголовніше зрозуміти, як працює стек. Він функціонує за принципом перший прийшов останній пішов - LIFO («Last In - First Out» або «останнім прийшов - першим пішов»). Розглянемо цей процес на прикладі: хтось ставить книгу на стіл, потім зверху ще одну книгу. Перша книга лежить внизу. Що б витягнути її спочатку потрібно зняти верхню і тільки після цього буде доступ до першої.

Зазвичай стек використовується для збереження адрес повернення і передачі аргументів під час виклику процедур, також у ньому виділяється пам'ять для локальних змінних. Крім того, в стеку можна тимчасово зберігати значення регістрів.

Стек є невід'ємною частиною архітектури процесора і підтримується на апаратному рівні: у процесорі є спеціальні регістри (SS, BP, SP) і команди для роботи зі стеком.

Схема організації стека в процесорі 8086 показана на рисунку:



Стек розміщується в оперативній пам'яті в сегменті стека, і тому адресується відносно початку сегментного регістра SS. Шириною стека називається розмір елементів, які можна поміщати в нього або витягувати. У нашому випадку ширина стека дорівнює двом байтам або 16 бітам. Регістр SP (вказівник стека) містить адресу останнього доданого елемента. Ця

адреса також називається вершиною стека. Протилежний кінець стека називається дном.

Дно стека знаходиться у верхніх адресах пам'яті. При додаванні нових елементів у стек значення регістра SP зменшується, тобто стек росте в бік молодших адрес.

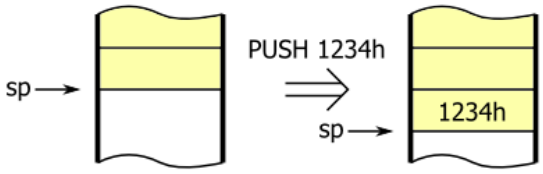
Для стека існують всього дві основні операції:

- додавання елемента на вершину стека (PUSH);
- витягування елемента з вершини стека (POP);

1.2.1 Додавання елемента в стек командою PUSH

Виконується командою PUSH. У цієї команди один операнд, який може бути безпосереднім значенням, 16-бітним регістром (у тому числі сегмент) або 16-бітної змінної в пам'яті. Команда працює наступним чином:

1. Значення в регістрі SP зменшується на 2 (так як ширина стека - 16 біт або 2 байти);
2. Операнд поміщається в пам'ять за адресою в SP.



Приклади:

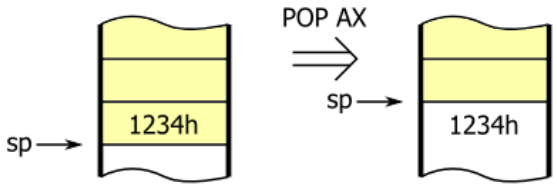
```
push -5          ;Помістити -5 в стек
push ax          ;Помістити AX в стек
push ds          ;Помістити DS в стек
push x           ;Помістити x в стек (змінна x визначена як слово)
```

Існують ще 2 команди для додавання в стек. Команда PUSHF поміщає в стек вміст регістра прапорів. Команда PUSHA поміщає в стек вміст всіх регістрів загального призначення в наступному порядку: AX, CX, DX, BX, SP, BP, SI, DI (значення DI буде на вершині стека). Значення SP поміщається те, яке було до виконання команди. Обидві ці команди не мають операндів.

1.2.2 Витягування елемента з стека командою POP

Виконується командою POP. У цієї команди також є один операнд, який може бути 16-бітним регістром (у тому числі сегментним, але крім CS) або 16-бітною змінною в пам'яті. Команда працює наступним чином:

1. Операнд читається з пам'яті за адресою в SP;
2. Значення в регістрі SP збільшується на 2.



Зверніть увагу, що витягнутий зі стеку елемент не обнуляється і не затирається в пам'яті, а просто залишається як сміття. Він буде перезаписаний при поміщенні нового значення в стек.

Приклади:

```
pop cx                  ;Помістити значення з стека в CX
pop es                  ;Помістити значення з стека в ES
pop x                   ;Помістити значення з стека в змінну x
```

Відповідно, є ще 2 команди. POPF відновлює значення з вершини стека в регістр прапорів. POPA відновлює з стека всі регістри загального призначення (але при цьому значення для SP ігнорується).

1.3 Процедури

Процедурою називається код, який може виконуватися багаторазово і до якого можна звертатися з різних частин програми. Зазвичай процедури призначені для виконання якихось окремих, закінчених дій програми і тому їх іноді називають підпрограмами. В інших мовах програмування процедури можуть називатися функціями або методами, але по суті це все одне й те ж.

1.3.1 Команди CALL і RET

Для роботи з процедурами призначені команди CALL і RET. Крім того для позначення початку і закінчення процедури використовуються директиви PROC і ENDP, які потрібно вставляти після назви процедури відповідно перед тілом процедури та після неї. За допомогою команди CALL виконується виклик процедури. Ця команда працює майже так само, як команда безумовного переходу (JMP), але з однією відмінністю - одночасно в стек зберігається поточне значення регістра IP. Це дозволяє потім повернутися до того місця в коді, звідки була викликана процедура. В якості операнда вказується адреса переходу, яка може бути безпосереднім значенням (міткою), 16-розрядним регістром (крім сегментних) або коміркою пам'яті, яка містить адресу.

Повернення з процедури виконується командою RET. Ця команда відновлює значення з вершини стека в регістр IP. Таким чином, виконання програми триває з команди, яка є наступною відразу після команди CALL. Зазвичай код процедури закінчується цією командою. Команди CALL і RET не змінюють значення прапорів (крім деяких особливих випадків у захищеному режимі). Невеликий приклад різних способів виклику процедури:

```
mov ax,myproc
mov bx,myproc_addr

call myproc             ; Виклик процедури (адреса перехода – мітка myproc)
```

```
    call ax              ; Виклик процедури по адресі в AX
    call [myproc_addr]   ; Виклик процедури по адресі в змінній

    mov ax,4C00h
    int 21h              ; Завершення програми
;-------------------------------------------------------------------
;Процедура, яка додає між собою 3 числа

myproc PROC
add ax,bx
add ax,cx
ret                  ; Вихід з процедури
myproc ENDP
;-------------------------------------------------------------------
myproc_addr dw myproc  ;Змінна с адресом процедури
```

1.3.2 Близький та далекий виклик процедур

Існує два типи викликів процедур. ***Близьким*** називається виклик процедури, яка знаходиться в поточному сегменті коду. ***Далекий*** виклик - це виклик процедури в іншому сегменті. Відповідно існують 2 види команди RET - для ближнього і далекого повернення. Компілятор автоматично визначає потрібний тип машинної команди, тому в більшості випадків не потрібно про це турбуватися.

1.3.3 Передача параметрів

Дуже часто виникає необхідність передати процедурі якісь параметри. Наприклад, якщо ви пишете процедуру для обчислення суми елементів масиву, зручно в якості параметрів передавати їй адреси масиву і його розмір. У такому випадку одну і ту ж процедуру можна буде використовувати для різних масивів у вашій програмі. Найпростіший спосіб передати параметри - це помістити їх в регістри перед викликом процедури.

