МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Електронне видання

# Архітектура комп’ютера

Методичні вказівки до виконання лабораторних робіт

для студентів денної і заочної форми навчання

СПЕЦІАЛЬНОСТІ

121 – Інженерія програмного забезпечення

Затверджено методичною радою ФІОТ

Київ КПІ ім. Ігоря Сікорського

2020

Архітектура комп’ютера : Метод. вказівки до виконання лаб. робіт для студентів усіх форм навчання спеціальності 121 – Інженерія програмного забезпечення / Уклад.:Павло Катін, – К.: КПІ ім. Ігоря Сікорського, 2020. – с.105

*Гриф надано Методичною радою ФІОТ*

Електронне видання

# Архітектура комп’ютера

Методичні вказівки до виконання лабораторних робіт
для студентів

спеціальності 121 – Інженерія програмного забезпечення, спеціалізація «Інтегровані інформаційні системи»

Укладач: Катін Павло Юрійович, канд. техн. наук, доц. каф. АУТС, ФІОТ

Рецензенти:

За редакцією укладачів

ЗМІСТ

# ВСТУП

Методичні вказівки до виконання лабораторних робіт з дисципліни архітектура комп’ютера формують у студентів потрібні знання, навички та вміння відповідно до вимог освітньої програми підготовки фахівця за спеціальністю 121 – Інженерія програмного забезпечення.

Надають можливість самостійного набуття знань і навичок у ході практичної роботи з програмування, налагодження і підтримки програмного забезпечення.

# ЛАБОРАТОРНА РОБОТА №1

# ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМ У АРХИТЕКТУРІ IA32 (X86)

# REAL ADRESS MODE

Мета лабораторної роботи полягає у набутті твердих навичок і знань технологічної основи розробки ПЗ на Асемблері, у ході якої застосовуються знання архітектури комп'ютерів.

Програма роботи складається з наступних кроків:

- підготувати персональний комп'ютер до розробки ПЗ на Асемблері Tasm;
- виконати повний цикл розробки, тестування і налагодження програмного забезпечення;
- зберігати отриману програму, зробити висновки щодо необхідності знань архітектури комп'ютера у ході розробки ПЗ.

## *Завдання для ЛР 1*

1. Виконати повний технологічний цикл створення програми на Асемблері.
2. Доопрацювати вихідний код програми і вивести на консоль призвища всіх студентів робочої бригади.
3. Описати всі архітектурні елементи x86, що задіяні у програмі з використанням налагоджувача.

## *Теоретичні відомості для ЛР 1*

Звіт має містити типовий титульний аркуш для лабораторних робіт, з підписом студента, результати дослідження і висновки. У результатах дослідження необхідно додати вихідний код з коментарями, при відсутності коментарів оцінка знижується. Робота виконується виключно у діалекті асемблеру TASM. Для підтвердження результатів необхідно запустити

налагоджувач Turbo Debugger і зробити фотографії екранів, з виділенням результатів.

Крім того необхідною умовою захисту лабораторної роботи є:

тверді навички розробки вихідного коду, його асемблювання, лінкування;

вміння користуватися Turbo Debugger, здійснювати покроковий запуск програми, відкривати вікно дампу пам'яті, переходити на необхідну адресу;

визначати адресу з використанням дампу пам'яті будь якого байту у сегменті стека або сегменті коду;

пояснити зміни у пам'яті програми під час кожного кроку, пояснити призначення кожної інструкції.

1. Розробка вихідного коду, асемблювання, лікування першого проекту.

Перша тестова програма буде виводить на екран повідомлення "Hello World!", зупинятися і очікувати натискання клавіші користувачем і на цьому її функціональність завершується. Розкриємо етапи створення програми.

Етап 1. Створення вихідного коду. Приклад вихідного коду показаний у фрагменті коду. Для розробки необхідно, користуючись одним з текстових редакторів, створити файл hello.asm. Скопіювати фрагмент коду до файлу hello.asm і зберегти його у робочому каталозі.

Етап 2. Асемблювання вихідного коду. Запускаємо консоль windows і, використовуючи команду cd, переходимо до робочого каталогу. За допомогою програми TASM.EXE переводимо hello.asm у машинний код hello.obj. Для асемблювання вихідного коду у командному рядку виконується команда:

*tasm.exe /l /zi hello.asm.*

Прапор */l* створює у робочому каталозі файл лістингу *hello.lst,* що містить адреси, машинні коди, текст, коментарі. Це інформація для налагодження.

Прапор */x* створює файл карти пам'яті. Наприклад при виконанні команди: *tlink /x hello.obj* створюється файл карти пам'яті *hello.map* (Linked Address Map).

Прапор /zi  додає до файлу hello.obj коментарі, назви змінних, тощо. Це необхідно для спрощення процесу налагодження. У випадку відсутності помилок у hello.asm у робочому каталозі з'являється об'єктний файл, *hello.obj*.

Етап 3. Компонування або лінкування. На цьому етапі завершується визначення адресних посилань і об'єднання, якщо потрібно, декількох об'єктних файлів і бібліотек. В командному рядку виконується команда:

*tlink.exe  /v  hello.obj.*

Прапор */v*  додає до файлу *hello.exe* інформацію для спрощення процесу налагодження (коментарі, назви змінних, тощо). Якщо етап 3 є успішним, створюється файл – *hello.exe.* Його можна запустити, виконуючи команду *hello.exe* і отримати у консолі *Hello world.*

Крім того для проведення експерименту необхідно проводити покрокове виконання програми, контролювати зміст змінних, регістрів і дампу пам'яті. Для цього використовується Turbo Debugger, що надається у комплекті з Асемблером. Для трасування програми необхідно виконати у командному рядку, находячись у робочому каталозі наступну команду:

*td.exe  hello.exe.*

Фрагмент коду, що може бути  взятий за основу представлений далі.

```
TITLE ЛР_1
;-------------------------------------------------------------------------
;ЛР  №1.1
;-------------------------------------------------------------------------
; Програмування 3. Системне програмування
; Завдання:            Основи розробки і налагодження
; ВУЗ:               КНУУ "КПІ"
; Факультет:    ФІОТ
; Курс:              2
; Група:       __ __ __
;-------------------------------------------------------------------------
; Автор:        __ __ __
; Дата:         __/__/__
;--------------------------------
                                ;I.ЗАГОЛОВОК ПРОГРАМИ
IDEAL                   ; Директива - тип Асемблера tasm
MODEL small             ; Директива - тип моделі пам'яті
STACK 256               ; Директива - розмір стеку
                                ;II.МАКРОСИ
                                ;III.ПОЧАТОК СЕГ arr_rnd1МЕНТУ ДАНИХ
```

```
DATASEG
exCode db 0
message db "Hello world!",10,13,'$';Рядок символів для виводу на екран
                                ;VI. ПОЧАТОК СЕГМЕНТУ КОДУ
CODESEG
Start:
;---------------------------------- 1. Ініціалізація DS и ES-------------------------------------
        mov ax,@data; @data ідентифікатор, що створюються директивою model
        mov ds, ax      ; Завантаження початку сегменту даних в регістр ds
        mov es, ax      ; Завантаження початку сегменту даних в регістр es
;----------------------------------2. Операція виводу на консоль-------------------------------
                        ; Пересилання адреси рядка символів message в регістр dx
        mov dx, offset message
                        ; Завантаження числа 09h до регістру ah
                        ; (Функція DOS 9h - команда виводу на консоль рядка)
        mov ah,09h
        int 21h         ; Виклик функції DOS 9h
;----------------------------------3. Операція зупинки програми, очікування натискання клавіш-----
        mov ah,01h
                        ; Завантаження числа 01h до регістру ah
                        ; (Функція DOS 1h - команда очікування натискання клавіші...)
        int 21h         ; Виклик функції DOS 1h
                        ; Завантаження числа 4ch до регістру ah
                        ; (Функція DOS 4ch - виходу з програми)
;----------------------------------4. Вихід з програми-----------------------------------------
        mov ah,4ch
        mov al,[exCode]         ; отримання коду виходу
        int 21h         ; виклик функції DOS 4ch
        end Start
```

```
;-------------------------------------------------------------------------------
;ЛР  №1.1(А)
;-------------------------------------------------------------------------------
; Програмування 3. Системне програмування
; Завдання:             Основи розробки і налагодження переривання BIOS
; ВУЗ:                КНУУ "КПІ"
; Факультет:    ФІОТ
; Курс:               2
; Група:       __ __ __
;-------------------------------------------------------------------------------
; Автор:        __ __ __
; Дата:         __/__/__
;---------------------------------
                                ;I.ЗАГОЛОВОК ПРОГРАМИ
IDEAL                   ; Директива - тип Асемблера tasm
MODEL small             ; Директива - тип моделі пам’яті
STACK 256               ; Директива - розмір стеку
                                ;II.МАКРОСИ
                                ;III.ПОЧАТОК СЕГ arr_rnd1МЕНТУ ДАНИХ
DATASEG
```

```
exCode db 0
message db "Hello world!",10,13,'$';Рядок символів для виводу на екран
                                   ;VI. ПОЧАТОК СЕГМЕНТУ КОДУ
CODESEG
Start:
;---------------------------------- 1. Ініціалізація DS и ES-------------------------------------
        mov ax,@data; @data ідентифікатор, що створюються директивою model
        mov ds, ax      ; Завантаження початку сегменту даних в регістр ds
        mov es, ax      ; Завантаження початку сегменту даних в регістр es
;----------------------------------2. Визанчення відеорежиму-------------------------------
        mov ax, 0003h
        int 10h                 ;Очищення екрану і встановлення курсору 0. 0
        mov dh, 0       ;Координата курсору
        mov dl, 0       ;Координатк курсору
;----------------------------------3. Операція виводу на консоль-------------------------------
                        ; Завантаження числа 13h до регістру ah
                        ; (Функція BIOS 13h - команда виводу на консоль рядка)
        mov ah,13h
        mov al, 0001h           ;Режим виводу, курсор до кінця рядка
        mov cx, 15      ;Довжина рядка, тільки кількість символів
        mov bl, 12      ;Атрибут, якщо рядок містить тільки символи
        mov dh, 0       ;Координата курсору
        mov dl, 0       ;Координатк курсору
        mov dx, offset message ; Пересилання адреси рядка message в регістр dx
        int 10h         ; Виклик функції DOS 9h

;----------------------------------3. Операція зупинки програми, очікування натискання клавіш-----
        mov ah,01h
                        ; Завантаження числа 01h до регістру ah
                        ; (Функція DOS 1h - команда очікування натискання клавіші...)
        int 21h         ; Виклик функції DOS 1h
                        ; Завантаження числа 4ch до регістру ah
                        ; (Функція DOS 4ch - виходу з програми)
;----------------------------------4. Вихід з програми-----------------------------------------
        mov ah,4ch
        mov al,[exCode]         ; отримання коду виходу
        int 21h         ; виклик функції DOS 4ch
        end Start
```

Будь яка програма на Асемблері складається з окремих частин. Ці частини програми розміщуються у окремих ділянках памяті, що називаються сегментами. Механізми адресації, що пов’язані з цім визначенням будуть більш докладно описані далі.

Як можна побачити з прикладу рис.2, TD відображає вікно коду, дамп пам’яті, регістри і вікно стеку. Ліворуч вікна коду і дампу пам’яті знаходяться логічні адреси, наприклад cs:0000, ds:0000.

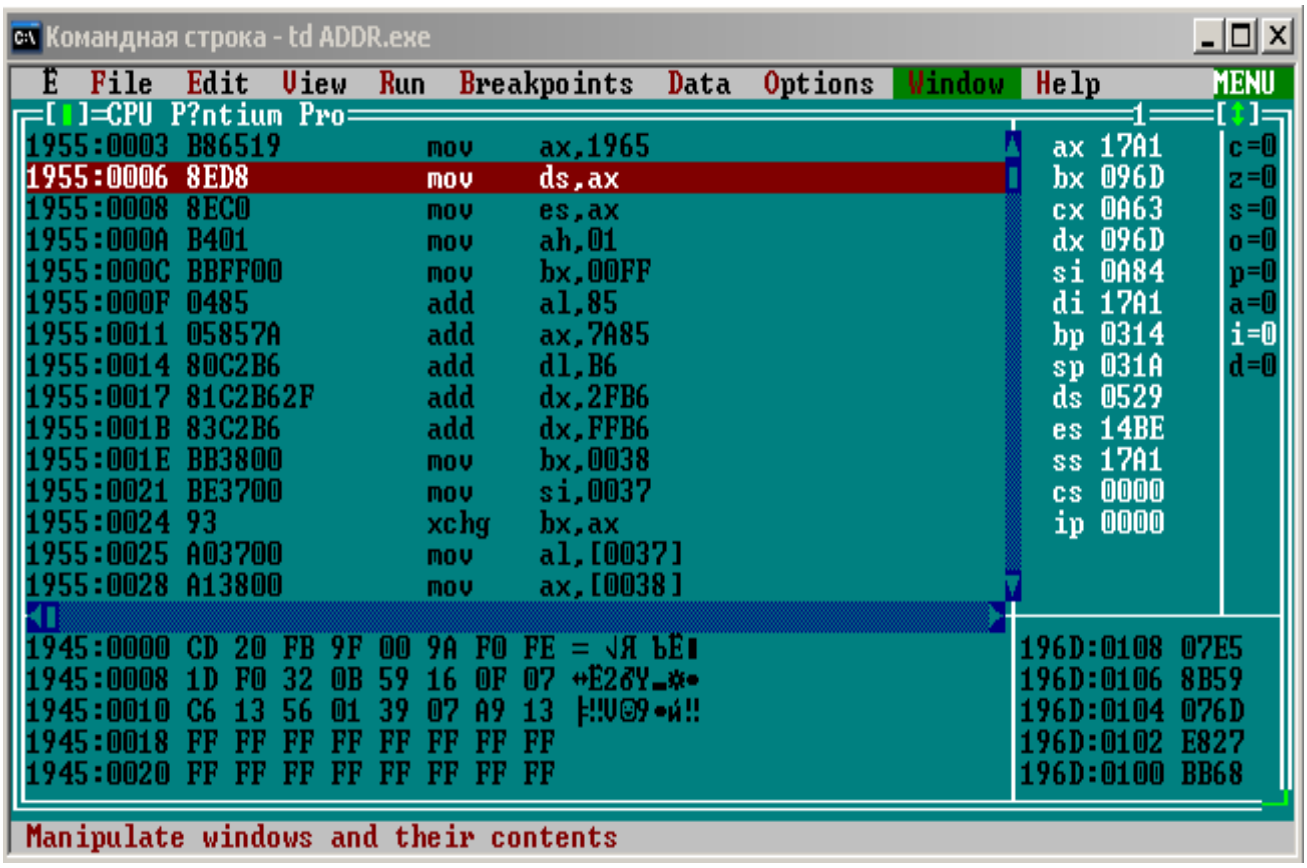


Рисунок 1- Вікно програми Turbo Debugger

Вихідний код можна умовно розподілити на 4 базових частини. Перша частина це заголовок програми, що показаний нижче.

```
                                ;I.ЗАГОЛОВОК ПРОГРАМИ
IDEAL                           ; Директива - тип Асемблера tasm
MODEL small                     ; Директива - тип моделі пам’яті
STACK 256                ; Директива - розмір стеку
```

Перша директива заголовку визначає тип Асемблеру. Асемблер TASM підтримує два діалекту: TASM і діалект MASM. Директива IDEAL в нашому прикладі визначає діалект TASM. Опишемо більш докладніше сегмент даних прикладу коду 1.1.

```
;III.ПОЧАТОК СЕГМЕНТУ ДАНИХ
DATASEG
exCode db 0
message db "Hello world!",10,13,'$';Рядок символів для виводу на екран
```

Він призначений для збереження змінних, масивів і інших даних програми. В цьому сегменті визначаються глобальні змінні. Наприклад, у якості змінної використовується exCode. Ця змінна має тип (розмір) 1 байт та має початкове

значення 0. Друга змінна є рядком символів, що будуть виводитися на консоль. Фактично це покажчик на нульовий елемент масиву символів.

*Програмна модель МПС на базі процесорів Intel 8086*. Для розуміння роботи команд асемблера необхідно чітко представляти, як виконується адресація даних, які регістри процесора і у який спосіб можуть використовуватися при виконанні інструкцій. Для цього необхідно розглянути базову програмну модель процесорів Intel. До неї звичайно входять:

- 8 регістрів загального призначення, що служать для зберігання даних і покажчиків;
- регістри сегментів зберігають адреси початку сегментів програми;
- регістр прапорів FLAGS, що дозволяє управляти виконання програми й станом процесора;
- регістр-покажчик IP команди процесора;
- система команд (інструкцій) процесора;
- режими адресації даних у командах процесора.

Розглянемо більш докладно, що таке регістр. Регістр це пам'ять, що убудована до мікропроцесора і має найбільш швидку дію. У архітектурі процесорів Intel 8086 цей регістр має місткість 2 байти або одне машинне слово (16 біт). За кожним з регістрів закріплені власні імена, з використанням яких регістри адресуються у Асемблері.

Регістри загального призначення AX, BX, CX, DX мають можливість адресувати старший і молодший байт інформації і мають для них окремі назви. Розкриємо особливості цих регістрів більш докладно:

AX (AH, AL) акумулятор, що призначений для всіх операцій вводу-виводу, більш ніж інші регістри загального призначення спеціалізований для виконання арифметичних операцій, деякі арифметичні операції виконуються виключно з його використанням;

BX (BH, BL) базовий регістр, рівноцінний іншим регістрам загального призначення, має особливість, яка характерна тільки для нього - може зберігати адресу покажчика на масив під час базової адресації;

CX (CH, CL) регістр лічильник, рівноцінний іншим регістрам загального призначення, має особливість, яка характерна тільки для нього - забезпечує автоматичний декремент змінної під час циклічних операцій;

DX (DH, DL) регістр даних, рівноцінний іншим регістрам загального призначення, має особливість, яка характерна тільки для нього - забезпечує ввід і вивід інформації на зовнішні пристрої (операції IN, OUT).