1.3.4 Значення, що повертається

Крім передачі параметрів часто потрібно отримати якесь значення з процедури. Наприклад, якщо процедура щось обчислює, хотілося б отримати результат обчислення. А якщо процедура щось робить, то корисно дізнатися, завершилася дія успішно чи виникла помилка. Існують різні способи повернення значення з процедури, але найбільш популярний - це помістити значення в один з регістрів. Зазвичай для цієї мети використовують регістри AL і AX.

1.3.5 Збереження регістрів

Хорошим прийомом є збереження регістрів, які процедура змінює в ході свого виконання. Це дозволяє викликати процедуру з будь-якої частини коду і не турбуватися, що значення в регістрах будуть зіпсовані. Зазвичай регістри зберігаються в стеку за допомогою команди PUSH, а перед поверненням з процедури відновлюються командою POP. Звичайно, що відновлювати їх треба в зворотному порядку. Приблизно ось так:

```
Myproc PROC
  push bx          ; Збереження регістрів
  push cx
  push si
  ...          ; Код процедури
  pop si           ; Відновлення регістрів
  pop cx
  pop bx
  ret          ; Вихід з процедури
myproc ENDP
```

2. Хід роботи

1. Вивчити основні поняття та оператори, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму яка просить користувача ввести змінну **X**, коефіцієнти **A**, **B**. Потім програма обчислює формулу відповідно до варіанту і видає результат обчислення на екран. Програма повинна працювати як з від'ємними так і з додатними цілими числами. Після цього програма запитує користувача чи він хоче завершити програму і пропонує ввести символи 'Y' ('y') або 'N' ('n'). Якщо користувач вводить символ 'Y' ('y') – програма завершується, а якщо користувач вводить символ N' ('n'), то програма переходить на початок, тобто знову пропонує ввести змінну і коефіцієнти, після чого знову видає результат обчислення. В коді програми повинні використовуватись підпрограми і стек.

№ варіанту	**Завдання**
1	*якщо x<0, то* $Y = \frac{(X + 3 \cdot A)^2}{2} - 1 \cdot B$ *якщо x≥0, то* $Y = \frac{(X - 3 \cdot A)^2}{2} - 1 \cdot B$
2	*якщо x<10, то* $Y = \frac{2 \cdot X - 5}{4 \cdot A} - 3 \cdot B$ *якщо x≥10, то* $Y = \frac{2 \cdot X - 5}{4 \cdot A} + 3 \cdot B$
3	*якщо x<20, то* $Y = \frac{(3 \cdot X + 4 \cdot A)^2 - 5}{2 \cdot B}$ *якщо x≥20, то* $Y = \frac{(3 \cdot X - 4 \cdot A)^2 - 5}{2 \cdot B}$
4	*якщо x<30, то* $Y = \frac{X^3 - 2 \cdot A}{3} + 4 \cdot B$

	якщо x≥30, то $Y = \frac{X^3 - 2 \cdot A}{3} - 4 \cdot B$
5	*якщо x<40, то* $Y = \left(\frac{X \cdot A + 2}{10 \cdot B} \right)^2 - 3$ *якщо x≥40, то* $Y = \left(\frac{X \cdot A - 2}{10 \cdot B} \right)^2 - 3$
6	*якщо x<6, то* $Y = \frac{2 \cdot X - 4 \cdot A}{5} + 2 \cdot B$ *якщо x≥6, то* $Y = \frac{2 \cdot X - 4 \cdot A}{5} - 2 \cdot B$
7	*якщо x<7, то* $Y = \left(\frac{2 \cdot A - X}{5} + 3 \cdot B \right)^2$ *якщо x≥7, то* $Y = \left(\frac{2 \cdot A - X}{5} - 3 \cdot B \right)^2$
8	*якщо x<8, то* $Y = \frac{2 \cdot X \cdot A - 4}{3 \cdot B} + 4$ *якщо x≥8, то* $Y = \frac{2 \cdot X \cdot A - 4}{3 \cdot B} - 4$
9	*якщо x<9, то* $Y = \left(\frac{4 \cdot A - X}{5 \cdot B} \right)^3 + 1$ *якщо x≥9, то* $Y = \left(\frac{4 \cdot A + X}{5 \cdot B} \right)^3 + 1$
10	*якщо x<100, то* $Y = \frac{125 \cdot A}{X \cdot B + 4} - 1$ *якщо x≥100, то* $Y = \frac{125 \cdot A}{X \cdot B - 4} + 1$
11	*якщо x<12, то* $Y = \frac{101}{5 \cdot BX} + 4 \cdot A$ *якщо x≥12, то* $Y = \frac{101}{5 \cdot B - X} - 4 \cdot A$
12	*якщо x<25, то* $Y = \frac{95 \cdot A}{3 \cdot B + X^2} - 4$ *якщо x≥25, то* $Y = \frac{95 \cdot A}{3 \cdot B - X^2} - 4$
13	*якщо x<15, то* $Y = \frac{88 \cdot A}{(4 \cdot B - X)^2} + 8$ *якщо x≥15, то* $Y = \frac{88 \cdot A}{(4 \cdot B + X)^2} + 8$
14	*якщо x<31, то* $Y = \frac{200 \cdot A}{X^3 + 1} - 4 \cdot B$ *якщо x≥31, то* $Y = \frac{200 \cdot A}{X^3 + 1} + 4 \cdot B$
15	*якщо x<42, то* $Y = \frac{115 \cdot B}{X^2 - 3 \cdot A} + 1$ *якщо x≥42, то* $Y = \frac{115 \cdot B}{X^2 - 3 \cdot A} - 1$

16	*якщо x<55, то* $Y = \frac{62}{3 \cdot A - X^3} + 2 \cdot B$ *якщо x≥55, то* $Y = \frac{62}{3 \cdot A + X^3} + 2 \cdot B$
17	*якщо x<99, то* $Y = \frac{50}{5 \cdot A - X^4} + 8 \cdot B$ *якщо x≥99, то* $Y = \frac{50}{5 \cdot A - X^4} - 8 \cdot B$
18	*якщо x<17, то* $Y = \left(\frac{4}{1 + X \cdot A}\right)^2 - 5 \cdot B$ *якщо x≥17, то* $Y = \left(\frac{4}{1 - X \cdot A}\right)^2 - 5 \cdot B$

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці результат обчислення формули, яка задана в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Яка команда використовується для виклику процедури в мові програмування Асемблер?
2. Яка команда використовується для завершення процедури в мові програмування Асемблер?
3. Як оголошується змінна в мові програмування Асемблер?
4. Які є типи змінних в мові програмування Асемблер?
5. Що таке стек?
6. Які команди використовуються для роботи з стеком?
7. Що таке процедура?

ЛАБОРАТОРНА РОБОТА №5

ТЕМА: Команди роботи з бітами. Логічні операції в мові програмування Assembler.

МЕТА: Вивчити основні команди для роботи з бітами мови програмування Асемблер для здійснення логічних операцій та навчитись використовувати їх в процесі написання програм.

1. Теоретичні відомості

1.1 Вивід на екран числа в двійковій формі

Даний код здійснює вивід двобайтового числа в двійковій системі числення. Число, яке потрібно вивести, повинно бути в регістрі AX.

```
   mov bx,ax
   mov cx,16
ob1:
   shl bx,1
   jc ob2

   mov dl,'0'
   jmp ob3

ob2:
   mov dl,'1'
ob3:
   mov ah,2
   int 21h
   loop ob1
```

1.2 Логічні операції

Таблиця 1 – Команди мови Асемблер для роботи з бітами

Логічні операції	
NOT	Інверсія байта або слова
AND	Операція "І" над байтами або словами
OR	Операція "АБО" над байтами або словами
XOR	Операція "ВИКЛЮЧАЮЧЕ АБО" над байтами або словами
TEST	Перевірка байта або слова
Команди зсуву	
SHL / SAL	Логічний / арифметичний зсув вліво байта або слова
SHR	Логічний зсув вправо байта або слова
SAR	Арифметичний зсув вправо байта або слова
Команди циклічного зсуву	

ROL	Циклічний зсув вліво байта або слова
ROR	Циклічний зсув вправо байта або слова
RCL	Циклічний зсув вліво байта або слова через розряд переносу
RCR	Циклічний зсув вправо байта або слова через розряд переносу

Логічні операції виконуються порозрядно, тобто окремо для кожного біта операндів. В результаті виконання змінюються прапори. У програмах ці операції часто використовуються для скидання в стан логічного «0», встановлення в стан логічної «1» або інверсії окремих бітів двійкових чисел.

1.2.1 Логічне І

Якщо обидва біти рівні 1, то результат дорівнює 1, інакше результат дорівнює 0.

AND	**0**	**1**
0	0	0
1	0	1

Для виконання операції логічного І призначена команда AND. У цієї команди 2 операнда, результат поміщається на місце першого операнда. Часто ця команда використовується для обнулення певних бітів числа. При цьому другий операнд називають маскою. Обнуляються ті біти операнда, які в масці рівні 0, значення інших бітів зберігаються.

Приклади:

```
and ax, bx          ; AX = AX & BX
and cl, 11111110b   ; Обнулення молодшого біта CL
and dl, 00001111b   ; Обнулення старшої тетради DL
```

Ще одне використання цієї команди - швидке обчислення залишку від ділення на степінь 2. Наприклад, так можна обчислити залишок від ділення на 8:

```
and ax, 111b; AX = залишок від ділення AX на 8
```

1.2.2 Логічне АБО

Якщо хоча б один з бітів дорівнює 1, то результат дорівнює 1, інакше результат дорівнює 0.

OR	**0**	**1**
0	0	1
1	1	1

Логічне АБО обчислюється за допомогою команди OR. У цієї команди теж 2 операнда, і результат поміщається на місце першого. Часто це команда використовується для встановлення в 1 певних бітів числа. Якщо біт маски дорівнює 1, то біт результату буде дорівнювати 1, інші біти збережуть свої значення.

Приклади:

```
or al, dl            ; AL = AL | DL
or bl, 10000000b     ; Встановити знаковий біт BL
or cl, 00100101b     ; Встановити біти 0,2,5 CL
```

1.2.3 Логічне НЕ (інверсія)

Кожен біт операнда змінює своє значення на протилежне (0 → 1, 1 → 0). Операція виконується за допомогою команди NOT. У цієї команди є тільки один операнд. Результат поміщається на місце операнда. Ця команда не змінює значення прапорів.

Приклад:

```
not bx               ; Інверсія байта за адресою в BX
```

1.2.4 Логічне ВИКЛЮЧАЮЧЕ АБО (сума за модулем два)

Якщо біти мають однакове значення, то результат дорівнює 0, інакше результат дорівнює 1.

XOR	**0**	**1**
0	0	1
1	1	0

Виключаючим АБО ця операція називається тому, що результат дорівнює 1, якщо один біт дорівнює 1 або інший дорівнює 1, а випадок, коли обидва рівні 1, виключається. Ще ця операція нагадує додавання, але в межах одного біта, без переносу. 1 + 1 = 10, але перенесення в інший розряд ігнорується і виходить 0, звідси назва «сума за модулем 2». Для виконання цієї операції призначена команда XOR. У команди два операнди, результат поміщається на місце першого. Команду можна використовувати для інверсії певних бітів операнда. Інвертуються ті біти, які в масці рівні 1, останні зберігають своє значення.

Приклади:

```
xor si, di           ; SI = SI ^ DI
xor al, 11110000b    ; Інверсія старшої тетради AL
xor bp, 8000h        ; Інверсія знакового біта BP
```

Позначення операції в коментарі до першого рядка використовується в багатьох мовах високого рівня (наприклад C, C + +, Java і т.д.). Часто XOR використовують для обнулення регістрів. Якщо операнди рівні, то результат операції завжди дорівнює 0. Такий спосіб обнулення працює швидше і, на відміну від команди MOV, не містить безпосереднього операнда, тому команда виходить коротшою (і не містить нульових байтів):

```
mov bx, 0            ; Ця команда займає 3 байти
xor bx, bx           ; А ця - всього 2
```

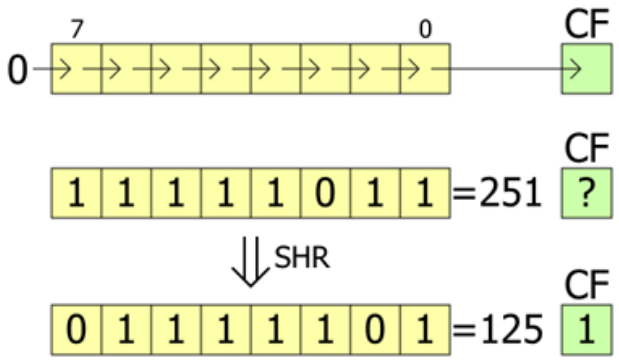
1.3 Лінійний зсув

Зсув - це особлива операція процесора, яка дозволяє реалізувати різні перетворення

даних, працювати з окремими бітами, а також швидко виконувати множення і ділення чисел на степінь 2.

1.3.1 Логічний зсув вправо

Логічний зсув завжди виконується без урахування знакового біта. Для логічного зсуву вправо призначена команда SHR. У цієї команди два операнда. Перший операнд являє собою значення яке потрібно зсувати і на його місце записується результат операції. Другий операнд вказує, на скільки біт потрібно здійснити зсув. Цим операндом може бути або безпосереднє значення, або регістр CL. Схема виконання операції показана на малюнку:



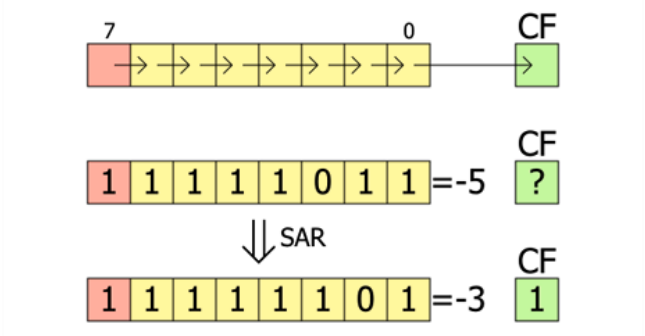
Всі біти операнда зсуваються вправо (від старших бітів до молодших). Зсунутий біт стає значенням прапора CF. Старший біт отримує нульове значення. Ця операція повторюється кілька разів, якщо другий операнд більший за одиницю. Логічний зсув вправо можна використовувати для ділення цілих чисел без знаку на ступінь 2, причому зсув працює швидше, ніж команда ділення DIV.