До регістрів загального призначення також відносять регістри покажчики і індексні регістри. Вони можуть бути використані у арифметичних і логічних операціях. На відміну від вищеописаних вони не можуть окремо адресувати молодший і старший байт слова. Основне призначення регістрів покажчиків це забезпечення доступу до інформації у сегменті стеку, а саме:

регістр SP покажчик на стек, дає можливість адресувати вершину стеку, разом з регістром SS визначають адресу операнду у вершині стеку;

регістр BP покажчик бази стеку, дає можливість адресувати будь-який операнд, що знаходиться у стеку, довільним образом. Отже три регістра SP, SS, BP повністю керують стеком і не бажано їх використовувати з іншою метою.
Основне призначення індексних регістрів виконання операції пересилання символьних рядків, а саме: регістр SI індекс джерела рядку символів, регістр DI індекс призначення рядку символів. Ці регістри також використовуються у індексної адресації. У програмах для налагодження прийнято позначати зміст регістрів з використанням цифр системи числення з основою 16.

Вони не призначені для виконання математичних і інших операцій, їх основне призначення забезпечення адресації в архітектурі МПС. Кожний з сегментних регістрів може забезпечити адресацію 64К ОЗП. Вони не можуть бути адресовані прямо, доступ до них здійснюється через РЗП. Як було сказано раніше програми у Real Address Mode складається з частин, що

називаються сегментами. Для адресації кожного з сегментів призначені ці регістри.

Регістр CS, сегмента коду, що містить початкову адресу сегменту коду, зміст цього сегменту, помножений на 16 із додаванням величини зміщення у регістрі IP, визначає адресу поточної команди, що буде виконуватися.

Регістр DS, сегмента даних, містить початкову адресу сегменту даних, у нашому прикладі вимагає ініціалізації і дає можливість адресувати будь яку змінну, об'єкт або символьний рядок, що знаходиться у сегменті даних.

Регістр SS, регістр сегмента стеку містить початкову адресу сегменту стеку, зміст цього сегменту, помножений на 16 із додаванням величини зміщення у регістрі SP, визначає адресу слова у вершині стеку.

Регістр ES, додатковий регістр, може містити тимчасові значення адрес у різних способах адресації, потребує ініціалізації. До регістра покажчика IP у звичайному режимі роботи у розробника немає доступу до цього регістру.

Регістр прапорів FLAGS, відображає і керує поточним станом архітектурі МПС. Перечислимо прапори і їх призначення: O переповнення, встановлюється у 1 при переповнені старшого біта операнду, D визначає правий або лівий напрямок пересилання строкових рядків у пам'яті, I встановлений у 1, коли дозволені зовнішні переривання, T встановлений у 1 у покровому режимі під час налагодження програми, S містить знак результату під час арифметичних операцій, Z ознака результату арифметичної операції нульовий встановлюється в 1, A зовнішній перенос, використовується для спеціалізованих арифметичних операцій, P контроль парності встановлюється у 1 при парності результату у молодшої частині слова, C містить перенос з старшого біту після арифметичних операцій, або останній біт при циклічних зсувах.

Перелічимо ще переривання DOS для виводу на консоль.

**Функція DOS 06h**: записати символ на консоль без перевірки на Ctrl-Break

Вхідні параметри: AH = 06h

DL = ASCII-код символа

Вихідні параметри: AL = код записаного символа

**Функція DOS 09h**: записати символ на консоль без перевірки на Ctrl-Break

Вхідні параметри: AH = 09h

DS:DX = адреса рядка, що закінчується символом $(24h)

Вихідні параметри: AL = код крайнього символа

Після цей команди, МПС виводить на консоль рядок символів, початкова адреса якого записана у DX. У аналогічний спосіб працюють всі переривання.

```
;---------------------------------3. Операція зупинки програми, очікування натискання клавіш-----
        mov ah,01h
                        ; Завантаження числа 01h до регістру ah
                        ; (Функція DOS 1h - команда очікування натискання клавіші...)
        int 21h         ; Виклик функції DOS 1h
```

У ділянці коду, що показана вище, здійснюється операція затримки консолі до першого натискання клавіші. Регістр AH призначений для визначення типу операції очікування. У нашому випадку до AH треба занести цифру 01 (команда mov ah,01h). Інших операндів нам не потрібно, тому відразу викликається переривання DOS int 21h. Після цього робота програми зупиняється і програма очікує натискання на клавішу користувачем, після натискання програма продовжує роботу.

```
;---------------------------------4. Вихід з програми-----------------------------------------
                        ; Завантаження числа 4ch до регістру ah
                        ; (Функція DOS 4ch - виходу з програми)

        mov ah,4ch
        mov al,[exCode]         ; отримання коду виходу
        int 21h         ; виклик функції DOS 4ch
        end Start
```

У ділянці коду, що показана вище, здійснюється операція виходу з програми з використанням функції DOS. Для цього задіяні два регістра. Як було показано у попередніх прикладах, регістр AH призначений для визначення типу операції виходу. У нашому випадку до AH треба занести число 4ch (команда mov ah,4ch). Другий регістр AL призначений для збереження коду виходу (команда mov al, [exCode]). Далі викликається переривання DOS int 21h. Після цього робота програми завершується.

Для виводу на консоль можна скористатися іншим видом переривань. Переривань BIOS існує багато видів.

Перелічімо ще переривання BIOS для виводу на консоль.

**Переривання** BIOS **10 h (**00h**)**: Встановити відеорежим

Вхідні параметри: AH = 00h

AL = номер режиму у молодших 7 бітах

Вихідні параметри:

**Переривання** BIOS **10h (**02h**)**: Управління положенням курсору

Вхідні параметри: AH = 02h Управління положенням курсору

BH = Номер сторінки

DH= рядок

DL = стовпчик

Вихідні параметри:

**Переривання** BIOS **10h (**AH = 13h**)**: Вивід рядку із заданими параметрами

Вхідні параметри: AH = 13h

AL = режим виводу, біт 0 курсор до кінця, біт 1 наявність атрибутів, кожний символ містить код ASCII(1 байт) і атрибут (2 байт)

CX = довжина рядка (тільки число символів)

BL = атрибут для всього рядка, якщо рядок містить символи

DH, DL = рядок і стовпчик, координати початку виводу

ES:BP = адреса рядка, що виводиться у памяті

Вихідні параметри:

*Індивідуальні завдання*

Вивести на консоль імена студентів робочих груп. Кількість разів для для виводу відповідає номеру групи.

*Перелік питань для підготовки до лабораторної роботи 1*

1. Перелічіть регістри загального призначення і сегментні регістри архітектурі МПС Intel 8086. Розкрийте поняття розрядності регістрів, призначення, особливості регістрів.

2. Як здійснюється сегментація пам’яті архітектурі МПС на базі Intel 8086, які бувають сегменти. Який може бути максимальний і мінімальний розмір сегменту?

3. Команда  mov у архітектурі Intel 8086. Дайте поняття операнду. Які операнди команди mov застосовуються? Поясніть на прикладі коду.

# ЛАБОРАТОРНА РОБОТА №2 ПРЯМИЙ ДОСТУП ДО ВІДЕОПАМЯТІ АРХІТЕКТУРІ IA-32 (X86) У REAL ADRESS MODE

Мета лабораторної роботи полягає у набутті впевнених знань і навичок з розробки ПЗ на Асемблері для управління відеопам'яттю з урахуванням знань архітектури IA-32 у real address mode.

*Програма роботи складається з наступних кроків:*

- вивчити відеопам'ять архітектури IA5-32 у real address mode;
- виконати повний цикл розробки, тестування і налагодження програмного забезпечення;
- зберегти отриману програму, зробити висновки щодо необхідності знань архітектури комп'ютера у ході розробки ПЗ.

*Теоретичні відомості для ЛР 2*

Необхідною умовою захисту лабораторної роботи є: тверді навички розробки вихідного коду, його асемблювання, лінкування ; вміння користуватися Turbo Debugger, здійснювати покроковий запуск програми, відкривати вікно дампу пам'яті, переходити на необхідну адресу; визначати адресу з використанням дампу пам'яті будь якого байту у сегменті стека або сегменті коду; пояснити зміни у пам'яті програми під час кожного кроку, пояснити призначення кожної інструкції. В режим real address mode архітектурі МПС переходить під час завантаження операційної системи. В цьому режимі архітектурі МПС може адресувати 1Мбайт ОЗП. Повний обсяг ОЗП комп'ютер може адресувати тільки у protected mode, у захищеному режимі роботи процесора. Опис розподілу памяті у архітектурі 8086 Сучасні компютери обладнуються опреративною паміттю обсягом від 4 і більше Гбайт. Компютери архітектури IA32 працюють у декількох режимах. В режим real address mode МПС переходить під час завантаження операційної системи. В цьому режимі МПС може адресувати 1Мбайт ОЗП.

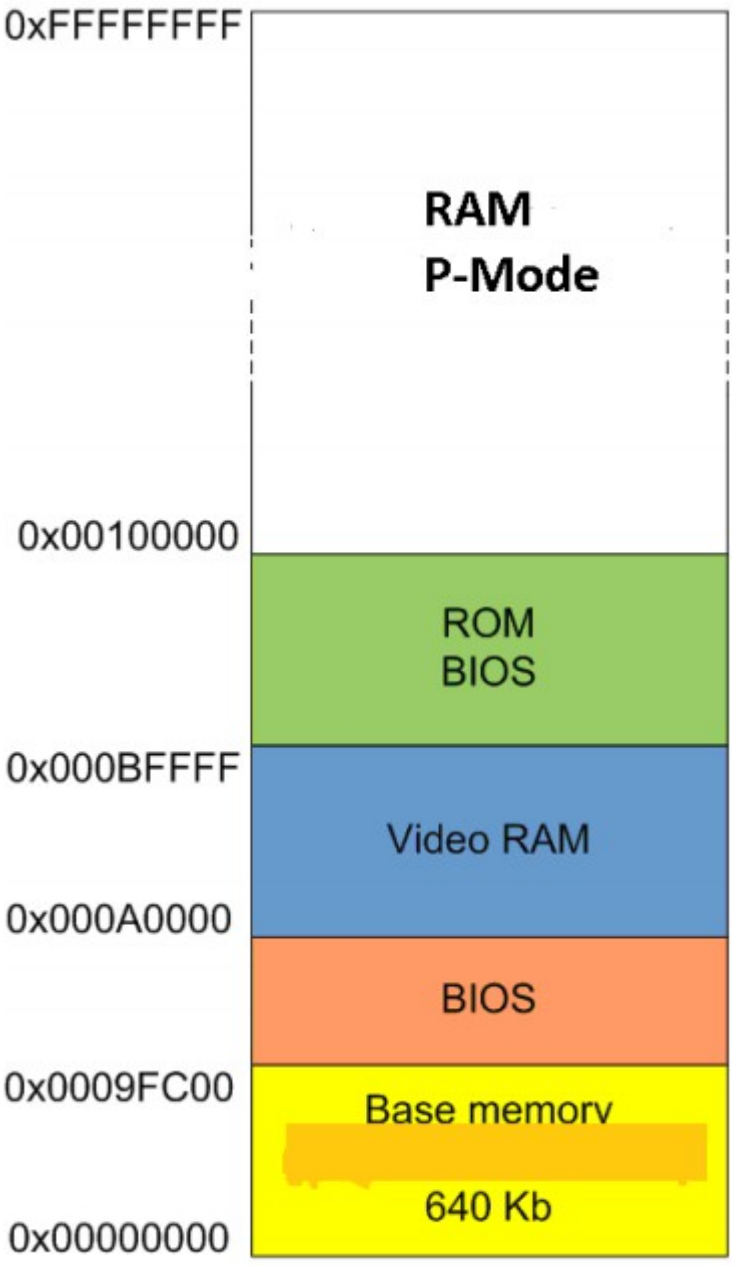


Рисунок 2 - Розподіл ОЗП у режимі real address mode

Повний обсяг ОЗП компютер може адресувати тільки у protected mode, у захищеному режимі роботи процесора. Необхідно розділити два терміна: адресний простір і оперативна пам'ять (ОЗП). Адресний простір це набір адрес, іноді абстрактних, що може адресувати процесор. Вони не обовязково відповідають реальному фізичному ОЗП. Типова схема адресного простору показана на рисунку для real address mode. Перші 640 Кбайт адресного

простору фізичної памяті (від 00000h до 9FFFFh) відводять під основну оперативну пам'ять, що називають стандартною або звичайною. Цьому адресному простору відповідає зміст сегментних адрес (від 0000h до 9FFFh). На початок цієї памяті заванатажуть таблиці и програми. На самому початку ОЗП, на першому килобайті реалізовані вектори переривань, 256 векторів по 4 байта кожний. Вони забезпечують роботу апаратної частини МПС. Стандартні вектори заповнюються автоматично, інші розробник може додати або змінити. За векторами переривань розташована так звана ділянка даних BIOS, починая з сегментної адреси 40h. До функції BIOS входить тестування вузлів МПС, завантаження ОС і управління штатними пристроями МПС під час завантаження (клавиатура, дисплей, таймер тощо). Після векторів переривань і BIOS залишкі стандартного ОЗП призаначються для завантаження прикладних і системних програм. Описання і функції відеопамяті. Дуже важливо розділяти два терміну: адресний простір і оперативна пам'ять (ОЗП). ОЗП це реально існуючий набір інстальованої оперативної пам’яті у архітектурі МПС. Адресний простір це набір адрес, іноді абстрактних, що може адресувати процесор. Вони не обов’язково відповідають реальному фізичному ОЗП. Особливо це актуально у protected mode. Для формування псевдографіки на екрані у real address mode архітектурі МПС IA-32 використовується відеопам’ять. Запис певних значень до відеопам’яті призводить до відображення на екрані відповідних символів і зафарблення екрану. Після звичайної пам’яті починається графічна відеопам’ять. Вона призначена для відображення інформації під час початкової роботи ПК, управління BIOS, і налаштування системи. Графічна відеопам’ять актуальна до закінчення завантаження драйверів відео і виходу у p-mode. Відеопам’ять починається з фізичної адреси A0000h і закінчується B0000h. Графічна пам'ять займає теоретично 64 Кбайт адресного простору, до адреси AFFFFh. Текстова відеопам’ять розташована на певної відстані від графічної і займає 32 Кбайта, починаючи з адреси B8000h. Текстова пам'ять включає до себе 8 відео сторінок

і займає у адресному просторі 32 Кбайту. Початок іде від відео сторінки з адресою відеопам’яті B800h.

Кожна сторінка займає 4 Кбайт, отже:

0 сторінка B800h,

1 сторінка B900h,

2 сторінка BA00h, тощо. При включенні комп’ютера видимої стає відео сторінка 0. Заміна старинок здійснюється викликом функції 05h переривання 10h BIOS. Будь який код, що записуєтеся до відеопам’яті, відразу відображається на екрані у вигляді зафарбленого символу на певному знакомісці. Кожний символ займає у відео пам’яті 2 байт. Молодші (парні) байти всіх полів відводять під коди ASCII. Старші (непарні) під атрибути відповідного знаку. Отже для опису символу потрібне слово. Таким чином кожному знако місту відповідає два байт. Розмір екрану у цьому режимі роботи визначається 80x25, тобто кожному рядку відповідає одномірна матриця типу слово довжиною у 80 знаків. Отже перехід на наступний рядок екрану визначається не керуючими кодами ASCII а розміщенням знаку у полі сторінки пам’яті. Залишок першого мегабайта пам’яті, починаючи з C0000h по 100000h передається для потреб завантаження ОС. Адреси вище 1 мегабайту відповідають розширеної пам’яті і доступні після завантаження ОС. Для виводу інформації до відеопам’яті будемо звертатися до комірки пам’яті з відомими фізичними адресами. Для визначення фізичної адреси потрібно знати термінологію, щодо адрес. Логічна адреса складається з двох частин, наприклад ds:0000. Першу частину адреси визначає адреса початку сегменту, що находиться у сегментному регістрі, наприклад ds. Це значення апаратно множиться на 16 і до нього додається зміщення у сегменті (або ефективної адреси). Зміщення у сегменті називають ефективної адресом. Приведемо приклад, нехай для певного експерименту значення DS дорівнює 52F6. Зміщення у сегменті дорівнює 0. Отже його логічна адреса може бути записана DS:0000, або 52F6:0000. Фізична адреса відповідно дорівнює 52F60. Для запису

інформації до відеопам’яті необхідно користуватися фізичної адресом. Як визначити фізичну адресу для практичних потреб.

Розглянемо рис.3. На ньому показаний фрагмент роботи програми. З нього видно, як розташувати ініціалізований масив ar_ans у оперативну пам’ять і звернутися до відповідної адреси дампу пам’яті, відповідного сегменту. У нашому випадку це сегмент даних. Розмірність масиву 3x3, всього дев’ять елементів, тип даних – слово.

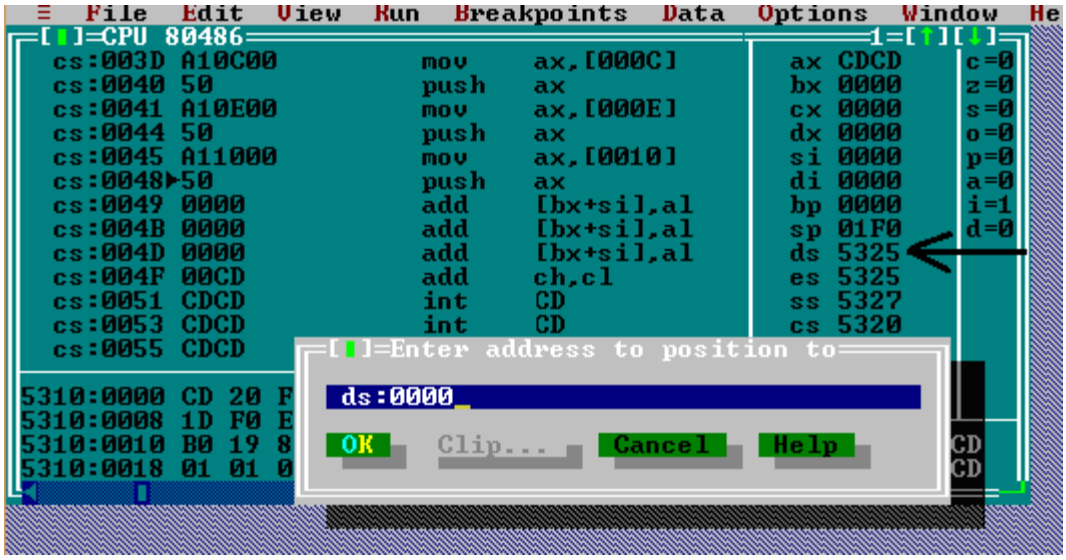


Рисунок 3 - Визначення фізичної адреси масиву у сегменті даних

Масив ar_ans записано у сегмент даних, що підтверджує дамп пам’яті, який показаний на рис.4. Можна переконатися, що всі елементи двовимірного масиву розташовані у оперативній пам’яті послідовно. Нульовий елемент масиву ar_ans має логічну адресу DS:0000h, крайній елемент має логічну адресу DS:0010h. Рисунок 3 - Визначення фізичної адреси масиву у сегменті даних Масив ar_ans записано у сегмент даних, що підтверджує дамп пам’яті, який показаний на рис.4. Можна переконатися, що всі елементи двовимірного масиву розташовані у оперативній пам’яті послідовно. Нульовий елемент масиву ar_ans має логічну адресу DS:0000h, крайній елемент має логічну адресу DS:0010h.


Рисунок 4 - Зміст початку сегмента пам’яті

Таким чином можна визначити ефективну адресу (зміщення у сегменті) для ar_ans[0][0] та ar_ans[3][3]. Для нульового елементу масиву ar_ans воно, відповідно рис.3, дорівнює 0000h. Для крайнього елементу масиву ar_ans - 0010h. Визначимо фізичну адресу нульового елементу масиву ar_ans. Для цього зміст сегментного регістра, що відповідно рис.2, містить число 5325h, множимо на 10h(16) і додаємо значення зміщення (ефективної адреси). Отже для ar_ans[0][0] фізична адреса у оперативної пам’яті дорівнює 53250h, для ar_ans[3][3] відповідно дорівнює 53260h. Приклад одного з варіантів виведення на консоль інформації через відеопамять.

```
;---------------------------------------------------------------
; ЛР №2
;---------------------------------------------------------------
; Архітектура ком
; Завдання:
; ВУЗ: КНУУ КП?
; Факультет: Ф?ОТ
; Курс: 2
; Група:
;---------------------------------------------------------------
; Автор:
; Дата: 08.09.2020
;---------------------------------------------------------------
;-------------------------ЗАГОЛОВОК ПРОГРАМИ------------------
IDEAL
; Директива - тип Асемблера tasm
MODEL small
; Директива - тип модел? пам'ят?
STACK 256
```

```
        ; Директива - розм?р стеку в байтах
;---------------------------МАКРОСИ-----------------------------
        ; макрос для ?н?ц?ал?зац?ї
        MACRO M_Init
        ; Початок макросу
        mov ax, @data ; ax <- @data
        mov ds, ax ; ds <- ax
        mov es, ax ; es <- ax
        ENDM M_Init
        ; К?нець макросу
        ;---------------------------ПОЧАТОК СЕГМЕНТУ ДАНИХ---------- DATASEG
exCode db 0
        ; Одна Л?н?я прямокутника

    rect_line db 31h,31h,32h,31h,33h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
     db 00h,31h,00h,31h,00h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
     db 00h,31h,00h,31h,00h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
     db 37h,22h,37h,22h,00h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
     db 00h,31h,00h,31h,00h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
     db 37h,31h,37h,31h,00h,31h,00h,31h,00h,31h,33h,31h,00h,31h,00h,31h
        rect_line_length=$-rect_line
        test_data dw 0aa77h,0aa77h, 0aa77h
        test_data_length=$-test_data
        ;---------------------------ПОЧАТОК СЕГМЕНТУ КОДУ---------- CODESEG
        Start:
        M_Init
        ;-------------------------------------------------------------
        mov dx,1604 ; Початок виводу прямокутника
        mov ax,0B800h ; 1. Сегментна адреса в?деопамят?
        mov es,ax ; 2. До ES ; Налаштування SI,DI и CX для movsb
        mov di,dx; di <- Початок виводу на екран
        mov si,offset rect_line
        mov cx,rect_line_length ; Число байт?в на пересилання
        cld ; DF - вперед
```

```
rep movsb ; Пересилання Exit:
mov ah,04Ch
mov al,[exCode] ; отримання коду виходу
int 21h ; виклик функц?ї DOS 4ch
;---------------------------------------------------------------
END Start
;---------------------------------------------------------------
```

Таблиця 1

| Варіанти | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Координата x | 2 | 40 | 2 | 40 | 2 | 40 | 30 | 50 |
| Координата y | 2 | 2 | 10 | 10 | 15 | 15 | 30 | 50 |
| Кольори прямокутника | Синій | Зелений | Бірюза | Червон. | Білий | Блакитний | Жовтий | Салатов. |
| Кольори надпису | Зелений | Синій | Червон. | Бірюза | Блакітний | Білий | Салатов. | Жовтий |

ЗАВДАННЯ 1. Вивести до знакової відеопам’яті архітектурі МПС у реальному режимі інформацію так, щоб на консолі утворився прямокутник розміром 20 знаків по горизонталі і 10 знаків по вертикалі. Колір надпису наданий у таблиці, відповідно до варіанту. Кольори прямокутника і координати верхнього лівого кута прямокутника відносно верхнього лівого кута екрану визначені у таблиці 1 відповідно до варіантів.

2. Всі вищеописані елементи утворюються або набором у масиві або з використанням циклічних конструкцій і у вигляді процедур.

# ЛАБОРАТОРНА РОБОТА №3

# ДОСЛІДЖЕННЯ МЕХАНІЗМІВ АДРЕСАЦІЇ АРХІТЕКТУРІ IA-32 (X86) У REAL ADRESS MODE

Мета лабораторної роботи полягає у набутті впевнених знань і навичок технологічної основи розробки ПЗ на Асемблері, у ході якої застосовуються знання архітектури комп'ютерів.

Програма роботи складається з наступних кроків:

- вивчити механізми адресації у архітектурі IA-32 у real address mode;
- виконати повний цикл розробки, тестування і налагодження програмного забезпечення;
- зберегти отриману програму, зробити висновки щодо необхідності знань архітектури комп'ютера у ході розробки ПЗ.

## *Теоретичні відомості для ЛР 3*

Кожна архітектура МПС має свою особливість управління пам'яттю. Розглянемо у найбільш загальних рисах архітектуру МПС на базі Intel 8086. Як більшість архітектурі МПС вона містить базові елементи: процесор, внутрішня пам'ять, периферійні пристрої. Все це об'єднано між собою системною магістраллю. Вона включає шину адрес, шину даних і шину управління. Кількість провідників у шині визначає її розрядність. Шина даних 16-розрядна, шина адрес 20- розрядна. Звичайно архітектура МПС містить два тип внутрішньої пам'яті: постійний пристрій для запам'ятовування (ПЗП) read-only memory (ROM), оперативний пристрій запам'ятовування (ОЗП) random access memory (RAM). Всі змінні, що оголошені у сегменті даних розміщаються у пам'яті ОЗП і до них можливий доступ з будь-якої частини програми. Звичайно програма розподіляється на окремі частини, що не перетинаються, які називаються сегментами. Змінні розташовуються у сегменті даних. Синтаксис оголошення змінних показаний далі.

```
;Оголошення змінних
v1_byte     DB    1
v2_word     DW    0aaffh
```

```
v3_dword        DD      011ff11ffh
```

У даному прикладі оголошено і три змінних. Перша змінна має тип однобайтової змінної, друга змінна має тип машинне слово (максимальний розмір 2 байта), третя змінна має розмір подвійне слово (максимальний розмір 4 байта).

Для доступу до цих змінних важно знати способи адресації архітектурі МПС, особливо це важно для розробці на Асемблері. Існують різні варіанти опису способів адресації у 16 розрядної архітектурі 8086 у режимі Real Adress Mode. Використаємо варіант Рудакова-Фіногенова.

Відповідно варіанту усі способи адресації розділимо на групи: *безпосередня,*

*регістрова,*

*пряма,*

*визначенням адреси пам'яті.*

Крайня група включає: базову, індексну, базово-індексну, базово-індексну зі зміщенням. Почнемо розглядання способів адресації із визначенням адреси пам'яті. Для цього скористуємося вихідним кодом. Як можна побачити з вихідного коду, у самого початку сегменту даних знаходиться двовимірний масив array2Db, що складається з елементів в один байт, має розмір 16x16. Такий тип і розмір масиву обраний тому, що він зручно розміщується у пам'яті і гарно відображається у TD. Запустимо програму і відкриємо значення дампу (змісту у поточний стан часу) пам'яті з використанням TD. Дамп пам'яті має зміст, як показаний на рис.5.

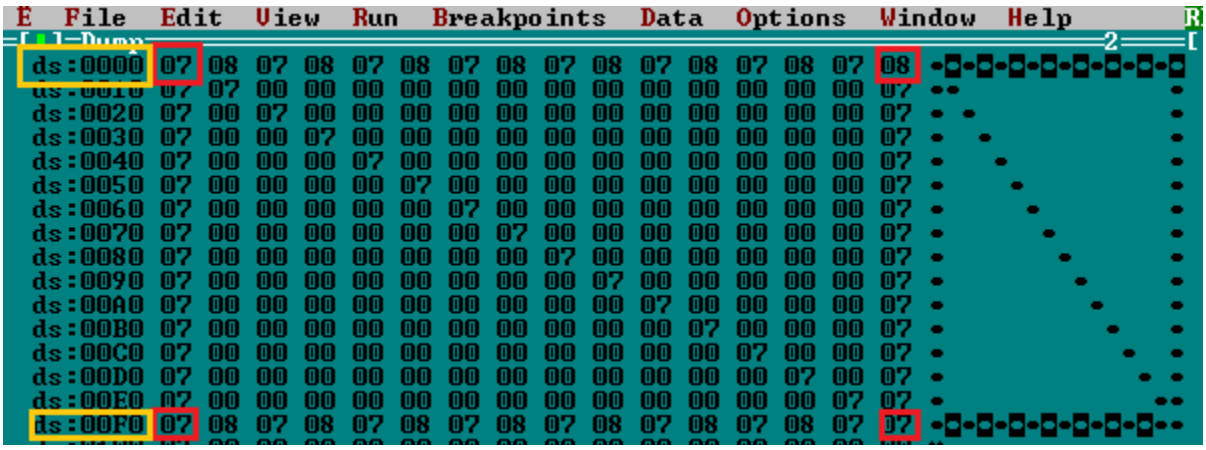


Рисунок - 5 Вихідний зміст пам’яті.

Як можна побачити з рис.5, у вікні дампу пам’яті Turbo Debugger відображає в кожному рядку зміст 16 байтів або 8 машинних слів. Ліворуч від кожного рядка показана так звана логічна адреса самого лівого байта у рядку. Вона складається з двох частин ds:0000 (виділено жовтим кольором). Ця логічна адреса відповідає адресі початку масиву, його нульового елементу. Вона складається з адреси початку сегменту, частина якої находиться у ds і зміщення у сегменті (або ефективної адреси). У якості пояснення визначимо логічні адреси кутових елементів матриці (виділені червоним кольором). Визначимо логічну адресу крайнього верхнього лівого байту двовимірного масиву. Для мого експерименту значення DS дорівнює 52F6. Зміщення цього елементу у сегменті дорівнює 0. Отже його логічна адреса може бути записана DS:0000, або 52F6:0000.

Визначимо логічну адресу крайнього верхнього правого байту двовимірного масиву. Значення DS однаково для всього масиву. Підраховуючи відступ від нульового елементу визначаємо зміщення, що дорівнює 15. У системі числення з основою 16 це відповідає цифрі F. Таким чином логічну адресу можна записати DS:000F, або 52F6:000F. Аналогічно вираховуємо логічну адресу для інших елементів, отримаємо разом:

елемент 0:0, його логічна адреса DS:0000, або 52F6:0000;

елемент 0:16, його логічна адреса DS:000F, або 52F6:000F;

елемент 16:0, його логічна адреса DS:00F0, або 52F6:00F0;

елемент 16:16, його логічна адреса DS:00FF, або 52F6:00FF.

В такий спосіб можна визначити логічну адресу будь-якого елементу (байту) масиву, або іншого об'єкту у ОЗП. Зрозуміло, що у реальних програмах такого зручного розташування масиву у пам'яті не буде. Користуючись Turbo Debugger проведемо дослідження прямої адресації. Синтаксис для TASM показаний у фрагменті, що взятий з вихідного коду і показаний нижче. Для демонстрації прямої адресації до регістру AX записується число 3 і далі, з використанням прямої адресації здійснюється запис до експериментального масиву. Знаючи розташування експериментального масиву у пам'яті, необхідно записати до нього значення з регістру AX.

```
; **1. Пряме звернення до пам'яті з відомою абсолютною ;адресою
mov ax,03h; В ax записується константа.
; Дуже небезпечний механізм. Проте дає повну владу над кодом.
mov [DS:[00h]], ax ; Прямий запис у пам'ять за адресою DS:0000 змісту AX
mov [DS:[01h]], ax ;
mov [DS:[02h]], ax ;
mov [DS:[03h]], ax ;
mov [DS:[04h]], ax ;
mov [DS:[05h]], ax ;
mov [DS:[06h]], ax ;
mov [DS:[07h]], ax
```

Для цього виконаємо програму покроково і перевіримо ділянку дампу пам'яті з використанням TD. Результат показаний на рис.6.

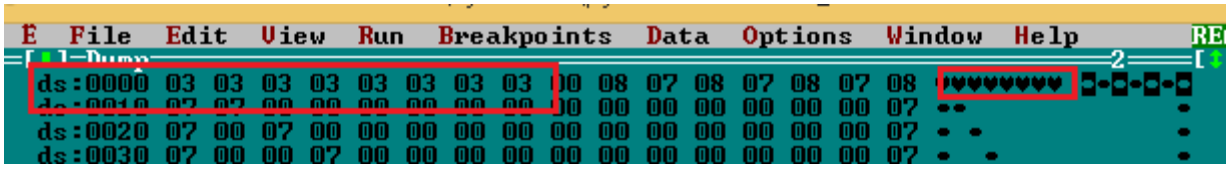


Рисунок - 6 Результат роботи прямої адресації

Під час роботи програми вихідні елементи експериментального масиву було зміщено значеннями регістру AX у діапазоні адрес ds:0000-ds:0007, що відповідає вимогам вихідного коду програми. Таким чином експериментально підтверджено положення синтаксису і результатів роботи. Узагальнімо синтаксис прямої адресації до пам'яті.

1. Може бути використано безпосереднє значення, наприклад:

```
mov [DS:[07h]], ax
```

2. Може бути використана назва операнду, наприклад:

```
mov [v1_byte], ax
```

3. Може використовуватися додатковий сегмент, або інші сегменти, наприклад сегмент коду:

```
mov [ES:[07h]], ax
mov [CS:[07h]], ax.
```

У якості висновку необхідно відмітити, що цей механізм дає можливість повної влади над пам'яттю у Real Address Mode і є небезпечним механізмом. Ми можемо на свій розсуд записати до будь якої ділянки пам'яті будь яке число, що може привести до пошкодження даних. Зрозуміла і незручність цього механізму, що полягає у запису до коду значення абсолютної адреси у вигляді констант. Цей механізм доцільно використовувати для адресації змінних.

Для зручного доступу елементів масивів і ділянок пам'яті призначена базова і індексна адресація. Базова адресація передбачає використання для адресації регістрів BX або BP, що відповідно до синтаксису замикають у квадратні скобки. При використанні регістру BX по умовчанню архітектура МПС береться логічна адреса DS:BX. Приклад синтаксису показаний у коді нижче. При використання регістра BP по умовчанню архітектурі МПС береться логічна адреса SS:BP, дається довільний доступ до стеку, докладно буде описано далі.

```
    ;**Приклад 1, базова адресація.
mov al, 02h
mov bx, 08h;Етап 1. До BX заносимо ефективну адресу потрібної ділянки коду
mov [bx], al ;Етап 2. До пам'яті за адресою [DS]:[BX]. заносимо значення  AX
inc bx              ;Збільшуємо значення BX на 1
mov [bx], al ;Записуємо в інші ділянки пам'яті, все це здійснюється циклічно
inc bx
mov [bx], al
inc bx
mov [bx], al
inc bx
mov [bx], al
inc bx
mov [bx], al
inc by
mov [bx], al
inc by
```

Розглянемо код, що описаний вище. У молодшу частину регістру AX заносимо числове значення 2. До регістру BX заносимо зміщення у сегменті (ефективну адресу), числове значення 08. Для нашого прикладу, це

відповідає другої половині першого рядка експериментального масиву. На першому кроці заносимо значення 2. Далі використовуємо нову команду (inc bx). Ця команда збільшує значення операнду на одиницю. Використовуємо базову адресацію і записуємо у наступну ділянку пам'яті число 2 (mov [bx], al). Виконуємо цю операцію 7 разів, збільшуючи значення BX на одиницю і заповняємо другу половину першого рядка. Результат роботи програми показаний на рис.7.