Приклади:

```
shr ax, 1           ; Логічний зсув AX на 1 біт вправо
shr byte bx, cl             ; Логічний зсув байта за адресою BX на CL біт вправо
shr cl, 4           ; CL = CL / 16 (для числа без знака)
```

1.3.2 Арифметичний зсув вправо

Арифметичний зсув відрізняється від логічного тим, що він не змінює значення старшого біта, і призначений для чисел із знаком. Арифметичний зсув вправо виконується командою SAR. У цієї команди теж 2 операнда, аналогічно команді SHR. Схема виконання операції показана на малюнку:



Зсунутий біт стає значенням прапора CF. Знаковий біт не змінюється. При зсуві на 1 біт скидається прапор OF. Цю команду можна використовувати для ділення цілих чисел із знаком на ступінь 2 («округлення» завжди в бік меншого числа, тому для негативних чисел

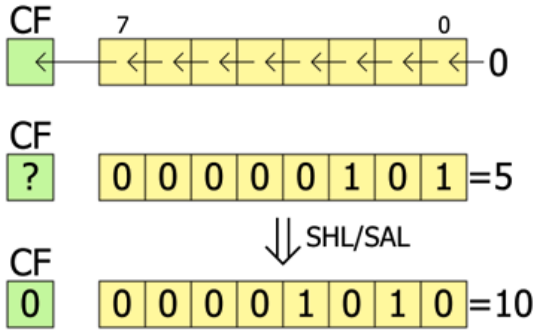
результат буде відрізнятися від результату ділення за допомогою команди IDIV).

Приклади:

```
sar bx, 1          ; Арифметичний зсув BX на 1 біт вправо
sar di, cl         ; Арифметичний зсув DI на CL біт вправо
sar x, 3                ; x = x / 8 (x - 8-бітове значення зі знаком)
```

1.3.3 Логічний і арифметичний зсув вліво

Логічний зсув вліво виконується командою SHL, а арифметичний – командою SAL. Однак, насправді це просто синоніми для однієї і тієї ж машинної команди. Зсув вліво однаковий для чисел із знаком і чисел без знаку. У команди 2 операнда, аналогічно до команд SHR і SAR. Схема цієї операції показана на малюнку:



Старший біт стає значенням прапора CF, а молодший отримує нульове значення. За допомогою зсуву вліво можна швидко множити числа на ступінь 2. Але потрібно слідкувати за тим, щоб не отримати в результаті переповнення. Якщо при зсуві на 1 біт змінюється значення старшого біта, то встановлюється прапор OF.

Приклади:

```
shl dx,1           ; Зсув DX на 1 біт вліво
sal dx,1           ; Те ж саме
shl ax,cl          ; Зсув AX на CL біт вліво
sal x,2                 ; x = x * 4
```

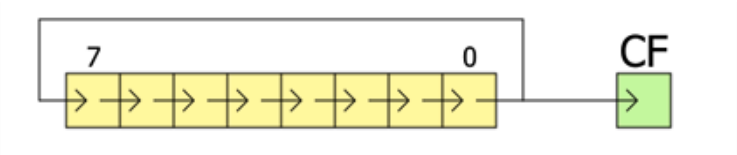
1.4 Циклічний зсув

Циклічний зсув відрізняється від лінійного тим, що висунуті з одного кінця біти з'являються з іншого боку, тобто рухаються по колу. У процесора існує 2 види циклічного зсуву: простий і через прапор переносу (CF). У всіх команд, які розглядаються в цій частині навчального курсу, по 2 операнда, таких же, як у команд лінійного зсуву. Перший операнд – значення, яке потрібно зсувати і місце для запису результату. Другий операнд - лічильник зсувів, який може знаходиться в регістрі CL або вказуватися безпосередньо.

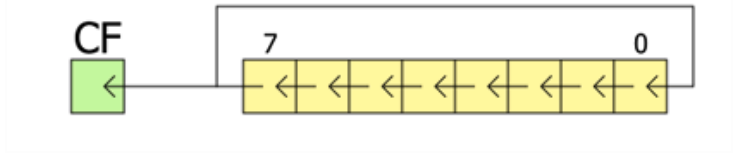
1.4.1 Простий циклічний зсув

Циклічний зсув вправо виконується командою ROR, а вліво - командою ROL. Схема роботи цих команд представлена на малюнку (на прикладі 8-бітного операнда):

Циклічний зсув вправо (ROR)



Циклічний зсув вліво (ROL)



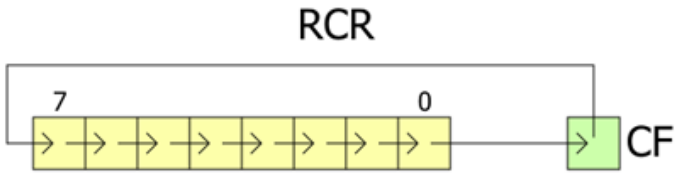
Значення останнього висунутого біта копіюється в прапор CF. Для зсуву на 1 біт встановлюється прапор OF, якщо в результаті зсуву змінюється знаковий біт операнда.

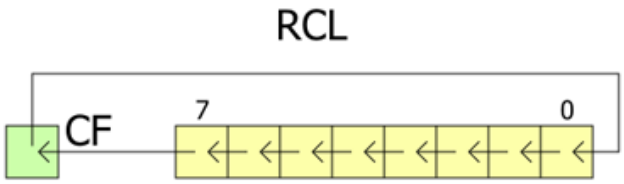
Приклади:

```
rol bl, 1              ; Циклічний зсув BL на 1 біт вліво
ror word si, 5              ; Циклічний зсув слова за адресою в SI на 5 біт вправо
rol ax, cl             ; Циклічний зсув AX на CL біт вліво
```

1.4.2 Циклічний зсув через прапор переносу

Відмінність від простого циклічного зсуву в тому, що прапор CF бере участь у зсуві нарівні з бітами операнда. При зсуві на 1 біт, який зсувається, поміщається в CF, а значення CF зсувається в операнд з іншого боку. При зсуві на кілька біт ця операція повторюється багато разів. Циклічний зсув через прапор перенесення виконується командами RCR (вправо) і RCL (вліво).





Для зсуву на 1 біт встановлюється прапор OF, якщо в результаті зсуву змінюється знаковий біт операнда.