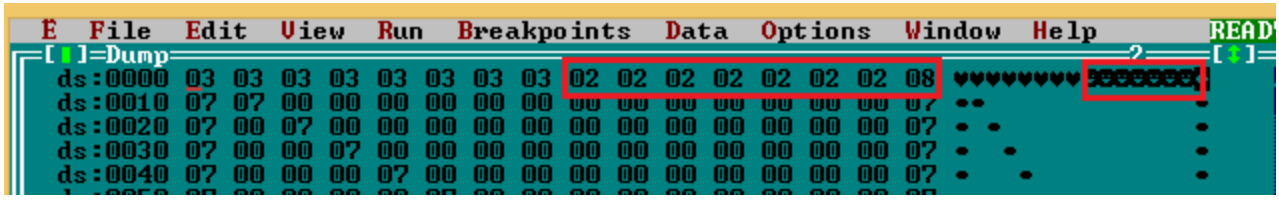


Рисунок - 7 Результат базової адресації

У такий спосіб експериментально підтверджена робота архітектура МПС з використанням базової адресації. Узагальнимо синтаксис базової адресації. Одним з операндів має бути базовий регістр, як це показано далі:

mov [BX], AL

mov [BP], AL.

Сегментний регістр вказувати на пряму не потрібно, по умовчанню при використанні BX береться логічна адреса DS:BX. При використанні BP береться логічна адреса SS:BP.

Індексна адресація передбачає використання для адресації регістрів SI або DI, що відповідно до синтаксису замикають у квадратні скобки. При використанні SI, DI по умовчанню архітектура МПС береться логічна адреса DS:SI або DS:DI. Синтаксис індексної адресації показаний нижче. В цьому коді регістр BX використовується як звичайний РЗП і призначений для зберігання числа, що буде записаний до ділянки пам'яті.

```
;**Приклад 1, індексна адресація.
mov di, 010h ;Визначаємо початкову адресу для запису даних
mov bx, 04h ; Записуємо число, що буде використано
mov [di], bx ;          M(DS*16+DI)<-BX
inc di
mov [di],  bx; M(DS*16+DI+1)<-BX
inc di
```

```
mov [di],  bx; M(DS*16+DI+2)<-BX
inc di
mov [di],  bx; M(DS*16+DI+3)<-BX
inc di
mov [di],  bx
inc di
mov [di],  bx
inc di
mov [di],  bx
inc di
mov [di],  bx
```

До регістру DI заносимо зміщення у сегменті (ефективну адресу), числове значення 10h. Для нашого прикладу, це буде початок другого рядка матриці. До нульового елементу другого рядка матриці записуємо число 4. Далі використовуємо нову команду  inc di, що збільшує значення операнду на одиницю. Використовуємо індексну адресацію і записуємо у наступну ділянку пам'яті число 4. Результат показаний на рис.8.

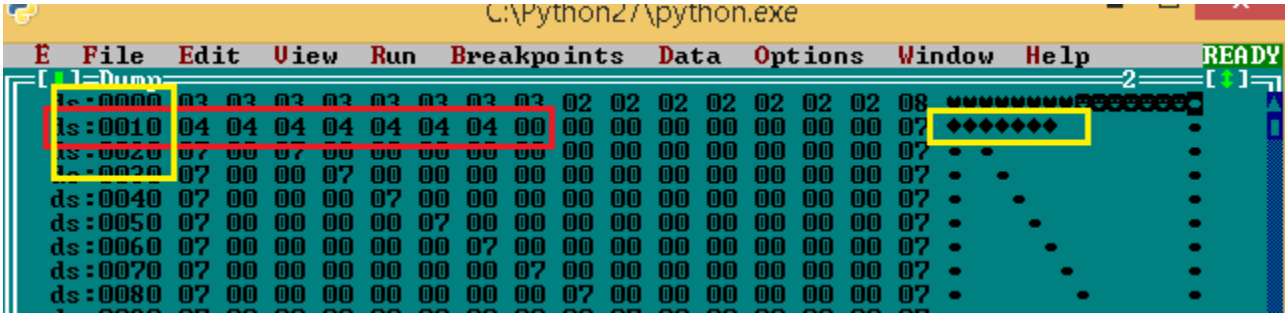


Рисунок - 8 Результат індексної адресації

Таким чином можна зробити висновок. Базова і індексні способи адресації дають можливість доступу до довільної ділянки пам'яті у сегменті даних або у сегменті стеку. Але такий спосіб адресації не дає зручної роботи з багатовимірними масивами.

Базово-індексна адресація дає можливість керувати пам'яттю у двох координатах дампу пам'яті. Відносна адреса операнду визначається добутком базового і індексного регістрів. Дозволяється використовувати наступні  пари регістрів:

[BX][SI], [BX][DI], [BP][SI], [BP][DI].

Продемонструємо, як можна управляти записом до двовимірного масиву array2Db (вихідний код 1.2) у двох координатах. Нехай треба записати до третього і четвертого рядка вихідного масиву по три числа дев'ять,

починаючи з початку рядка. Для рішення цей задачі використовується базово-індексна адресація, фрагмент рішення коду показаний нижче.

```
mov al, 9                       ; Записуємо до молодшої частини AX 9
mov bx, 20h                 ; Готуємо базовий регістр, пересування по вертикалі
mov si, 0h                      ;Готуємо індексний регістр, пересування по горізонт.
mov [array2Db+si+bx], al    ;M(DS*16+ array2Db+SI+BX)
inc si
mov [array2Db+si+bx], al
inc si
mov [array2Db+si+bx], al
add bx, 10h
mov si, 0h
mov [array2Db+si+bx], al
inc si
mov [array2Db+si+bx], al
inc si
mov [array2Db+si+bx], al
```

У якості бази обрано значення початку 3 рядку, це значення записано до базового регістру (mov bx, 20h). До індексного регістру записано значення 0. Далі іде збільшення індексного регістру на одиницю, разом заповнюється третій рядок, три елемента. Далі збільшується значення базового регістру на 10h, що означає стрибок на початок наступного рядку. Нова команда (add bx, 10h). Аналогічно заповнюється другий рядок, результат роботи показаний на рис.9.

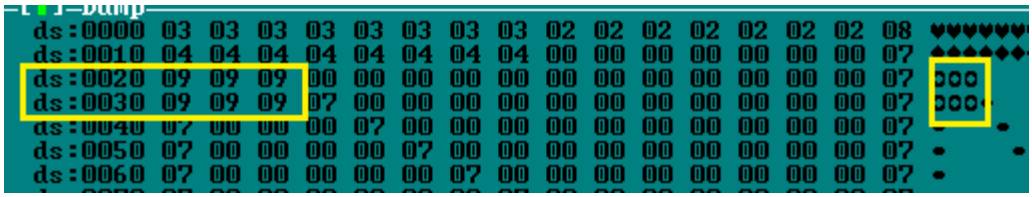


Рисунок - 9 Результат роботи базово-індексної адресації

Таким чином експериментально була підтверджена можливість використання базово-індексної адресації. Крім вищеописаного розробники архитектура МПС передбачили базово-індексну адресацію зі зміщенням. Для чого вона потрібна? Як вже було згадано раніше масиви в практичних кодах не розмірюються, як розміщений масив array2Db. Перед початком масиву завжди може бути ділянка пам’яті. Її треба урахувати і користуватися базово-індексною адресацією. Для цього передбачена базово-індексна адресація зі зміщенням. Вона дає можливість додати константу-зміщення і зручно

керувати пам’яттю у двох координатах. Такий спосіб адресації також є зручним для доступу до масиву, елементами якого є невбудовані типи даних, наприклад структури.

Для базово-індексної і базово-індексної зі зміщенням адресації дозволяється використовувати наступні пари регістрів: [BX][SI], [BX][DI], [BP][SI], [BP][DI] і додаткове зміщення у вигляді константи. Докладного роз’яснення цей спосіб не потребує, оскільки в цілому відповідає базово-індексної адресації. У якості прикладу приведений фрагмент з вихідного коду 1.2.

```
mov [array2Db+si+bx+3], al.
```

Інші види адресації регістрова, безпосередня, стекова особливих пояснень не потребують і описані у вихідному коді. Робота стеку більш докладно буде розкрита у наступній лабораторній роботі.

```
TITLE Vihidni kod 2.1
;-------------------------------------------------------------------------
;ЛР  №1.2 Кодування Кіріліца Windows-1251
;-------------------------------------------------------------------------
;----------------I.ЗАГОЛОВОК ПРОГРАМИ-----------------------
IDEAL
MODEL SMALL
STACK 512
;----------------------II.МАКРОСИ-----------------------------------
; Складний макрос для ініціалізації
MACRO M_Init                ; Початок макросу
mov    ax, @data     ; ax <- @data
mov    ds, ax        ; ds <- ax
mov    es, ax        ; es <- ax
ENDM M_Init

;-------------------III.ПОЧАТОК СЕГМЕНТУ ДАНИХ--------------
DATASEG
;Оголошення двовимірного експериментального масиву 16x16
array2Db      db  7, 8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8
              db  7, 7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,7
              db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,7
```

```
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,7
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,7
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,7
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,7
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,7
                db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,7
                db  7, 8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,7

; Для вирівнювання у дампі
arr_def1 dw  3, 0 , 0 , 0 , 0 , 0 , 0 , 0

;Оголошеня двовимірного масиву 8x8
array2Dw        dw  8, 8 ,8 ,8 ,8 ,8 ,8 ,8
                dw  8, 8 ,8 ,8 ,8 ,8 ,8 ,8
                dw  8, 8 ,8 ,8 ,8 ,8 ,8 ,8
                dw  7, 7 ,7 ,8 ,8 ,7 ,7 ,7
                dw  7, 7 ,7 ,8 ,8 ,7 ,7 ,7
                dw  7, 7 ,7 ,8 ,8 ,7 ,7 ,7
                dw  7, 7 ,7 ,8 ,8 ,7 ,7 ,7
                dw  7, 7 ,7 ,8 ,8 ,7 ,7 ,7
; Для вирівнювання у дампі
arr_def2 dw  3, 0 , 0 , 0 , 0 , 0 , 0 , 0

;Оголошеня двовимірного масиву 4x4
rray2Dd         DD  34925, 34925 ,34925 ,34925
                DD  34925, 30583 ,30583 ,34925
                DD  34925, 30583 ,30583 ,34925
                DD  34925, 34925 ,34925 ,34925

arr_def3 DW  3, 0 , 0 , 0 , 0 , 0 , 0 , 0

; Рядки повідомлень
msg_love  DB  "Max love Natali$"
msg_asm  DB  "Assembler AUTS $"
msg_vb     DB  "Variable in byte"
;Оголошення змінних
v1_byte         DB 11111111b ; однобайтова змінна
arr_def4        DB  3, 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,  0 , 0 , 0 , 0 , 0 , 0
msg_vw          DB    "Variable in word"
v2_word         DW 0aaffh ; змінна 1 машинне слово
arr_def5        DW  3, 0 , 0 , 0 , 0 , 0 , 0
msg_vdd         DB   "Variable in dd  "; 4 байтова змінна
v3_dword        DD 011ff11ffh
arr_def6        DW  3, 0 , 0 , 0
exCode          DB 0
CODESEG
;----------------------VI. ПОЧАТОК СЕГМЕНТУ КОДУ-------------------
Start:
M_Init
;Способи адресації по Рудакову-Фіногенову----------------------
```

```
; 1. Пряме звернення до пам’яті з відомою абсолютною адресою
mov ax,03h   ; В ax записується константа.
;Дуже небезпечний механізм. Проте дає повну владу над кодом.
mov [DS:[00h]], ax   ; Прямий запис у пам’ять за адресою DS:0000 змісту AX
mov [DS:[01h]], ax   ;
mov [DS:[02h]], ax   ;
mov [DS:[03h]], ax   ;
mov [DS:[04h]], ax   ;
mov [DS:[05h]], ax   ;
mov [DS:[06h]], ax   ;
mov [DS:[07h]], ax

mov [v2_word], ax

;2. Базова адресація. Призначена для роботи з масивами

mov al, 02h          ;Число, що буде записано до ділянки дампу
mov bx, 08h          ;До BX заносимо ефективну адресу потрібної ділянки коду
mov [bx], al         ;До  дампу заносимо значення AX
inc bx               ;Збільшуємо значення BX на 1
mov [bx], al   ;Записуємо в інші ділянки пам’яті
inc bx
mov [bx], al         ;Записуємо в інші ділянки пам’яті, все це здійснюється циклічно
inc bx
mov [bx], al         ;
inc bx
mov [bx], al         ;
inc bx
mov [bx], al         ;
inc bx
mov [bx], al         ;
inc bx

mov bp, 01h          ;Етап 1. До BP заносимо ефективну адресу потрібної ділянки
стеку
mov cx, [bp]         ;Етап 2. До CX заносимо значення з пам’яті за адресою [SS]:[BP].



; Індексна адресація.
mov di, 010h
mov bx, 04h
mov [di], bx   ;M(DS*16+DI)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+1)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+2)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+3)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+4)<-BX
```

```
inc di
mov [di],  bx  ;M(DS*16+DI+5)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+6)<-BX
inc di
mov [di],  bx  ;M(DS*16+DI+7)<-BX

mov si, 01h          ;Етап 1. До SI заносимо ;значення зміщення у сегменті даних.
mov ax, [si]         ;Етап 2. До AX заносимо зміст значення з адреси пам'яті [DS]:[SI].


;Базово-індексна адресація
mov al, 9                          ; Записуємо до молодшої частини AX 9
mov bx, 20h                        ; Готуємо базовий регістр, пересування по вертикалі
mov si, 0h                         ;Готуємо індексний регістр, пересування по горізонт.
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
inc si
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
inc si
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
add bx, 10h
mov si, 0h
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
inc si
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
inc si
mov [array2Db+si+bx], al;M(DS*16+ array2Db+SI+BX)
;Приклад базово-індексної адресації у "чистому вигляді"
mov ax, [BP+SI]      ; Наприклад:
mov cx, [BP+DI]
mov dx, [BX+SI]
mov dx, [BX+DI]

;Базово-індексна і індексна адресація зі зміщенням
mov [array2Db+si+bx+3], al;M(DS*16+ array2Db+SI+BX+3)

mov si, 0            ; Ініціалізуємо регістр si - індекс, елемента масиву.
mov dl, [array2Db+si]        ; dl<- array2Db[00]
mov si, 1            ; Аналогічні дії з першим елементом масиву array2Db[1].
mov dh, [array2Db+si]        ; dh<-array2Db[10]
mov si, 2            ; Аналогічні дії з другим елементом масиву array2Db[20].
mov cl, [array2Db+si]        ; cl<- array2Db[20]


;Безпосередня адресація. Операнд входить до складу ;команди
mov ax, 1                    ;(ax<-1)
mov ah, 0ffh                 ;( ah <-0ffh)
mov al, 0aah                 ;( al <-0aah)
mov bx, 02h                  ;(bx<-02h)
mov bh, 011h
mov bl, 0aah
```

```
mov dx, '7'                ;dx (dx<-"ASCII код знаку 7")
mov si, 4                  ;si (si<-4)
mov di, 04h                ;di (di<-04h)
mov di, 11101110b


;Стекова адресація.
; Приклад. В стеку збережені сегментні регістри
push ds                    ; Розміщення в стеку змісту регістру ds, зміст пам'яті
; M(SSx16-SP*2)<-ds
push ss
push es
; Приклад. Відновлення значення СР
pop ds                     ; Відновлення зі стеку змісту регістру cs
pop ss                     ; Аналогічна операція, зміст пам'яті M(SSx16+SP*2)<-ds
pop es
;pop cs Не дозволено

;Регістрова адресація. Операндами є регістри.
mov ds, bx
                           ;mov ss, cx. Дозволено. Але не треба, поламає стек
mov es, dx
Exit:
    mov ah,4ch
```

Методичні вказівки щодо проведення роботи. У якості прототипу обирається вихідний код 1.2. Створюється повний цикл виконання програми і видаляються зайві елементи коду, що не потрібні у ЛР.

Далі обирається потрібний рядок (стовпчик) масиву і, використовуючи потрібний вид адресації, дає можливість заповнити потрібну ділянку коду, літерою що вимагається.

Виконується повний цикл створення програми і виправляються помилки, що були виявлені. Для підтвердження результатів Turbo Debugger фотографується ділянка дампу пам'яті масиву.

ЗАВДАННЯ. Створити двовимірний масив array2Db, що складається з елементів в один байт, має розмір 16x16. Записати на діагональ масиву ініціали студентів групи. Для парних варіантів діагональ починається з лівого

верхнього кута, для непарних з правого верхнього кута. Початок діагоналі по вертикалі змістити до низу на число, що дорівнює варіанту.

Перелік питань для підготовки до лабораторної роботи 3

1. Що відбувається на етапі асемблювання вихідного коду. Які команди викликаються (на прикладі TASM), які їх параметри?
2. Що відбувається на етапі компонування програми (TLINK ). Які команди використовуються у командному рядку і які їх параметри?
3. Яку команду треба набрати для виклику Turbo Debugger (TD). Що відображається на робочому вікні TD?
4. Перелічить регістри загального призначення архітектури Intel 8086. Призначення регістрів, особливості. Розкрийте поняття розрядності регістрів. Яка розрядність регістрів, скільки цифр системи числення з основою 16 необхідно використати для опису змісту регістрів Intel 8086?
5. Перелічіть сегментні регістри архітектури Intel 8086. Розкрийте поняття розрядності регістрів. Яка розрядність сегментних регістрів, скільки двійкових цифр необхідно для опису змісту регістру Intel 8086?
6. Поясність термін реальний режим у архітектурі Intel 8086. Які режими роботи мають процесори архітектури Intel 80386 (та вище), яка розрядність цих процесорів?
7. Розкрийте поняття оперативна пам'ять, адреса і зміст за адресою. Як здійснюється сегментація пам’яті 8086, навіщо розроблений цей механізм?
8. Дайте поняття ефективної адреси, зміщення у сегменті, логічної адреси, фізичної адреси операнду. Як визначається фізична адреса операнду?
9. Команда mov у архітектурі Intel 8086. Дайте поняття операнду. Які операнди команди mov застосовуються при різних способах адресації? Поясніть на прикладі коду.
10. Дайте поняття сегменту даних, сегменту стеку, сегменту коду. Синтаксис їх описання у Асемблері. Як можна визначити адресу початку сегментів з використанням TD ? Поясніть на прикладі коду.