Приклади:

```
rcr dh, 3              ; Цикл. зсув DH на 3 біти вправо через прапор CF
rcl byte bx, cl        ; Цикл. зсув байта за адресою в BX на CL біт вліво через CF
rcl dx, 1              ; Цикл. зсув DX на 1 біт вліво через прапор CF
```

1.4.3 Порівняння бітів TEST

Команда TEST працює так само як і логічна операція AND. Різниця полягає в тому, що результат не зберігається, але зате змінюється стан прапорів для подальшого використання команд умовного переходу. Дана властивість дозволяє використовувати TEST для перевірки прапорів. Отже, як поведе себе команда TEST. Розглянемо певне число в двійковій системі числення:

00000100 (4 в десятковій системі числення)

Щоб перевірити це число на встановлення цього біта в 1, потрібно написати:

test 4,4

А далі можна використати команду умовного переходу наприклад JNZ. Перехід буде здійснений по мітці, якщо одиниці збігаються. Але справа в тому, що цей біт встановлений не тільки для числа 4, але ще і для:

5 (00000101)

7 (00000111)

12 (00001100)

І у всіх випадках команда TEST повинна працювати однаково, оскільки встановлено цей біт.

2. Хід роботи

1. Вивчити основні поняття та команди, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму яка просить користувача ввести число X. Після введення числа, програма виводить його в новому рядку в двійковій формі. Після цього над числом здійснюються операції відповідно до варіанту, які вказані в таблиці. Після кожної операції виводити проміжний результат перетворення в новому рядку і вказувати яка операція здійснювалась. Після цього програма запитує користувача чи він хоче завершити програму і пропонує ввести символи ‘Y’ (‘y’) або ‘N’ (‘n’). Якщо користувач вводить символ ‘Y’ (‘y’) – програма завершується, а якщо користувач вводить символ N’ (‘n’), то програма переходить на початок.

№ варіанту	**Завдання**
1	1. Обнулити 1,2,10 біт 2. Встановити всі біти старшого байта в 1 3. Інвертувати 4,5,13,14 біти
2	1. Арифметично зсунути всі байти вправо на 1 біт 2. Інвертувати всі біти молодшого байта 3. Обнулити 11,13,15 біти
3	1. Логічно зсунути всі байти вліво на 2 біти 2. Встановити всі біти молодшої тетради старшого байта в 1 3. Інвертувати 3,4,7,15 біти

4	1. Циклічно зсунути всі байти вправо на 3 біти 2. Інвертувати всі біти старшого байта 3. Встановити всі біти старшої тетради молодшого байта в 0
5	1. Циклічно зсунути всі байти вліво на 2 біти 2. Встановити всі біти молодшого байта в 1 3. Інвертувати 3,4,7,15 біти
6	1. Інвертувати 1,4,12,15 біти 2. Встановити 12,13,15 біти в 1 3. Встановити всі біти молодшої тетради молодшого байта в 0
7	1. Логічно зсунути всі байти вліво на 4 біти 2. Інвертувати 4,6,8,15 біти 3. Встановити всі біти старшої тетради старшого байта в 1
8	1. Встановити 1,2,14,15 біти в 0 2. Встановити в 1 всі біти старшого байта 3. Інвертувати 6,9,11,14 біти
9	1. Встановити в 1 всі біти молодшого байта 2. Циклічно зсунути всі байти вправо на 4 біти 3. Інвертувати 0,2,11,15 біти
10	1. Інвертувати 8,12,13,15 біти 2. Встановити 0,2,4,12 біти в 1 3. Циклічно зсунути всі байти вправо на 1 біт
11	1. Встановити 0,3,7,12 біти в 1 2. Арифметично зсунути всі байти вправо на 2 біти 3. Інвертувати 1,2,4,15 біти
12	1. Обнулити 11,13,15 біти 2. Інвертувати 1,4,7,15 біти 3. Встановити 0,5,9,14 біти в 1
13	1. Встановити 0,2,4,6 біти в 1 2. Обнулити 11,13,15 біти 3. Інвертувати 3,4,7,15 біти
14	1. Інвертувати 3,4,7,15 біти 2. Встановити 0,2,10,11 біти в 1 3. Обнулити 5,7,9 біти
15	1. Встановити 1,3,9,15 біти в 1 2. Циклічно зсунути всі байти вліво на 3 біти 3. Інвертувати 3,4,7,15 біти

16	1. Обнулити 2,13,15 біти 2. Інвертувати 3,4,7,15 біти 3. Встановити 1,6,9,14 біти в 1
17	1. Інвертувати 3,4,7,15 біти 2. Обнулити 4,8,9 біти 3. Встановити 0,2,3,13 біти в 1
18	1. Інвертувати 3,4,7,15 біти 2. Встановити 2,5,8,11 біти в 1 3. Обнулити 1,3,5 біти

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці результат обчислення формули, яка задана в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи.
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Які ви знаєте логічні команди в мові програмування Асемблер?
2. Як здійснюється циклічний зсув вліво? Які команди для цього використовуються?
3. Як здійснюється циклічний зсув вправо? Які команди для цього використовуються?
4. Як здійснюється арифметичний зсув вліво? Які команди для цього використовуються?
5. Як здійснюється арифметичний зсув вправо? Які команди для цього використовуються?
6. Як виконується логічне АБО? Які команди для цього використовуються?
7. Як виконується логічне І? Які команди для цього використовуються?
8. Як виконується логічне НЕ? Які команди для цього використовуються?

ЛАБОРАТОРНА РОБОТА №6

ТЕМА: Застосування циклів в мові програмування Assembler.

МЕТА: Вивчити принципи організації циклів в мові програмування Асемблер.

1. Теоретичні відомості

1.1 Організація циклу з допомогою команди LOOP

В мові програмування Асемблер існує простий спосіб організації циклу використовуючи команду LOOP. Ця команда виглядає так:

LOOP мітка

У цієї команди є лише один операнд - ім'я мітки, на яку здійснюється перехід. В якості лічильника циклу використовується регістр CX. Команда LOOP виконує декремент (зменшення на 1) вмісту регістру CX, а потім перевіряє його значення. Якщо вміст CX не дорівнює нулю, то здійснюється перехід на мітку, в іншому випадку керування переходить до наступної після LOOP команди.

Вміст регістру CX інтерпретується командою як число без знаку. У CX потрібно помістити число, яке рівне необхідній кількості повторень циклу. Зрозуміло, що максимум може бути 65535 повторень. Ще одне обмеження пов'язане з дальністю переходу. Мітка повинна знаходитися в діапазоні -128 ... +127 байт від команди LOOP (якщо це не так, з’явиться повідомлення про помилку).

1.2 Приклад циклу

В якості прикладу розглянемо просту програму, яка відображатиме всі букви англійського алфавіту. ASCII-коди цих символів розташовані послідовно, тому можна виводити їх у циклі. Для виведення символу на екран використовується функція DOS 02h. Значення коду символу, який потрібно вивести на екран, потрібно занести в регістр DL.

```
.MODEL small
.STACK 4096
.DATA
        Press db 13, 10, 'Press any key ... $'
.CODE
main PROC
        mov ax,@Data
        mov ds,ax      ; встановити регістр DS таким чином, щоб він вказував на сегмент даних
        mov ah, 02h ; Для виклику функції DOS 02h - вивід символу
        mov dl, 'A'          ; Перший виведений символ
        mov cx, 26           ; Лічильник повторень циклу
mitka:
```

```
        int 21h             ; Звернення до функції DOS
        inc dl          ; Наступний символ
        loop mitka          ; Команда циклу
        mov dx, offset press ; поміщуємо DX адреса рядка
        mov ah, 09h ; Функція DOS 09h - вивід рядка
        int 21h             ; Звернення до функції DOS
        mov ah, 08h ; Функція DOS 08h - введення символу
        int 21h             ; Звернення до функції DOS
        mov ax, 4C00h       ;
        int 21h             ; Завершення програми;
main ENDP
END main
```

Команди «int 21h» і «inc dl» (рядки 8 і 9) будуть виконуватися в циклі 26 разів. Для того, щоб програма не закрилася відразу, використовується функція DOS 08h – очікування введення символу з клавіатури без відображення його на дисплеї. Перед цим виводиться на екран пропозиція натиснути будь-яку кнопку.

1.3 Вкладені цикли

Іноді потрібно організувати вкладений цикл, тобто цикл всередині іншого циклу. У цьому випадку необхідно зберегти значення CX перед початком вкладеного циклу і відновити після його завершення (перед командою LOOP зовнішнього циклу). Зберегти значення можна в інший регістр, в тимчасову змінну або в стек. Наступна програма виводить всі доступні ASCII-символи у вигляді таблиці 16×16. Значення лічильника зовнішнього циклу зберігається в регістрі BX.

```
.MODEL small
.STACK 4096
.DATA
        Press db 13, 10, 'Press any key ... $'
.CODE
main PROC
        mov ax,@Data
        mov ds,ax           ; встановити регістр DS таким чином, щоб він вказував на сегмент даних
            mov ah,02h      ; Для виклику функції DOS 02h - вивід символу
            sub dl,dl       ; Перший символ
            mov cx,16       ; Лічильник зовнішнього циклу (по стрічках)
```

```
lp1:
    mov bx,cx          ; Зберігаємо лічильник в BX
    mov cx,16          ; Лічильник внутрішнього циклу (по стовпчиках)
lp2:
    int 21h            ; Звернення до функції DOS
    inc dl             ; Наступний символ
    loop lp2           ; Команда внутрішнього циклу

    mov dh,dl          ; Зберігаємо значення DL в DH
    mov dl,13
    int 21h
    mov dl,10          ; Перехід на наступну стрічку
    int 21h
    mov dl,dh          ; Відновлюємо значення DL

    mov cx,bx          ; Відновлюємо значення лічильника
    loop lp1           ; Команда зовнішнього циклу

    mov dx,offset press        ; поміщуємо DX адреса рядка
    mov ah,09h         ; Функція DOS 09h - вивід рядка
    int 21h            ; Звернення до функції DOS
    mov ah,08h         ; Функція DOS 08h - введення символу
    int 21h            ; Звернення до функції DOS
    mov ax,4C00h       ;
    int 21h            ; Завершення програми;
main ENDP
END main
```

1.4 Команди LOOPZ і LOOPNZ

Окрім команди LOOP і команд умовних переходів існують ще дві команди, які дозволяють організовувати цикли. Це команди LOOPZ (або її синонім LOOPE) і LOOPNZ (синонім - LOOPNE). Дія цих команд дуже нагадує LOOP, за винятком того, що додатково аналізується прапор нуля ZF.

Перехід до мітки циклу здійснюється в тому випадку, якщо після декремента вміст CX не дорівнює 0 і виконується умова: ZF = 1 (для команди LOOPZ / LOOPE) або ZF = 0 (LOOPNZ / LOOPNE).

Ці команди зручно використовувати в алгоритмах, де цикл повинен завершуватися у

двох випадках:

- виконано необхідну кількість ітерацій;
- виконана деяка умова дострокового завершення циклу.

Найпростіший приклад такого алгоритму – пошук числа або символу в масиві. Пошук завершується, якщо один з елементів масиву збігся з шуканим або якщо досягнуто кінця масиву.

2. Хід роботи

Лабораторна робота виконується на ЕОМ в інтерактивному режимі.

1. Вивчити основні поняття та оператори, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму, яка просить користувача ввести початкове значення змінної X_1, коефіцієнти B, C та індекс n. Потім програма обчислює формули відповідно до варіанту і в циклі відображає результат на екрані монітора в такому форматі:

   X1 = , X2 = , … , Xn =

   Y1 = , Y2 = , … , Yn =

   F = .

   Після цього програма запитує користувача чи він хоче завершити програму і пропонує ввести символи ‘Y’ (‘y’) або ‘N’ (‘n’). Якщо користувач вводить символ ‘Y’ (‘y’) – програма завершується, а якщо користувач вводить символ N’ (‘n’), то програма переходить на початок, тобто знову пропонує ввести початкові дані, після чого ще раз видає результат обчислення формул.

№ варіанту	Завдання
1	$X_{i+1} = X_i + 2, \quad i = 1 \dots n;$ $Y_i = X_i \cdot B + 2 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
2	$X_{i+1} = X_i \cdot 2, \quad i = 1 \dots n;$ $Y_i = (X_i + 3 \cdot B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
3	$X_{i+1} = X_i + 1, \quad i = 1 \dots n;$ $Y_i = 2 \cdot X_i + B \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$

4	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = X_i \cdot B - 2 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
5	$X_{i+1} = X_i + 4, \quad i = 1 \ldots n;$ $Y_i = (X_i - 6B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
6	$X_{i+1} = X_i \cdot 3, \quad i = 1 \ldots n;$ $Y_i = 4 \cdot X_i - B - C;$ $F = \sum_{i=1}^{n} Y_i,$
7	$X_{i+1} = X_i + 5, \quad i = 1 \ldots n;$ $Y_i = X_i - B + 3 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
8	$X_{i+1} = X_i + 2, \quad i = 1 \ldots n;$ $Y_i = (3 \cdot X_i - B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
9	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = 4 \cdot X_i + B - C;$ $F = \sum_{i=1}^{n} Y_i,$
10	$X_{i+1} = X_i + 1, \quad i = 1 \ldots n;$ $Y_i = (6 \cdot X_i - C) \cdot B;$ $F = \sum_{i=1}^{n} Y_i,$
11	$X_{i+1} = X_i + 2, \quad i = 1 \ldots n;$ $Y_i = 5 \cdot X_i + 2 \cdot C - B;$ $F = \sum_{i=1}^{n} Y_i,$
12	$X_{i+1} = 2 \cdot X_i, \quad i = 1 \ldots n;$ $Y_i = 4 \cdot X_i + 2 \cdot B - C;$