11. Синтаксис описання змінної, чисельного масиву, масиву символів у асемблері. Як визначити фізичну адресу змінної, початку масиву?
12. Як з використанням TD визначити ефективну адресу змінної у сегменті даних, її логічну адресу?
13. Поняття програмного переривання, переривання int 21h, як використовується це переривання для виводу змінної на консоль. Поясніть на прикладі коду.
14. Як в асемблерних командах задається ефективна адреса змінної (початку масиву)?
15. Пряма адресація. Синтаксис, небезпека використання у реальному режимі. Поясніть на прикладі коду.
16. Синтаксис, використання сегментних регістрів.
17. Безпосередня адресація. Синтаксис, небезпека використання у реальному режимі. Поясніть на прикладі коду.
18. Регістрова адресація. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
19. Індексна адресація. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
20. Базова адресація з BX. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
21. Базова адресація з BP. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
22. Базово-індексна адресація. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
23. Базово-індексна адресація зі зміщенням. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
24. Адресація зі стеком. Синтаксис, причина використання у реальному режимі. Поясніть на прикладі коду.
25. Доступ до довільної ділянки стеку.

# ЛАБОРАТОРНА РОБОТА №4
## ДОСЛІДЖЕННЯ РОБОТИ СТЕКУ МПС АРХІТЕКТУРІ IA-32 (X86) У REAL ADRESS MODE

Мета лабораторної роботи полягає у набутті впевнених знань і навичок технологічної основи розробки ПЗ на Асемблері, у ході якої застосовуються знання архітектури комп’ютерів.

*Програма роботи складається з наступних кроків:*

- вивчити механізми адресації у архітектурі IA-32 у real address mode та вивчити додаткові команди Асемблеру, що дають можливість керувати ходом програми;
- виконати повний цикл розробки, тестування і налагодження програмного забезпечення відповідно до завдання, що надано далі;
- зберегти отриману програму, зробити висновки щодо необхідності знань архітектури комп’ютера у ході розробки ПЗ.

*Завдання на лабораторну роботу*

1. Написати процедуру, що створює декілька масивів arr_stack розміром 16x16, тип даних слово у сегменті даних. Кількість масивів відповідає номеру варіанту. Заповнити масив псевдо-випадковими числами, використовуючи способи організації циклів.

2. Розрахувати розмір стеку для розміщення масиву. Користуючись механізмом стеку перенести значення масиву до стеку.

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 1, 5, 0, 7, 1, 9, 9, 9, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3

Рисунок - 10 Вигляд масиву

3. Заповнити у стеку рядок масиву числами дня, місяця, року народження студентів робочої групи. Номер рядку має відповідати номеру варіанту. Скористатися базової адресацією. Визначити фізичну, логічну адреси крайніх елементів рядка з числами народження.

4. Засобами DOS або BIOS зарезервувати пам'ять у розмірі 2 параграфів, визначити і зафіксувати початок нового сегменту, розмір памяті що виділена. Записати до нової ділянки памяті масив, що отриманий у п.2 завдання. Переконатися що він є у новій ділянці, зафіксувати результат. Звільнити зарезервовану пам'ять.

У результатах дослідження необхідно додати вихідний код з коментарями, при відсутності коментарів оцінка знижується на 20%. Робота виконується виключно у діалекті асемблеру TASM. Для підтвердження результатів необхідно запустити налагоджувач Turbo Debugger і зробити фотографії екранів, з виділенням замінених рядків і стовпчиків, аналогічно тому, як це показано на рисунках у методиці проведення експерименту.

Крім того необхідною умовою захисту лабораторної роботи є:

тверді навички розробки вихідного коду, його асемблювання, лінкування ;

вміння користуватися Turbo Debugger, здійснювати покроковий запуск програми, відкривати вікно дампу пам’яті, переходити на необхідну адресу;

визначати адресу з використанням дампу пам’яті будь якого байту у сегменті стека або сегменті коду;

пояснити зміни у пам’яті програми під час кожного кроку, пояснити призначення кожної інструкції.

*Теоретичні відомості*

Сучасні компютери обладнуються опреративною паміттю обсягом від 4 і більше Гбайт. Компютери архітектури IA32 працюють у декількох режимах. В режим real address mode МПС переходить під час завантаження операційної системи. В цьому режимі МПС може адресувати 1Мбайт ОЗП. Повний обсяг

ОЗП компютер може адресувати тільки у protected mode, у захищеному режимі роботи процесора. Необхідно розділити два терміна: адресний простір і оперативна пам'ять (ОЗП). Адресний простір це набір адрес, іноді абстрактних, що може адресувати процесор. Вони не обовязково відповідають реальному фізичному ОЗП.

Типова схема адресного простору показана на рисунку для real address mode. Перші 640 Кбайт адресного простору фізичної памяті (від 00000h до 9FFFFh) відводять під основну оперативну пам'ять, що називають стандартною або звичайною. Цьому адресному простору відповідає зміст сегментних адрес (від 0000h до 9FFFh). На початок цієї памяті заванатажуть таблиці и програми. На самому початку ОЗП, на першому килобайті реалізовані вектори переривань, 256 векторів по 4 байта кожний. Вони забезпечують роботу апаратної частини МПС. Стандартні вектори заповнюються автоматично, інші розробник може додати або змінити.

За векторами переривань розташована так звана ділянка даних BIOS, починая з сегментної адреси 40h. До функції BIOS входить тестування вузлів МПС, завантаження ОС і управління штатними пристроями МПС під час завантаження (клавиатура, дисплей, таймер тощо). Після векторів переривань і BIOS залишкі стандартного ОЗП призаначються для завантаження прикладних і системних програм.

Перед поясненням роботи стеку приведемо приклади опису числових даних. При цьому використаються головним чином три директиви Асемблера:
db (define byte,визначити байт) для запису байтів,
dw(define word, визначити слово) для запису машиного слова,
dd(define double, визначити подвійне слово) для запису подвійних слів. Кожна директива дозволяє записувати як одиночні (скалярні) дані, так і масиви, причому дані можна задавати у двійкової, десяткової або 16-ричной системах числення.

Символьні данні визначаються як тип db, записуються у одинарних або подвійних кавичках. Кодуються у вигляді таблиці ASCII і розміщаються у памяті у вигляді звичайних цілих чисел зі знаком.

Якщо число у коді у десятковій формі, то воно залишается не змінним. Для позначки 16-річного числа використовується буква h, наприклад: 21h, 3A7h, 8FFFh. Якщо ж число починається з цифри-букви (A, B, C…), то перед нею треба обов'язково ставити 0, щоб транслятор зрозумів, що це саме число, а не ім'я регістру, наприклад: 0FFF8h, 0Alh, 0C000h. Крім того у програми можуть використовуватися додаткові оператори для роботи із змінними і їх адресами. У синтаксисі TASM для розміщення слова у ділянці памяті з визначеною адресою записується:

```
mov [word ES:DI], 0E40Fh.
```

Аналогичний запис буде використаний для байта подвійного слова, тощо

```
mov [byte ES:DI], 0E4Fh,
mov [dword ES:DI], 0EABB40h.
```

Стек це програмно-апаратний елемент у архітектурі МПС. Основне його призначенн – тимчасове зберігання даних. Стек відрізняє найбільша швидкість під час читання і запису інформації до ОЗП. Саме тому він часто використовується при забезпеченні процедур, зміннх, сутності, масивів тощо. У високорівневих мовах це здійснюється не залежно від розробника. У Асемблері розробник має безпосередньо використовувати команди, що призначені для роботи з стеком. Є дві базові команди push [operand], pop[operand]. Розглянемо більш докладніші як працює стек.

Як відомо з курсу лекцій, програма складається звичайно з трьох сегментів, що не перетинаються у ОЗП один з одним, а саме: сегменту коду, сегменту даних, сегменту стеку. Кожному з них відповідають сегментні регістри, наприклад для зберігання початку сегменту стеку використовується сегмент SS. На початку програми директивою визначається розмір сегменту стеку STACK 512, тобто розмір 512 байт або 200h. Це означає, що у сегменті стеку резервується 200h памяті і покажчик стеку SP у пустому стеку вказує на

логічну адресу SS:0200h, див. рис.2. Тобто на вершину стеку. У сегменті стеку записується адреса початку стеку. Таким чином вершина стеку знаходится на відстані від нульових адрес стеку SS:0000h. Під час зростання стеку, що відбувається при виконанні наприклад команди push AX, значення покажчика ЗМЕНЬШУЄТСЯ на 2, у відповідну ділянку стеку розміщується значення слова з регістру AX.

Кажуть, що стек зростає, хоча адреса у SP покажчику стеку зменшується на 2. Покажемо це на прикладі коду і роботи TD, що показаний на рис.11. Після ініціалізації сегменту даних і виконанні команди push, показано на рис.11 корічневим кольором, значення покажчика стеку зменшується на 2 і дорівнює 01FEh, виділено жовтим кольором. Значення SP (виділено блакітним) містить 01FEh, що на 2 меньше ніж 0200h.  А в сегмент стеку розміщається значення AX (виіделно червоним). Отже значення AX і значення у сегменті співпадають AABB.

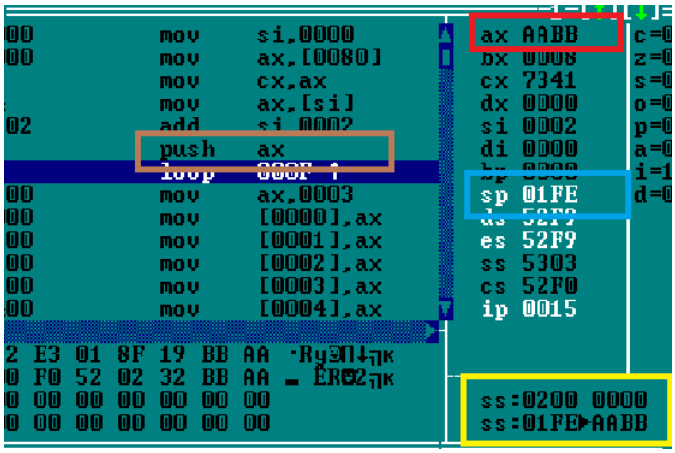


Рисунок - 11 Виконання команди push AX

Такий спосіб організації стеку характерний саме для архітектури процесорів Intel і аналогічних до них. Тобто стек зростає а ефективна адреса вершині стеку, зміст SP має зменшення адреси. Це при вивченні може привести і доволі часто приводить до певної плутанини, оскільки  у інших архітектурах МПС можливо збільшення адресів вершини стеку під час його зростання. *Для будь якого виду МПС треба запам'ятати, що при виконанні команди push*

*[operand], вважається, що стек зростає і на вершині стеку знаходится крайній поточний операнд.* Цей операнд доступний командою pop [operand].

Отже операнд може бути вийнятий зі стеку командою pop [operand]. У цьому випадку до операнду розміщується елемент, що знаходився на вершині стеку. Покажчик зсувается з вершини нижче до стеку. Кажуть стек зменшується. При цьому, для нашої архітектури Intel 8086, після виконання команди pop [operand] ефективна адреса вершини стеку (зміст SP) збільшиться на 2. Отже стек зменьшуется, а адреси вершини збільшуються, що теж приводить до плутанини. Таким чином *для будь якого виду МПС треба запам'ятати, що при виконанні команди pop [operand], вважається, що стек зменшується, зміст вершини стеку розміщується у [operand], а вершина стеку пересувається на наступний нижчий операнд у стеці.* Не зважаючи на адреси.

Розглянемо приклад коду, що демонструє виконання команди *push [operand]*. Організація циклів. Розглянемо перший випадок у канонічному варіанті, користуючись фрагментом коду. У результаті роботи цього коду масив з логічною адресою ds:[si] заповнюється числами 3.

```
; Типовая организация цикла, індексна адрес
                mov al, 03h
                mov cx, 100h ;Заповнення усього масиву
ptr_1:
                mov [ds:[si]], al
                inc si
                loop ptr_1
```

Як це працює. У прикладі використана канонічна організація циклу з використанням команди loop ptr_1. Ця команда буде переводити управління на мітку ptr_1, поки значення регістру CX не стане дорівнювати 0. Використана індексна адресація. До регістру CX записано значення 100h, тобто число повторень у циклі. Інших маніпуляцій с CX не потрібно, він буде зменшуватися на 1 у кожній ітерації циклу. В цьому полягає особливість регістру CX, яка властива тільки ньому. Використаємо цю властивість і продемонструємо роботу команди push на прикладі коду що показаний нижче.

```
;Робота стеку, команда push
lea si, [array2Dw]
```

```
mov ax, [array2Dwlen]
mov cx, ax
stack1:
mov ax, [si]
add si, 2
push ax
loop stack1
```

Розглянемо більш докладно, як працює фрагмент програми для демонстрації роботи команди push. Цей фрагмент забезпечує розміщення вихідного масиву array2Dw, що знаходится в сегменті даних до сегменту стеку повністю. Перші три команди, що надані далі

```
lea si, [array2Dw]
mov ax, [array2Dwlen]
mov cx, ax
```

виконують наступні функції: розміщення адреси початку масиву у SI, розміщення у AX длини масиву, розміщення у CX значення длини масиву, що є підготовкою до організації типового циклу loop. Наступні чотирі команди

```
mov ax, [si]
add si, 2
push ax
loop stack1
```

виконують слідуючі функції:

до AX розміщується значення першого елементу масиву, перехід на другий елемент масиву за рахунок збільшення SI на 2, розміщення у масиві значення AX, зменшення CX на 1 (у фоні), нова ітерація циклу. У такий спосіб всі елементи вихідного масиву з сегменту даних розміщуються у сегменті стеку. Результат роботи програми показаний на рис. 12


Рисунок - 12 Результат роботи програми

Червоним виділено початок стеку, і вершину стеку. У стеку знаходится масив повністю. Кім масиву до стеку розміщені додаткові значення і вершина стеку представлена числом 2420.

Проведемо дослідження зворотної операції pop. Для дослідження показаний фрагмент коду

```
;Дослідження команди pop
lea si, [arrayRes]
mov cx, 64

stack2:
    pop ax
    mov [si], ax
    add si, 2
    loop stack2
```

Пояснимо фрагмент коду. Перші дві команди призначені для підготовки елементів масиву. До SI розміщується адреса початку масиву до якого буде здійснюватися запис інформації з сегменту стеку командою pop. Для підготовки виконання циклу до CX розміщується кількість елементів у масиві.

Далі у тілі циклу виконується команда pop AX. Після цієї команди у регістрі AX розміщується значення вершини стеку, а покажчик стеку заглиблюється на один елемент (одне слово). Далі зміст AX записується до масиву arrayRes. І цикл повторюється при виконанні команди pop адреса у SP покажчику стеку збілшуєтсся на 2. У нашій архітектурі МПС це ознака зменшення стеку. Покажемо результат роботи програми на рис.13.

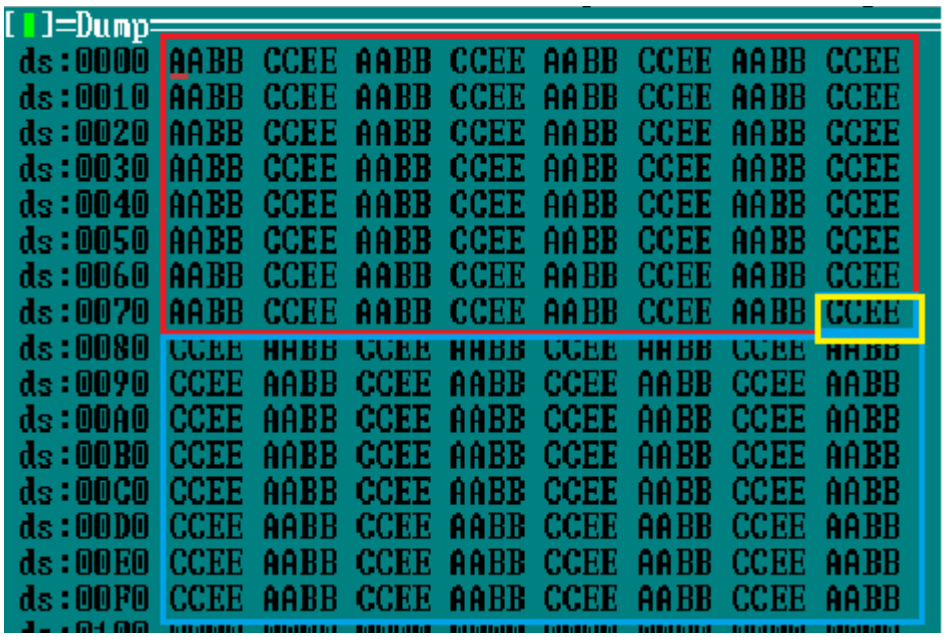


Рисунок - 13  Результат роботи програми, зміст сегменту даних

У червоному прямокутнику показаний масив, що був розміщений у стеку. Після його розміщення на вершині стеку знаходилося число, що виділено у жовтому прямокутнику. Внаслідок роботи програми до нового масиву були занесені значення зі стеку. Першим елементом став елемент, що розміщений у вершині. Внаслідок цього елемени вихідного масиву і нового помінялися місцями.

Архітектура МПС дозволяє довільний доступ до стеку з використанням базової адресації. Для цього приведемо фрагмент коду.

```
; Дослідження базової адресації для доступу до стеку
mov bp, 0200h; Вершина стеку
mov [bp], 0A0Ah
```

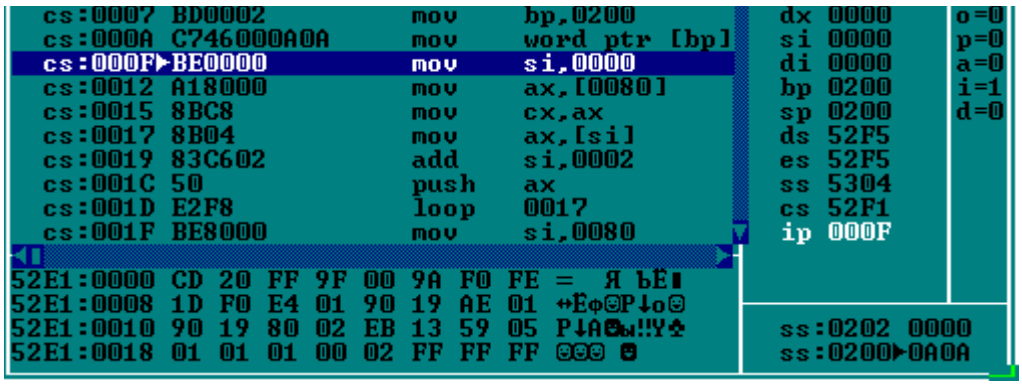


Рисунок - 14 Результат роботи програми базової адресації до стеку

Поясним фрагмент коду. У першому рядку коду розмістимо до базового регістру BP значення 0200h, тобто ефективну адресу вершини стеку. Далі, користуючись правилами базової адресації розмістимо до вершини стеку значення 0A0Ah, виконується команда mov [bp], 0A0Ah. МПС не вимагає явно вказувати сегментний регістр стеку. Результат роботи програми показаний на рис.14. Як можна побачити, зміст вершини сегменту стеку відповідає вихідному коду. Далі приведений повністю код методичних рекомендацій.

```
TITLE Vihidni kod 2.1
;-------------------------------------------------------------------------
;ЛР  №1.2 Кодування Кіріліца Windows-1251
;Дослідження стеку
;-------------------------------------------------------------------------
;----------------I.ЗАГОЛОВОК ПРОГРАМИ-----------------
IDEAL
MODEL SMALL
STACK 512

;-----------------II.МАКРОСИ------------------------
;2.1 Складний макрос для виходу з програми
MACRO M_Exit              ; Початок макросу
; На виході:AL = код завершення програми
; На вході:AH = ознака переривання DOS виходу 04Ch
mov          ah, 04Ch
int21h
ENDM M_Exit               ; Кінець макросу

;2.2 Складний макрос для ініціалізації
MACRO M_Init              ; Початок макросу
mov  ax, @data     ; ax <- @data
mov  ds, ax        ; ds <- ax
mov  es, ax        ; es <- ax
ENDM M_Init               ; Кінець макросу
```

```
;--------------------III.ПОЧАТОК СЕГМЕНТУ ДАНИХ
DATASEG

;Оголошеня двовимірного масиву 8x8
array2Dw
dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
 dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
 dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
 dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
 dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
 dw  0AABBh, 0CCEEh ,0AABBh ,0CCEEh, 0AABBh, 0CCEEh ,0AABBh ,0CCEEh
array2Dwlen = $-array2Dw

arrayRes dw '*' dup (64)
arrayReslen = $ - arrayRes

msg_asm  DB  "Assembler AUTS $"

exCode db 0
CODESEG
;----------------------VI. ПОЧАТОК СЕГМЕНТУ КОДУ
Start:
M_Init
; Дослідження базової адресації для доступу до стеку
        mov bp, 0200h; Вершина стеку
        mov [bp], 0A0Ah
;Дослідження команди push
        lea si, [array2Dw]
        mov ax, [array2Dwlen]
        mov cx, ax
stack1:
        mov ax, [si]
        add si, 2
        push ax
        loop stack1

;Дослідження команди pop
        lea si, [arrayRes]
        mov cx, 64

stack2:
        pop ax
        mov [si], ax
        add si, 2
        loop stack2
;                    =====Дослідження управління памяттю====
        nop
       nop
        ;***Функція виділення памяті. Вхідні параметри***
```

```
        mov ah, 048h ; Ознака переривання
        mov bx, 02h ; Розмір нової ділянки у параграфі
        ;***Parameters for output***
        ; CF = 0 if ok
        ; AX Адреса нового сегменту памяті
        ; CF = 1 Ознака помилки виконання перенивання
        ; AX = 7 Ознака помилки
        ; AX = 8 Ознака малого обсягу памяті
        ; BX розмір нової ділянки
        int 21h
        ; Перенесення масиву до нової ділянки памяті
        mov es, ax    ;Ініціалізація сегментного регістру новою адресою
        mov cx, 8     ;Визначення розмірну нової ділянки масиву множенням
        mov ax, 8
        mul cx
        mov cx, ax    ;Підготовка циклу до виконання, кількість ітерацій
        xor di, di
        lea si, [array2Dw] ;Ефективна адреса  масиву до індексного рег.

mem1:
        mov dx, [si]          ; Нульовий елемент в регістр. Індексна адресація
        mov [es:di], dx       ;Запис до нової ділянки, індексна адресація
        add si, 2             ; Перехід на новий елемент масиву
        add di, 2
        loop mem1

        nop
       nop
        ;**Зміна розміру ділянки. Вхідні параметри
        mov ah, 04Ah          ;Ознака команди
        mov bx, 01h           ;Новий розмір
        ; ES адреса сегменту, що буде змінюватися
        ;***Вихідні параметри***
        ; CF = 1 if not ok
        ; AX = 7 if memory blocks is destroyed
        ; AX = 9 Некоректна адреса
        ; AX = 8 Невистачає памяті
        ; BX розмір нової памяті
        int 21h

        nop
       nop
        ;**Звілнення памяті. Вхідні параметри***
        mov ah, 049h ; command mark
        ; ES адреса блоку, що звільняєтся
        ;***Вихідні параметри***
        ; CF = 0  ok
        ; CF = 1, AX = 7 if memory blocks is destroyed
        ; CF = 1, AX = 9 Некоректна адреса

        int 21h
```

```
Exit:
    mov ah,4ch
    mov al,[exCode]
    int 21h
   end Start
```

Для управляння памяттю і виділення додаткової ділянки, що призначення для роботи з даними у операційної системі DOS у Асемблері є засоби для розподілу ОЗП у розмірі 1 Мбайта. Ці засоби – переривання DOS є аналогом стандартних функцій мови програмування С, для виділення памяті функція С malloc, зміна обсягу памяті функция realloc, звільнення ділянки памяті free. Розглянемо перше переривання для виділення памяті, для цього використаємо фрагмент вихідного коду 2.1, що наданий далі.

```
mov ah, 048h ; Ознака переривання
mov bx, 02h ; Розмір нової ділянки у параграфі
;***Parameters for output***
; CF = 0 if ok
; AX Адреса нового сегменту памяті
; CF = 1 Ознака помилки виконання перенивання
; AX = 7 Ознака помилки
; AX = 8 Ознака малого обсягу памяті
; BX розмір нової ділянки
int 21h
```

У цьому фрагменті показан приклад виклику переривання DOS і виділення ділянки памяті розміром у 2 параграфа. Результат роботи цієї ділянки програми можна пояснити на результаі роботи програми на рис.15

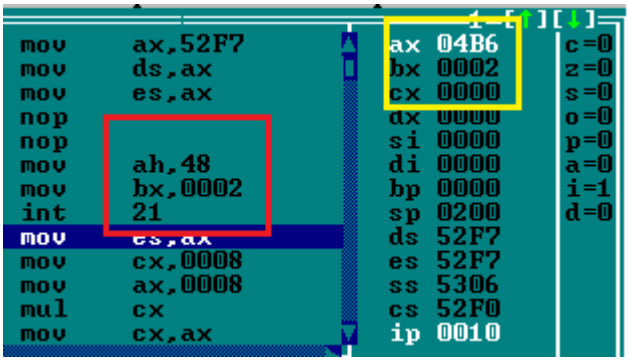


Рисунок - 15 Результат виділення памяті

Червоним кольором виідлені команди переривання для надання ділянки памяті. Жовтим кольором виділені регістри, що є вихідними для цього переривання. У регістрі АХ показана нова адреса нової ділянки, що

зарезервована перениванням, а саме 0048. У регістрі BX показан розмір нової ділянки памяті, у регістрі CX нульові значення, що означає корректне виконання програми.

Приведемо приклад як корисуватися нової ділянкою. Для цього запишемо до нової ділянки значення масиву array2Dw. Пояснимо це на прикладі кода, що є фрагментом вихідного коду.

Першим і головним кроком є ініціалізація сегментного регістра початком нової ділянки памяті mov es, ax, оскільки індексна адресація без цього неможлива. Далі у цьому коді використовуються два індексних регістра і індексна адресація. Регістр SI призначений для звернення до елементів вихідного масиву array2Dw, див. фрагмент коду. Регістр DI призначений для звернення до нової ділянки памяті. Використовується типова організація циклу, для визначння кількості ітерацій здійснюються обчислення, результат заноситься до CX. У регістр SI заноситься початкова адреса масиву array2Dw.

У циклі здійснюється запис нульового елементу array2Dw до DX, оскілки операції пам'ять-пам'ять не дозволені. Далі це значення записується до нової ділянки памяті mov [es:di], dx і здійснюється перехід на новий елемент шляхом збільшення двох індексних регістрів.

```
; Перенесення масиву до нової ділянки памяті
mov es, ax    ;Ініціалізація сегментного регістру новою адресою
mov cx, 8     ;Визначення розмірну нової ділянки масиву множенням
mov ax, 8
mul cx
mov cx, ax    ;Підготовка циклу до виконання, кількість ітерацій
xor di, di
lea si, [array2Dw] ;Ефективна адреса  масиву до індексного рег.

mem1:
    mov dx, [si]          ; Нульовий елемент в регістр. Індексна адресація
    mov [es:di], dx       ;Запис до нової ділянки, індексна адресація
    add si, 2             ; Перехід на новий елемент масиву
    add di, 2
    loop mem1
```

Результат показаний на рис.16. На горі рисунку показаний вихідний масив, що записувався до нової ділянки памяті.

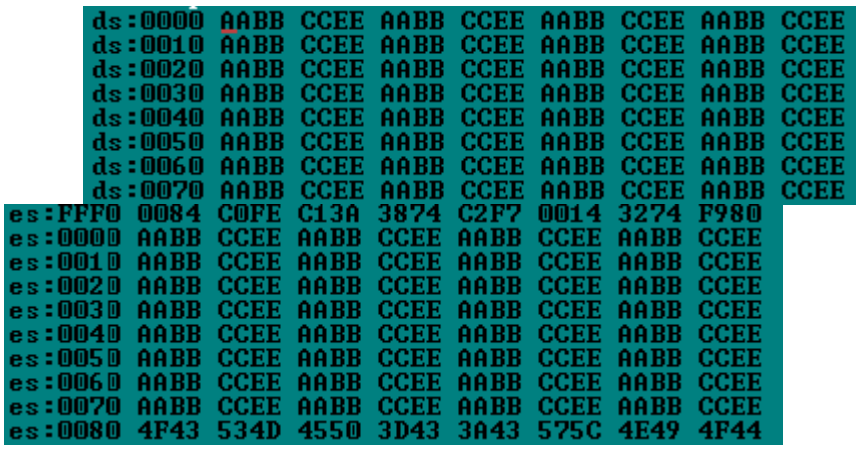

Рисунок - 16 Результат перепису масиву до нової ділянки

У нижньої часини рисунку показаний результат роботи запису до нової ділянки памяті. Інші функції управління памяттю особливих пояснень не потребують оскліки докладно описані у вихідному коді і мають відповідні коментарі.

Перелік питань для підготовки до лабораторної роботи

1. Як написати .bat файл для спрощення і пришвидшення асемблювання і компонування програми. Поясніть на прикладі коду.

2.* Перелічіть регістри загального призначення і сегментні регістри МПС Intel 8086. Розкрийте поняття розрядності регістрів, призначення, особливості регістрів.

3.* Дайте поняття ефективної адреси, зміщення у сегменті, логічної адреси, фізичної адреси операнду. Як визначається фізична адреса операнду?

4.* Як здійснюється сегментація пам’яті МПС на базі Intel 8086, як визначається фізична адреса операндів. Який може бути максимальний і мінімальний розмір сегменту?

5.*Команда mov у архітектурі Intel 8086. Дайте поняття операнду. Які операнди команди mov застосовуються при різних способах адресації? Поясніть на прикладі коду.

6.*Як можна визначити адресу початку сегментів з використанням TD ?

7.* Синтаксис описання змінної, чисельного масиву, масиву символів у асемблері. Поясніть на прикладі коду. Як визначити фізичну адресу змінної, початку масиву з використанням TD?

8.*Поняття програмного переривання, переривання int 21h, як використовується це переривання для виводу змінної на консоль? Поясніть на прикладі коду.

9.*Основні команди для роботи зі стеком. Які регістри використовуються для управління стеком? Призначення і використання цих регістрів в різних способах адресації. Поясніть на прикладі коду.

# ЛАБОРАТОРНА РОБОТА №5

# УПРАВЛІННЯ ХОДОМ ВИКОНАННЯ ПРОГРАМИ НА АСЕМБЛЕРІ АРХІТЕКТУРІ IA-32 (X86) У REAL ADRESS MODE

Перед виконанням роботи необхідно вивчити програмну модель МПС intel 8086, адресацію у режимі Real address mode. Для цього скористатися результатами лабораторної роботи 4. Вивчити додаткові команди Асемблеру, що дають можливість керувати ходом програми. Вивчити додаткові переривання DOS і BIOS для відображення інформації на консолі.

Завдання на лабораторну роботу 7 варіантів.

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3
3, 3, 3, 0, 7, 5, 7, 7, 5, 7, 1, 3, 3, 3, 3, 3
3, 3, 3, 0, 5, 7, 7, 5, 7, 1, 1, 3, 3, 3, 3, 3
3, 3, 3, 0, 8, 2, 7, 9, 9, 3, 1, 3, 3, 3, 3, 3
3, 3, 3, 0, 9, 9, 3, 7, 7, 9, 9, 3, 3, 3, 3, 3
3, 3, 3, 0, 0, 7, 5, 7, 7, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 0, 0, 7, 5, 7, 3, 7, 3, 3, 3, 3, 3, 3
3, 3, 3, 0, 0, 7, 5, 7, 0, 1, 7, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3

Рисунок - 17 Вигляд масиву

1. Користуючись результатами роботи 1, 2, заповнити внутрі массиву array2Db ділянку розміром 8x8 числами дня, місяця, року народження студенту, що є елементами масиву. Наприклад даті народження 27.07.1977 відповідає масив типу db на 8 елементів:

day_n db 2, 5, 0, 7, 1, 9, 7, 7, де кожна цифра відповідає даті, місяцу і року народження.

2. Кординати початку ділянки (i, j) верхнього лівого кута мають відповідати варіанту.

3. Створити процедуру Асемблера, що робить сортування масиву для парного варіанту за зростанням, для непарного варіанту за зменьшенням. arr_dup2D

За бажанням студента написати фрагмент коду, що здійснює вивід масиву на консоль з використанням функцій DOS або BIOS.

Звіт має містити типовий титульний аркуш для лабораторних робіт, з підписом студента, результати дослідження і висновки. У результатах дослідження необхідно додати вихідний код з коментарями, при відсутності коментарів оцінка знижується на 20%. Робота виконується виключно у діалекті асемблеру TASM. Для підтвердження результатів необхідно запустити налагоджувач Turbo Debugger і зробити фотографії екранів, з виділенням замінених рядків і стовпчиків, аналогічно тому, як це показано на рисунках у методиці проведення експерименту.

Крім того необхідною умовою захисту лабораторної роботи є:

тверді навички розробки вихідного коду, його асемблювання, лінкування ;

вміння користуватися Turbo Debugger, здійснювати покроковий запуск програми, відкривати вікно дампу пам’яті, переходити на необхідну адресу;

визначати адресу з використанням дампу пам’яті будь якого байту у сегменті стека або сегменті коду;

пояснити зміни у пам’яті програми під час кожного кроку, пояснити призначення кожної інструкції.

*Теоретичні відомості*

У синтаксисі TASM слово, що призначено для визначення делянки памяті. Приклад використання показаний нижче. Наприклад, при безпосередньої адресації для розміщення слова у ділянці памяті з визначеною адресою приклад запису наступний:

```
mov [word ES:DI], 0040Fh.
```

Аналогичний запис буде використаний для байта подвійного слова, тощо

```
mov [byte ES:DI], 0E4Fh,
mov [dword ES:DI], 0EABB40h.
```

У МПС можуть оброблятися позитивни (безнакові) числа і числа зі знаком. Якщо ціле число позитивне, в машине слово або регістр можна записати 64 Кбт информации, тобто число у діапазоні від 0000h до FFFFh, або

від 00000 до 65 535. Для сберігання чисел зі знаком, використовують ти самі регістри, тільки старший біт регістра (слова) резервується для сберігання знаку. У цьому випадку наявність у старшому біте одиниці означає, що це відємне число, в іншому випадку вважается, що це позитивне число. Відємні числа знаходяться у діапазоні 8000h...FFFFh, позитивні числа в діапазоні в діапазоні 0000h...7FFFh. При цьому негативні числа записуються в додатковому коді. Для перетворення позитивного числа у негативне його потрібно інвертувати і додати одиницю.

У МПС знак числа умовний і залежить від контекста використання. Наприклад FFFBh, можна розглядати як позитивне так і негативне, в залежності від контексту використання. Таким чином, знак числа є характеристикою не самого числа, а способу його обробки у МПС і команд Асемблеру. Процессор може виконувати операції не тільки над словами, але й над байтами. Число у байті можна розглядати як беззнакове, що має діапазон від 0 до 255, або як число зі знаком. Тоді діапазон позитивних значень зменшується у два рази (від 0 до 127 позитивних чисел, -1 до - 128 відємних чисел).