	$F = \sum_{i=1}^{n} Y_i,$
13	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = (6 \cdot X_i + B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
14	$X_{i+1} = X_i + 4, \quad i = 1 \ldots n;$ $Y_i = 3 \cdot X_i - B + 2 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
15	$X_{i+1} = 3 \cdot X_i, \quad i = 1 \ldots n;$ $Y_i = (X_i + 3 \cdot B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
16	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = X_i \cdot B + 4 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
17	$X_{i+1} = X_i \cdot 2, \quad i = 1 \ldots n;$ $Y_i = (2 \cdot X_i + B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
18	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = (4 \cdot X_i + B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
19	$X_{i+1} = X_i + 1, \quad i = 1 \ldots n;$ $Y_i = (X_i \cdot B - 2) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
20	$X_{i+1} = 2 \cdot X_i, \quad i = 1 \ldots n;$ $Y_i = X_i + 4B - C;$ $F = \sum_{i=1}^{n} Y_i,$

21	$X_{i+1} = X_i + 2, \quad i = 1 \ldots n;$ $Y_i = 3 \cdot X_i + B + 2 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
22	$X_{i+1} = X_i + 3, \quad i = 1 \ldots n;$ $Y_i = 2 \cdot X_i - B + C;$ $F = \sum_{i=1}^{n} Y_i,$
23	$X_{i+1} = X_i + 1, \quad i = 1 \ldots n;$ $Y_i = 4 \cdot X_i - B + 4 \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
24	$X_{i+1} = 2 \cdot X_i + 1, \quad i = 1 \ldots n;$ $Y_i = (3 \cdot X_i + 2 \cdot B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
25	$X_{i+1} = 3 \cdot X_i, \quad i = 1 \ldots n;$ $Y_i = 2 \cdot X_i + 3 \cdot C + B;$ $F = \sum_{i=1}^{n} Y_i,$
26	$X_{i+1} = X_i + 1, \quad i = 1 \ldots n;$ $Y_i = (8 \cdot X_i + 2 \cdot C) \cdot B;$ $F = \sum_{i=1}^{n} Y_i,$
27	$X_{i+1} = 3 \cdot X_i + 1, \quad i = 1 \ldots n;$ $Y_i = 3 \cdot X_i + B - C;$ $F = \sum_{i=1}^{n} Y_i,$
28	$X_{i+1} = 2 \cdot X_i + 3, \quad i = 1 \ldots n;$ $Y_i = (5 \cdot X_i + B) \cdot C;$ $F = \sum_{i=1}^{n} Y_i,$
29	$X_{i+1} = X_i + 2, \quad i = 1 \ldots n;$ $Y_i = 3 \cdot (X_i - B) + 2 \cdot C;$

	$F = \sum_{i=1}^{n} Y_i,$
30	$X_{i+1} = 3 \cdot X_i + 1, \ \ i = 1 \ldots n;$ $Y_i = 4 \cdot X_i + 2 \cdot B - C;$ $F = \sum_{i=1}^{n} Y_i,$

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці результат обчислення формули, яка задана в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Які ви знаєте команди для організації циклу в мові програмування Асемблер?
2. Який регістр використовується для задання кількості ітерацій циклу?
3. Як організувати цикл в циклі?
4. Які команди умовного переходу можна використати для організації циклу?

ЛАБОРАТОРНА РОБОТА №7

ТЕМА: Використання масивів в Асемблері.

МЕТА: Навчитися працювати з масивами, засвоїти методи описання та ініціалізації масивів.

1. Теоретичні відомості

При роботі з масивами необхідно чітко уявляти собі, що всі елементи масиву розташовуються в пам'яті комп'ютера послідовно. Але таке розташування нічого не говорить про призначення і порядок використання цих елементів. І тільки лише програміст за допомогою складеного ним алгоритму обробки визначає, як потрібно трактувати цю послідовність байт, що складають масив.

Для того щоб локалізувати визначений елемент масиву, до його імені потрібно додати *індекс*. Тому визначаючи масив потрібно подбати і про визначення індексу. У мові Асемблер індекси масивів — це звичайні адреси, але з ними працюють особливим чином. Іншими словами, коли при програмуванні на асемблері ми говоримо про індекс, то скоріше маємо на увазі під цим не номер елементу в масиві, а деяку адресу. Наприклад, у програмі статично визначена послідовність даних:

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

Нехай ця послідовність чисел трактується як одномірний масив. Розмірність кожного елемента визначається директивою dw, тобто вона дорівнює 2 байти. Щоб одержати доступ до третього елемента, потрібно до адреси масиву додати 6. Нумерація елементів масиву в асемблері починається з нуля.

Тобто в цьому випадку мова, фактично, йде про 4-й елемент масиву — 3, але про це знає тільки програміст; мікропроцесору в даному випадку все рівно — йому потрібна лише адреса.

У загальному випадку для одержання адреси елемента в масиві необхідно початкову (базову) адресу масиву додати до добутку індексу (номер елемента мінус одиниця) цього елемента на розмір елемента масиву:

база + (індекс*розмір елемента)

Якщо послідовність однотипних елементів у пам'яті трактується як двомірний масив, розташований по рядках, то адреса елемента (i, j) обчислюється по формулі:

(база + кількість_елементів_у_рядку * розмір_елемента * i + j)

Тут i = 0...n–1 вказує номер рядка, а j = 0...m–1 вказує номер стовпця.

Наприклад, нехай є масив чисел (розміром у 1 байт) mas(i, j) з розмірністю 4 на 4 (i= 0...3, j = 0...3):

23	04	05	67
05	06	07	99
67	08	09	23
87	09	00	08

У пам'яті елементи цього масиву будуть розташовані в наступній послідовності:

23 04 05 67 05 06 07 99 67 08 09 23 87 09 00 08

Якщо ми хочемо трактувати цю послідовність як двомірний масив, приведений вище, і витягти, наприклад, елемент mas(2, 3) = 23, то провівши простий підрахунок, переконаємося в правильності наших міркувань:

Ефективна адреса mas(2, 3) = mas + 4 * 1 * 2 + 3 = mas + 11

1.1 Створення масивів

Приклад програми для створення масиву.

```
.MODEL SMALL
.STACK 256
.DATA
    Data1 DB 48h, 45h, 4Ch, 4Ch, 4Fh, '$'
.CODE
 Start PROC
    mov ax, @data          ; встановлення в ds адреси
    mov ds, ax             ; сегменту даних
    mov dx, offset Data1 ; вказівник на масив символів
    mov ah, 09h            ; вивести рядок
    int 21h
 Exit:
    mov ah, 04Ch           ; функція виходу з програми
    int 21h                ; виклик переривання для зупинки програми
Start ENDP
 END Start
```

Отже, через кому визначаються значення елементів масиву як шістнадцятковим значенням так і наприклад ASCII кодом для символу '$'. Потім виводиться цей масив звичайною функцією виведення рядка.