Серед команд Асемблера, що виконують ту або іншу обробку чисел, можна виділити команди, нечутливі до знака числа, наприклад, INC, DEC, TEST, MUL, DIV, JA, JB і ін. Ці команди будь яке число бачать як позитивне, не залежно від стану старшого біту у слові або РЗП. Є група команд Асемблера спеціально призначені для обробки чисел зі знаком (IMUL, IDIV, JG, JL і т.д.). Для цих команд МПС обов'язково «дивится» у старший біт РЗП або слова і визначає позитивність числа.

Для прикладу розглянемо команди множення. Для множення беззнакових чисел використовується MUL(multiplication, множення), для чисел зі знаком команда - IMUL (integer multiplication). Обидві команди можуть працювати як зі словами, так і збайтами. Вони виконують множення числа, що перебуває в регістрах AX (у випадку множення на слово) або AL (у випадку множення на

байт), на операнд - регістр або комірці пам'яті. Недопускається множення на безпосереднє значення, а також на вміст сегментного регістра. Розмір результату в них завжди у два рази більше розміру співмножників. Для1-байтовых операцій отриманий добуток записується в регістр AX. Для 2-байтовых операцій результат множення, що має розмір 32 біта, записується в регістри DX і AX (в DX-старша половина, в AX-молодша).

Система команд. Більшість програм, незалежно від того, на якій мові вони написані, вимагають зміни лінійної послідовності виконання операторів і переходу на інші частини програмного коду. Також вимагається організація циклічних операцій і переходу в різні місця коду. Для цього використовуються команди загального призначення (general-purpose instructions), що можна розділити на кілька груп:

- команди переміщення (пересилання, передачі) даних;
- команди арифметики (додавання, вирахування, множення й ділення) з цілими числами;
- команди логічних операцій;
- команди передачі керування (умовних і безумовних переходів, виклик процедур);
- команди строкових операцій (іноді зустрічається назва "строкові, або ланцюгові, команди")

Розглянемо білшь доклдано групу команд, для передачі управління. Ця важлива группа команд розподіляється на підгрупи. Перша розрізняє числа зі знаком, друга - числа без знака, третя - це команди умовних переходів. Ці команди дозволяють утворювати циклічні операції і операції переходів. Для цього команди умовних переходів часто використають після команд порівняння (CMP), инкремента (INC), декремента (DEC), додавання(ADD), вирахування(SUB), перевірки (TEST) і ряду інших.

Приведемо перелік команд умовних переходів. Знакові команди:

jg (jump if greater - перехід, якщо більше),

jge(jump if greater or equal o- перехід, якщо більше або дорівнює),

jl (jumpifless-перехід,якщоменше),

jng (jump if not greater - перехід, якщо не більше),

jnge ( jump if not greater or equal - перехід, якщо не більше й не дорівнює),

jnl (jump if not less - перехід, якщо не менше),

jnle (jump if not less or equal - перехід, якщо не менше й недорівнює),

Беззнакові команди:

ja (jump fabove-перехід, якщовище),

jae (jump if above or equal - перехід, якщо вище або дорівнює),

jb(jump if below-перехід, якщо нижче),

jbe(jump if below or equal-перехід, якщо нижче або дорівнює),

jna(jump if not above - перехід, якщо не вище),

jnae(jump if not above ore qual - перехід, якщо не вище й недорівнює),

jnb(jump if not below - перехід, якщо не нижче),

Приклади команд, не чутливих до знака числа:

je (jump if equal - перехід, якщо дорівнює),

jne (jump if note qual - перехід, якщо не дорівнює),

jc (jump if carry - перехід, якщо прапор CF установлений),

jcxz (jump if CX=0 перехід, якщо CX=0).

Різниця між знаковими й беззнаковими командами умовних переходів полягає в тім, що знакові команди розглядають поняття "менше" стосовно до числової осі -32К до +32К. Беззнакові команди розглядають числову ось від 0 до 64К. Таким чином, при порівнянні знакових чисел використаються терміни "більше" і "менше", а при порівнянні беззнакових - "вище" і "нижче".

Команда безумовного переходу JMP, передає управління до визначеної у програмі мітки. Саме ця команда дає можливість керувати ходом виконання програми. Команда не впливає на прапори процессора. Команда JMP має 5 разновидів: перехід прямий короткий (в межах -128...+ 127 байт); перехід прямий ближчий (в межах поточного сегменту команд); переход прямий

дальній (в інший сегмент команд); перехід непрямий ближчий; перехід непрямий дальній. В деяких випадках Асемблер може визначити вид переходу по контексту, в деяких випадках для цього використовуються спеціальні атрибути, наприклад:

- SHORT - перехід прямий короткий (в межах -128...+ 127 байт);
- NEAR PTR - перехід прямий ближчий (в межах поточного сегменту команд);
- FAR PTR - переход прямий дальній (в інший сегмент команд);
- WORD PTR - перехід непрямий ближчий;
- DWORD PTR - перехід непрямий дальній.

Команди доступу до памяті. Команда LDS виділяє з памяти за вказаною адресою подвійне слово, що має логічну адресу. Відносну частину адреси загружають до вказаного регістру, а сегментну адресу загружають до регістру DS в контексті. Команда має вигляд:

LDS REG, MEM .

У якості першого операнда команды LDS має бути РЗП, у якості другого ділянка памяті, що визначається. Команда не впливає на прапори.

Команда LEA завантажуе до першого операнду, ефективну адресу второго операнду, що визначається у вигляді ділянки памяті, змінної, початку масиву, наприклад

LEA REG,MEM.

Команда не впливає на прапори процесора. Розглянемо приклад використаня команди, фрагментом коду:

lea dx, [no_ziro]

Після виконання цієї команди до регістру буде записана ефективна адреса масиву no_ziro.

Розглянемо типові конструкції для розгалуження і організації циклів. Організацію розгалужень у програмах на асемблері найкраще пояснити на прикладі. У наступному фрагменті програмного коду виконується перехід на мітку next при рівності нулю вмісту регістра ECX. Рівність нулю вмістру

регістру CX визначається за допомогою команди, що впливає на прапори AF, CF, OF, PF, SF і ZF. Отже приведемо приклад

```
; Организация конструкции ветвления (cmp)
        mov cx, 0
        cmp cx, 0
        jz next_z1
        ;обробка ситуації, коли CX не дорівнює O
        mov ah, 09h
        lea dx, [no_ziro]
        int 21h
        jmp next_2
next_z1:
        ;обробка ситуації, коли CX дорівнює O
        mov ah, 09h
        lea dx, [is_ziro]
        int 21h
next_2:
        mov ah, 09h
        lea dx, [out_of]
        int 21h
```

Як це працює. Якщо CX містить нульове значення, команда CMP установлює прапор нуля ZF в одиницю. Команда JZ перевіряє прапор ZF і, якщо він дорівнює 1, передає управління на на мітку next_z1. Після виконання коду, іде вихід з конструкції. В протилежному випадку виконуються код нижче і з використанням jmp next_2 іде вихід на зовні конструкції. Фактично даний фрагмент програмного коду реалізує логічну структуру if, з умовою CX=0.

Аналогічно можна реалізувати логічну структуру if з використанням команди TEST. Ця команда виконує операцію логічного "И" над двома операндами. Не змінює жодного з операндов й залежно від результату встановлює прапори SF, ZF і PF. При цьому прапори OF і CF скидаються, а прапор AF має невизначене значення. Приведемо приклад з вихідного коду.

```
        mov ax, 0
; Code organitation with TEST. Analog IF
        test ax, 10000000b
        jne next_z2
        ;Code for bit of AX IS 1
        mov ah, 09h
        lea dx, [no_ziro]
        int 21h
        jmp next_1
next_z2:
```

```
                ;Code for bit of AX IS 0
                mov ah, 09h
                lea dx, [is_ziro]
                int 21h
next_1:
                mov ah, 09h
                lea dx, [out_of]
                int 21h
```

Можливі інші модифікації циклів, але сутність залишається тією самою.

Організація циклів. Дуже часто умовні переходи використаються при програмуванні циклічних операцій, або циклів, коли обробляється група елементів. Кількість итераций (проходжень) у циклі найчастіше визначається кількістю елементів, хоча це й не обов'язково. Цикл може закінчитися в одному із двох випадків: при виконані всіх ітерації; коли виявлена умова, відповідно до якої повинен відбутися вихід із циклу. Розглянемо перший випадок у канонічному варіанті, користуючись фрагментом коду з прикладу. У результаті роботи цього коду масив array2Db заповнюється числами 3. Оскільки відносна адреса масиву відома і дорівнює 0, то до SI відразу записано 0.

```
; Типовая организация цикла, індексна адрес
                mov al, 03h
                mov cx, 100h ;Заповнення усього масиву
                mov si, 0
ptr_1:
                mov [ds:[si]], al
                inc si
                loop ptr_1
```

Як це працює. У прикладі використана канонічна організація циклу з використанням команди loop ptr_1. Ця команда буде переводити управління на мітку ptr_1, поки значення регістру CX не стане дорівнювати 0. Використана індексна адресація. У індексний регістр записано значення 0. До регістру CX записано значення 100h, тобто число повторень у циклі. Інших маніпуляцій с CX не потрібно, він буде зменшуватися на 1 у кожній ітерації циклу.В цьому полягає особливість регістру CX, яка властива тільки ньому.

```
        mov al, 05h
        mov cx, 100h ;Заповнення усього масиву
        mov si, 0
ptr_2:
```

```
mov [array2Db+si], al
inc si
loop ptr_2
```

У цьому прикладі до масиву записуються числа 5. Аналогічний приклад показаний вище, різниця тільки у способі застосування індексної адресації, де використовується значення початкової адреси масиву, для зручності і читабельності.

Наступна важлива група команд це строкові команди. Вони значно спрощують і пришвидшують процес запису великих обсягів інформації з однієї ділянки памяті до іншої. Це здійснюється без використання циклів. У складі команд процессора МП86 ця група команд, призначена для операцій з рядками символів або чисел тобто, з массивами даних. Таких команд усього 5:

omovs-пересилання рядка;

ocmps-порівняння рядків;

oseas - пошук у рядку заданного елемента (сканування рядка);

olods-завантаження з рядка регістрів AX або AL;

ostos-запис елемента рядка з регістрів AX або AL.

Команди мають загальні риси. Вони виконуються процесором у припущенні, що логічна адреса рядка-джерела визначається групою регістрів DS:SI, а логічна адреса рядка-приймача групою регістрів ES:DI. При однократному виконанні вони обробляють тільки один елемент масиву. Для обробки рядка повинні мати префікс повторення.У процесі обробки елементи регістри SI й DI автоматично зсуваються вперед (якщо прапор встановлений DF=0). Можуть зсуватися назад, якщо прапор DF=l має значення одиниці. Кожна команда має модифікації для роботи з байтами або словами (наприклад, movsb imovsw). Розглянемо особливості використання строкових команд на простих формальних прикладах з вихідного коду 2.1.

```
;Запис великих масивів без організації циклів
;Робота з рядками джерело (sourse) у DS:SI, приймач у  ES:DI

cld                     ; Обнулення прапору напрямку, прямий
lea si, [array2Db]      ; Завантаження відн. адреси джерела
```

```
lea di, [arr_dup2D]   ; Завантаження відн. адреси приймача
mov cx, 100h          ; Визначення кількості повторень
rep     movsb                 ; Завантаження
```

У цьому прикладі команда movsb використана із префіксом повторення rep (repeat, повторювати), що змушує процессор виконати команду movsb число раз, відповідно до змісту регістру CX. Ми бачимо, що перед використанням команди movsb треба виконати цілий ряд попередніх дій: помістити в регістри DS і ES сегментні адреси джерела й приймача, а в регістри SI й DI ефективну адресу тобто зміщення у регістрі; за допомогою командие LD (clear direction, скинути напрямок) скинути прапорпр оцесора DF; у регістр CX записати число байт для пересилання. Після цього одна команда movsb і з префіксом повторення rep виконує операцію переписування. Кількість переписаних у такий спосіб даних може дорівнювати 32К байт, якщо й джерело й приймач перебувають в одному сегменті. Це можна зробити з використанням циклічних операцій, але код значно спрощується.

Як відомо, для зручності роботи з більшою кількістю різнорідних файлів в DOS використається деревоподібна структура каталогів. Каталог являє собою файл, у якому є перелік всіх підкаталогів наступного рівня й файлів, що входять у даний каталог. Кожному підкаталогу або файлу в каталозі надається приділяється один елемент розміром 32байта, у який DOS заносить інформацію про файл, а саме: ім'я, початкову адресу на диску (номер кластера), дату й час створення файлу, довжину файлу у байтах, а також набір характеристик файлу - атрибутів. Крім цього кожний каталог містить інформацію про себе самому й батьківський каталог. При створенні нового файлу МПС під керівництвом DOS відшукує на диску вільне місце й призначає його новому файлу. Хоча мінімальною порцією інформації, переданої контролером диска є сектор, і переривання BIOS працюють саме із секторами, файлова система призначає місце на диску цілими кластерами. Розмір кластера на дискеті становить 12 Кб. В кластер можуть входити 4-8 секторів. Таким чином, мінімальний фізичний розмір файлу, становить один кластер.

В елементі каталогу вказується не фізична, а логічна довжина файлу, тобто обсяг даних, що записано у ньому вимірюється у байтах. Робота з файлами передбачає використання дескрипторів (файлових індексів). Це ідентифікатор на рівні операційної системі, для обслуговування файлів, що відкриті. Стандартна процедура читання-запису файлу в загальному випадку має таку послідовність виконання:

виклик переривання для відкриття або створення файлу;

робота з файлом, запис до файла або читання з файлу;

закриття файлу і звільнення всіх ресурсів, що були пов’язані з ним.

У більшості випадків робота з файлом починається з виконання операції його відкриття, для чого передбачена типове переривання DOS. Відкриваючи файл, DOS призначає йому черговий вільний запис у таблиці відкритих файлів System File Table (SFT). Знайшовши в системі каталогів диска файл, що відкривається, DOS заносить до SFT блок основніх характеристики файлу: ім'я, довжина, атрибути, дата й час створення, стартовий кластер, фізичну адресу. Частина інформаці записується в блок SFT з елементів каталогу, частина МПС під керівництвом DOS визначає сама. Важливим елементом блоку опису файлу є двухсловне значення покажчика, у якому зберігається покажчик-номер початку файлу. Саме з нього почнеться чергова операція запису або читання. Покажчиком можна керувати, це дозволяє організувати прямий доступ до файлу, тобто читання або запис починаючи з будь-якого місця файлу. Звертання до відкритого файлу (запис, читання, зміна характеристик файлу й т.д.) здійснюється по дескриптору. Під час виконання операцій з відкритим файлом DOS модифікує інформацію в блоці SFT. Зміст SFT завжди відбиває поточний стан файлу. При завершенні програми відоме вже нам переривання DOS 4Ch виконується автоматичне закриття всіх відкритих у програмі файлів.

Розглянемо кілька прикладів операцій запису й читання файлів на диску.

Переривання DOS  3Ch дозволяє  відкрити або створити новий файл для його роботи. Якщо функція 3Ch виявляє, що на диску вже є файл із зазначеним ім'ям, вона фактично знищує його й створює новий з ти мже ім'ям..