1.2 Виділення місця під масив

Якщо потрібно визначити масив скажімо з 50 елементів можна скористатись директивою DUP щоб створити масив зазначеного розміру і заповнити його будь-яким значенням.

```
.MODEL small
.STACK 4096
```

```
.DATA
    Data1 DB 50 DUP (1)
.CODE
main PROC
    mov ax, @data         ; встановлення в ds адреси
    mov ds, ax            ; сегменту даних
    mov dx, offset Data1 ; вказівник на масив символів
 Exit:
    mov ah, 40h                 ;
    int 21h                     ;
    mov ah, 04Ch          ; функція виходу з програми
    int 21h               ; виклик переривання для зупинки програми
main ENDP
END main
```

1.3 Доступ до елемента масиву

Для того, щоб навчитися міняти елемент масиву можна використати операцію отримання індексу в масиві.

```
.MODEL SMALL
.STACK 256
.DATA
    Data1 DB 10h DUP (38h)
.CODE
Start PROC
    mov ax, @data         ; встановлення в ds адреси
    mov ds, ax            ; сегменту даних
    mov dx, offset Data1 ; вказівник на масив символів
    mov Data1 [15], '$'
    mov Data1 [3], '7'
    mov dx, offset Data1
    mov ah, 09h
    int 21h
Exit:
    mov ah, 04Ch          ; функція виходу з програми
```

```
    int 21h                ; виклик переривання для зупинки програми
Start ENDP
END Start
```

2. Хід роботи

Лабораторна робота виконується на ЕОМ в інтерактивному режимі.

1. Вивчити основні поняття та оператори, які використовуються в програмі, поданій в теоретичних відомостях.
2. Ввімкнути ЕОМ.
3. Написати програму, яка просить користувача ввести число – кількість елементів масиву. Потім в циклі просить користувача ввести з клавіатури числа (як додатні так і від'ємні) – елементи масиву, кожен з яких повинен мати розмір слова (2 байти). Після цього виводить на екран в рядок через пробіл усі елементи початкового масиву. В наступному рядку виводить ті дані, які вказані відповідно до варіанту. Потім програма запитує користувача чи він хоче завершити програму і пропонує ввести символи 'Y' ('y') або 'N' ('n'). Якщо користувач вводить символ 'Y' ('y') – програма завершується, а якщо користувач вводить символ N' ('n'), то програма переходить на початок виконання.

№ варіанту	**Завдання**
1	Вивести на екран суму усіх чисел масиву крім максимального і мінімального.
2	Вивести на екран добуток максимального і мінімального числа масиву.
3	Вивести на екран суму усіх чисел масиву, які лежать в діапазоні від -20 до 30
4	Вивести на екран добуток усіх від'ємних чисел масиву
5	Вивести на екран суму усіх парних чисел масиву, помножену на найменше число масиву
6	Вивести на екран добуток усіх чисел масиву, які лежать в діапазоні від -40 до 25
7	Вивести на екран суму усіх додатних чисел масиву, помножену на найменше число масиву
8	Вивести на екран добуток усіх чисел масиву, які лежать в діапазоні від -30 до 15
9	Вивести на екран суму усіх чисел масиву, кратних числу 3
10	Вивести на екран суму усіх чисел масиву, які менші за -10 та більші за 100
11	Вивести на екран добуток усіх чисел масиву, кратних числу 5
12	Вивести на екран суму усіх чисел масиву, які лежать в діапазоні від -55 до 105
13	Вивести на екран добуток усіх непарних чисел масиву, поділених на найбільше число масиву
14	Вивести на екран середнє арифметичне від усіх чисел масиву крім

	мінімального
15	Вивести на екран добуток усіх додатних чисел масиву крім найбільшого

4. Проасемблювати програму.
5. Перевірити програму на наявність помилок, у випадку їх наявності — виправити і перейти до кроку 3.
6. Скомпонувати програму.
7. Якщо в результаті компонування отриманий виконуваний файл, то виконати програму.
8. Результатом виконання лабораторної роботи повинна бути програма, яка виводить в командній стрічці результат обчислення формули, яка задана в п. 3.

3. Зміст звіту

1. Титульна сторінка, оформлена відповідно до зразка.
2. Тема роботи
2. Мета роботи.
3. Завдання до виконання лабораторної роботи.
7. Текст (лістинг) програми.
8. Скріншот екрану з результатами асемблювання, компонування та виконання програми.
9. Висновки.

4. Контрольні запитання

1. Як описати масив даних у програмі?
2. Як ініціалізувати масив і задати початкові значення його елементів?
3. Як організувати доступ до елементів масиву?
4. Як організувати виконання типових операцій з масивами?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рисований О.М. Системне програмування: Підручник. – Х.: НТУ “ХПІ”, 2010. – 912 с.
2. Рысованый А.Н. Системное программирование, Ч.1. Программирование в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 108 с.
3. Рисований О.М. Системне програмування : навч. посіб. Для студентів напрямку «Комп’ютерна інженерія» вищих навчальних закладів у 4 ч. Ч. І. Основи асемблера та його використання. / О. М. Рисований. – Харків : «Слово», 2015. – 336 с.
4. Рисований О.М. Системне програмування : навч. посіб. Для студентів напрямку «Комп’ютерна інженерія» вищих навчальних закладів у 4 ч. Ч.2. Розширені можливості програмування на асемблері. / О. М. Рисований. – Харків : «Слово», 2015. – 224 с.
5. Аблязов Р. З. Программирование на ассемблере на платформе x86-64. - М.: ДМ К Пресс, 2011. – 304 с.
6. Смоленцев М.Ю. Программирование на языке Ассемблера для 32/64-разрядных микропроцессоров семейства 80x86: Учебное пособие в 3-х частях. Часть 3. – Иркутск: ИрГУПС, 2009. – 180 с.
7. Магда Ю. С. Ассемблер для процессоров Intel Pentium / Ю. С. Магда. – СПб.: Питер, 2006. – 410 с.
8. Пирогов В. Ю. Ассемблер и дизассемблирование / В. Ю. Пирогов. – СПб.: БХВ-Петербург, 2006. – 464 с.
9. Юров В. И. Assembler. Практикум. 2-е издание / В. И. Юров. – СПб.: Питер, 2006. – 399 с.
10. Магда Ю. С. Использование ассемблера для оптимизации программ на C++ / Ю. С. Магда. СПб.: БХВ-Петербург, 2004. – 496 с.
11. Паламар А. Комп'ютерна система для моніторингу параметрів джерел безперебійного живлення на основі технології Internet of Things. Матеріали IV Міжнародної науково-технічної конференції "Теоретичні та прикладні аспекти радіотехніки, приладобудування і комп’ютерних технологій", 20-21 червня 2019 року: збірник тез доповідей. Тернопіль: ФОП Паляниця В. А., 2019. с. 208-209.
12. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп’ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.
13. Паламар А.М., Пастернак Ю.В., Паламар Я.М. Двох-процесорна інформаційно-вимірювальна система керування пристроєм безперебійного електроживлення. Матеріали ІІІ Міжнародної науково-технічної конференції молодих учених та студентів "Актуальні задачі сучасних технологій", 19-20 листопада 2014 р. Тернопіль: ТНТУ, 2014. С. 211-212.
14. Дистанційний курс “Системне програмування”, доступний за адресою https://dl.tntu.edu.ua/bounce.php?course=1989.