Переривання DOS  40h дозволяє записувати дані на будь-який пристрій, у тому числій у файл на диску. Конкретний приймач даних задається його дескриптором. Необхідо відмітити, що при записі у файл і при виводі інформації на будь-який пристрій, у приймач даних надходять лише ті дані, які зазначені в програмі. Жодних службових кодів, наприклад ознаки кінеця файлу, не записуються.

```
TITLE Vihidni kod 2.2
;-------------------------------------------------------------------------
;ЛР  №2.2 Кодування Кіріліца Windows-1251
;-------------------------------------------------------------------------
;---------------I.ЗАГОЛОВОК ПРОГРАМИ-----------------------------------------
IDEAL
MODEL SMALL
STACK 512
;----------------II.МАКРОСИ-----------------------------------------------
;2.2 Складний макрос для ініціалізації
MACRO M_Init                ; Початок макросу
mov    ax, @data     ; ax <- @data
mov    ds, ax          ; ds <- ax
mov    es, ax          ; es <- ax
ENDM M_Init          ; Кінець макросу

;--------------------III.ПОЧАТОК СЕГМЕНТУ ДАНИХ
DATASEG
;Оголошеня двовимірного експеріментального масиву 16x16
array2Db db  7, 8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8
        db  7, 7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,0 ,7
        db  7, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,7 ,7
```

```
        db  7, 8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,8 ,7 ,7

; Для вирівнювання у дампі
arr_def1 dw  3, 0 , 0 , 0 , 0 , 0 , 0 , 0
; Аналог двовимірного масиву
arr_dup2D db 0100h DUP(4)
; Для вирівнювання у дампі
arr_def11 dw  3, 0 , 0 , 0 , 0 , 0 , 0 , 0
arr_rnd1 db 2, 3, 0 , 2, 1, 9, 7, 2
variant db 3
; Для вирівнювання у дампі
arr_def2 dw  3, 0 , 0 , 0 , 0 , 0 , 0 , 0
no_ziro db "Variable is NOT ziro", 10, 13, '$'
is_ziro db "Variable IS ziro", 10, 13, '$'
out_of  db "out OF constr", 10, 13, '$'
arr_def6 dw  3, 0 , 0 , 0
exCode db 0

;----------------------VI. ПОЧАТОК СЕГМЕНТУ КОДУ-----------------
CODESEG
Start:
M_Init
;------ Stage 1. Write to array2Db[3,3] the arr_rnd1 [0]. Example of first part of Lab 2-

        mov si, 0               ;Index for 2D of array2Db, in gorizontal
        mov di, 0               ;Index for 1D of arr_rnd1, in gorizontal
        mov dl, [arr_rnd1+di]           ;DL <- arr_rnd1[0]
        mov al, 10h
        mov dh, [variant]
        mul dh                          ;AX<-(variant*10h)
        mov bx, ax              ;Index for 2D of array2Db, in vertical
        mov [array2Db+bx+si+3], dl

;---- Stage 2. Cycle 1 to horisontal. Example of first part of Lab 2-------------------------
        mov cx, 8               ;Counter for gorizontal cycle
        mov si, 0               ;Index   for 2D of array2Db, in gorizontal
        mov di, 0               ;Index   for 1D of arr_rnd1, in gorizontal
horisontal:
        mov dl, [arr_rnd1+di]           ;DL <- arr_rnd1[0]
        mov al, 10h
        mov dh, [variant]
        mul dh                          ;AX<-(variant*10h)
        mov bx, ax                      ;Index for 2D of array2Db, in vertical
        inc si                          ;Inc   for 2D of array2Db, in gorizontal
        inc di                          ;Inc   for 1D of arr_rnd1, in gorizontal
        mov [array2Db+bx+si+3], dl
loop horisontal

;---------- Stage 3. The refactoring. Example of first part of Lab 2----------------------
; we no need for di index!!!
        mov cx, 8
```

```
        mov si, 0               ;Index for 2D of [array2Db] and [arr_rnd1], in gorizontal

horisontal_1:
        mov dl, [arr_rnd1+si]       ;DL <- arr_rnd1[0]
        mov al, 10h
        mov dh, 5                   ;For testing refactoring
        mul dh                      ;AX<-(variant*10h)
        mov bx, ax                  ;Index for 2D of array2Db, in vertical
        inc si                  ;inc for 2D of array2Db, in gorizontal
        mov [array2Db+bx+si+3], dl
loop horisontal_1
;---------- Stage 4. The gorisontal and vertikal.Example of first part of Lab 2----------
        mov dh, [variant]   ;Coutrer of cicle vertical init
vertical_1:
        cmp dh, 11          ;The 11 is vertical coord of left bottom of array 8x8
        jz  main_exit       ;!!! MAIN EXIT
        mov al, 10h         ;It is simple mult for vertical coord of strings
        mul dh              ;AX<-(variant*10h), execute of mult, result in AX
        mov bx, ax          ;BX<-AX Index for [array2Db], in vertical coord
        mov cx, 8           ;Counter of cicle gorizontal init
        mov si, 0           ;Ziro to index of [array2Db] and [arr_rnd1] in gorizontal
horisontal_2:
        mov dl, [arr_rnd1+si]       ;DL <- [arr_rnd1[si]]
        inc si                      ;inc index of [array2Db] in gorizontal
        mov [array2Db+bx+si+3], dl ;si and 3 are connect to gorizontal coord in 8x8
loop horisontal_2

        inc dh              ;inc coord vertical
        jmp vertical_1      ;come back
main_exit:


;------------------Code organitation with CMP. Analog IF-------------------------
        mov cx, 0
        cmp cx, 0
        jz next_z1
;Code for CX NOT ziro
        mov ah, 09h
        lea dx, [no_ziro]
        int 21h
        jmp next_2 ; go to end of constuct IF
next_z1:
                ;Code for CX IZ ziro
        mov ah, 09h
        lea dx, [is_ziro]
        int 21h
next_2:
        mov ah, 09h
        lea dx, [out_of]
        int 21h
```

```
        mov ax, 0
;--------------------Code organitation with TEST. Analog IF-----------------
        test ax, 10000000b
        jne next_z2
                ;Code for bit of AX IS 1
        mov ah, 09h
        lea dx, [no_ziro]
        int 21h
        jmp next_1
next_z2:
                ;Code for bit of AX IS 0
        mov ah, 09h
        lea dx, [is_ziro]
        int 21h
next_1:
        mov ah, 09h
        lea dx, [out_of]
        int 21h

;Запис великих масивів без організації циклів
;Робота з рядками джерело (sourse) у DS:SI, приймач у  ES:DI

        cld                     ; Обнулення прапору напрямку, прямий
        lea si, [array2Db]      ; Завантаження відн. адреси джерела
        lea di, [arr_dup2D]     ; Завантаження відн. адреси приймача
        mov cx, 100h            ; Визначення кількості повторень
        rep     movsb                   ; Завантаження


; Типовая организация цикла, індексна адрес
        mov al, 03h
        mov cx, 100h ;Заповнення усього масиву
        mov si, 0
ptr_1:
        mov [ds:[si]], al
        inc si
loop ptr_1

; Індексна адресація через ідентіфікатор масиву, індексна
        mov al, 05h
        mov cx, 100h ;Заповнення усього масиву
        mov si, 0
ptr_2:
        mov [array2Db+si], al
        inc si
loop ptr_2

Exit:
    mov ah,4ch
    mov al,[exCode]
```

```
int 21h
end Start
```

Методичні вказівки щодо проведення роботи. У якості прототипу обирається вихідний. Створюється повний цикл виконання програми і видаляються вісі зайві елементи коду, що не потрібні у ЛР.

Далі обирається потрібна координата делянки масиву 8х8 і, використовуючи потрібний вид адресації, пишіться код, що дає можливість заповнити потрібну ділянку коду, цифрами, що вимагаються.

Виконується повний цикл створення програми і виправляються помилки, що були виявлені. Для підтвердження результатів Turbo Debugger фотографується ділянка дампу пам’яті масиву.

Робляться висновки щодо практичного підтвердження теоретичних положень, що викладені у методичної розробки .

Перелік питань для підготовки до лабораторної роботи

1. Як написати .bat файл для спрощення і пришвидшення асемблювання і компонування програми. Поясніть на прикладі коду.

2.* Перелічіть регістри загального призначення і сегментні регістри МПС Intel 8086. Розкрийте поняття розрядності регістрів, призначення, особливості регістрів.

3.* Дайте поняття ефективної адреси, зміщення у сегменті, логічної адреси, фізичної адреси операнду. Як визначається фізична адреса операнду?

4.* Як здійснюється сегментація пам’яті МПС на базі Intel 8086, як визначається фізична адреса операндів. Який може бути максимальний і мінімальний розмір сегменту?

5.*Команда mov у архітектурі Intel 8086. Дайте поняття операнду. Які операнди команди mov застосовуються при різних способах адресації? Поясніть на прикладі коду.

6.*Як можна визначити адресу початку сегментів з використанням TD ?

7.* Синтаксис описання змінної, чисельного масиву, масиву символів у асемблері. Поясніть на прикладі коду. Як визначити фізичну адресу змінної, початку масиву з використанням TD?

8.*Поняття програмного переривання, переривання int 21h, як використовується це переривання для виводу змінної на консоль? Поясніть на прикладі коду.

9.*Основні команди для роботи зі стеком. Які регістри використовуються для управління стеком? Призначення і використання цих регістрів в різних способах адресації. Поясніть на прикладі коду.

# ЛАБОРАТОРНА РОБОТА 6

# СИСТЕМА ОБРОБКИ ПЕРЕРИВАНЬ АРХІТЕКТУРІ IA-32 (X86) У REAL ADRESS MODE

Завдання на лабораторну роботу.

1. Написати з використанням асемблеру функцію обробки переривання, що виводить на консоль номер групи і прізвище авторів.

2. Перепризначити цю функцію на вектор переривань, що відповідає номеру, який дорівнює 50 плюс номер робочої групи.

3. Повернути МПС у вихідний стан.

Дослідити механізм адресації МПС під час передачі управління до функції обробки переривань і механізм адресації МПС під час повернення з неї. Користуючись TD, під час покрокового виконання програми, визначити логічні адреси у сегменті коду у перелічені далі моменти:

у ході поточної операції безпосередньо перед викликом процедури (функції);

на початку роботи з процедурою (функцією) обробки переривань;

безпосередньо у час виходу з процедури функції.

Визначити зміст стеку в ці моменти. Зробити «фотографування екранів» для документації результатів. У якості методичних рекомендацій використати фрагмент коду і теоретичні відомості що надані далі.

```
%TITLE Власний оброблювач переривання COM1
;--------------------------------------------------------------------------
; Дисципліна:   Системне програмування
;КНУУ "КПІ"
; Факультет:    ФІОТ
; Курс:
; Група:
;--------------------------------------------------------------------------
; Автор:
; Дата:
;--------------------------------------------------------------------------
IDEAL
;--------------------------------------------------------------------------
MACRO M_Exit        ; Вихід з програми
                    ; На входе:     AL = код завершення програми
```

```
                        ; На виході:    ---
        mov     ah, 04Ch ; Номер вектора переривання DOS для виходу
        int     21h     ; Виклик переривання
ENDM

;-------------------------------------------------------------------------
MACRO M_Init        ; Ініціалізація DS і ES

        mov     ax, @data       ; ax <- @data
        mov     ds, ax          ; ds <- ax
        mov     es, ax          ; es <- ax
ENDM

MODEL  small
STACK  256
DATASEG
bak_int0Bh_offset     DW ?  ; Ефективна адреса функції - стандартного
                            ; обробника апаратного переривання COM1
bak_int0Bh_seg        DW ?  ; Адреса початку сегменту
                            ; функції апаратного переривання COM1
CODESEG
PROC main
M_Init                      ; макрос ініціалізації

;ЕТАП I. Отримання еф. адреси і зміщення переривання.--------------------
                        ; Отримання ефективної адреси і зміщення
                        ; стандартного обробника переривання COM1 для його заміни
                        ; Це виконується з використанням функції GetIntVector.
mov     di, 0Bh         ; Вхідний параметр функції GetIntVector.
                        ; Це типовий номер переривання 0Bh для COM1.
call    GetIntVector    ; Виклик.
                        ; Вихідні аргументи процедури (bx - еф. адреса, es сег. адреса)
                        ; Збереження у змінних для повернення ст. обробника переривання
mov     [bak_int0Bh_offset], bx         ; Зберігаємо значення еф. адреси у змінній
mov     [bak_int0Bh_seg], es            ; Зберігаємо значення адреси сегменту у змінній
;ЕТАП II. Збереження стандартного обробника переривань COM 2  за іншим вектором
                        ; Вільні вектора 60h - 6Bh
                        ; Перенесення апаратного переривання COM 2 на наш вектор - 62h
                        ; Використаємо функцію SetIntVector
                        ; DI  Вхідний аргумент - номер вектора, куди переносимо.
                        ; DX  Вхідний аргумент - еф. адреса процедури яку переносимо
                        ; ES  Вхідний аргумент - адреса сегмента процедури яку переносимо.
mov     di, 62h         ; Вхідний аргумент SetIntVector – новий вектор DI для COM1
mov     dx, bx          ; Нове зміщення процедури DX (еф. адреса)
                        ; Новий сегмент ES для процедури той же
call    SetIntVector    ; Виклик. Переносимо стандартну функцію обробки COM2 на вектор
-62h

  ;ЕТАП III. Визначення нового обробника переривання з дод. функціоналом на вектор 0Bh.
                        ; На вектор int0Bh заносимо новий функціонал і частину старого COM 2
                        ; Підготовка аргументів
mov     di, 0Bh                 ; DI Вхідний аргумент - номер старого вектора пер.COM
mov     dx, OFFSET int0Bh       ; DX Вхідний аргумент - еф. адреса нової проц.
mov     ax, SEG int0BhES        ; ES Вхідний аргумент - адреса сегмента нової проц.
mov     es, ax                  ; Завантажуємо таки ES. Оскільки
                                ; mov    ES, SEG int0BhES не дозволено
```

```
call    SetIntVector            ; Виклик.

                                ; Виконується інша частина програми
                                ; Виконується інша частина програми
                                ; Виконалася інша частина програми

; ЕТАП IV. Повернення переривання INT 0Bh у вихідний стан.
mov     di, 0Bh
mov     dx, [bak_int0Bh_offset]
mov     ax, [bak_int0Bh_seg]
mov     es, ax
call    SetIntVector
xor     al, al                  ; код  <0>
M_Exit                          ; Макрос для виходу з ОС
ENDP main                       ;Закінчення функції main

;//////////////////////////////Описання процедур (функцій)\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
;-------------------------------------------------------------------------
PROC GetIntVector
; Призначення: Отримання логічної адреси процедури (функції) обробки переривання за
; номером вектора переривання
; Вхід:         DI  <- номер вектора переривання
; Вихід:        BX  <- Ефективна адреса процедури (функції) обробки переривання
;               ES  <- Адреса сегмента процедури (функції) обробки переривання
;-------------------------------------------------------------------------
; Збереження стану регістрів
push    ax
push    di
xor     ax, ax          ; 0-> AX
mov     es, ax          ; Перехід на початок сегменту 0000h
shl     di, 2           ; Множимо DI на 4 (зв'язок номера переривання і адреси)
mov     bx, es:[di]     ; Ефективну адресу функції обробника в BX
mov     ax, es:[di + 2] ; Адресу сегменту функції обробника в AX
mov     es, ax          ; Адресу сегменту функції обробника таки до ES
                        ; Відновлення задіяних регістрів
pop     di
pop     ax
ret                     ; Не забуваємо повернутися з процедури (функції)
ENDP GetIntVector
;-------------------------------------------------------------------------
PROC SetIntVector
; Призначення: Установка на номер вектора нової функції обробника.
; Функціонально: до пам'яті що відповідає вектору заносимо ефективну адресу і адресу
; сегменту нового обробника переривань
; На вхід:      DI  - номер переривання де буде нова процедура
;               DX  - ефективна адреса нового обробника
;               ES  - адреса сегмента нового обробника
; На вихід: ---
;-------------------------------------------------------------------------
 Cli                    ; Заборона апаратних переривань
                        ; Наприкінці обов'язково <STI> бо залишаться забороненими - их
                        ; МПС буде працювати не правильно
                        ; <CLI ... STI> - це критична ділянка коду, яку не можна переривати
                        ; Збереження регістрів
push    ax
push    di
```

```
push    ds
xor     ax, ax              ; очистка AX
mov     ds, ax              ; переходимо до адреси 0000h
shl     di, 2               ; множимо DI на 4
mov     ds:[di], dx         ; Еф. Адресу до першої частини вектору
mov     ds:[di + 2], es     ; Адресу сегменту до другої частини вектору
                            ; Відновлення регістрів
pop     ds
pop     di
 pop     ax

sti                         ; Дозвіл переривань. Відновлення вихідного стану МПС.
ret                         ; Не забуваємо…
ENDP SetIntVector

;--------------------------------------------------------------------------
PROC int0Bh
; Призначення: Нова процедура (функція) обробника переривання для COM.
; На вхід:     ---
; На вихід:    ---
;--------------------------------------------------------------------------

int     62h;                ; Виклик стандартного обробника,
                            ; він вже перепризначений на вектор 62.
;       ...                 ; код розширення функціоналу
;       ...                 ; стандартного обробника

                            ; код закінчення АППАРАРАТНОГО переривання
        mov     al, 20h
        out     20h, al
        iret                ; Вихід з функції обробника відрізняється від звичайної процедрури
    ENDP int0Bh

END main                    ; Кінець коду
```

Переривання це такий вид процедур (функцій), запуск яких здійснюється асинхронно під виконання звичайних прикладних програм. В деяких випадках час їх запуску розробник не може передбачити. Це відноситься до апаратних переривань і вони реалізуються через певні пристрої МПС.

В деяких випадках переривання викликаються програмно і можуть бути обумовлені синтаксично у вихідному коді. У цьому випадку переривання називаються програмними і реалізуються через операційну систему або у BIOS. Відповідно переривання розподіляються на програмні і апаратні.

Апаратні переривання пов’язані з певним обладнанням. Вони викликаються сигналом від цього обладнання, наприклад: переривання від

клавіатури, таймеру, процесору, тощо. Апаратне переривання не залежить від програми, що виконується і може зупинити її в будь-який момент. Для цього апаратні переривання повинні бути дозволеними. У МПС дозвіл переривань визначається станом прапора IF. Якщо він піднятий до 1, то переривання дозволені, інакше апаратні переривання не дозволені. Існують дві команди, що керують цим прапором. Команда асемблеру cli обнуляє прапор IF, забороняючи переривання, команда sti встановлює прапор у 1.

Під час виклику апаратного переривання, програма що виконувалася зупиняється, зберігається в стеку основна інформація про початковий стан МПС у ході її виконання. Далі управління автоматично передається на процедуру (функцію) обробки переривання. Кожній процедурі відповідає своє переривання, що ідентифікуєтеся за допомогою номера. Після виконання переривання МПС повинна повернутися у вихідний стан. Для цього логічна адреса поточної команди (CS:IP) автоматично записується до стеку разом з змістом регістру прапорів. Далі до пари регістрів МПС CS:IP автоматично завантажується нова логічна адреса процедури (функції) обробки переривання. Ця логічна адреса містить 4 байта (32 біта). Вона звичайно зберігається у початковій ділянці ОП і називається вектором переривання. Кожному апаратному перериванню з номером відповідає свій вектор переривання. Після виконання функції обробки переривання управління повертається на той крок програми, що виконувалася. Про повернення з переривання докладно буде описано далі.

Процедура (функція) обробки переривання звичайно закінчується командою IRET (повернення з переривання). Ця команда повертає МПС на той рядок коду, у якому була програма була до початку виникнення переривання. Для цього зі стеку повертаються і завантажуються вихідні адреси до пари регістрів CS:IP. Повертається зі стеку регістр прапорів вихідного стану програми.

Аналогічно організовані програмні переривання. Відмінність їх полягає в тому, що вони не викликаються апаратною частиною МПС. Найбільш відомими є переривання DOS, наприклад INT21h. Це переривання було використано у ЛР1, ЛР2. Незалежну від ОС групу програмних переривань для PC IBM сумісних комп'ютерів утворюють переривання BIOS, наприклад INT09h.

Механізм виклику і роботи програмних переривань аналогічний апаратним. Відмінність полягає в тому, що програмні переривання викликаються безпосередньо з асемблерного коду. Функції обробки переривань вбудовані до ОС або до BIOS. Частково ці функції пов'язані з апаратними подіями, але виклик програмних переривань здійснюється з асемблерного коду. МПС автоматично забороняє виконання переривань при роботі функції обробки переривання. Після закінчення МПС повертається у вихідний стан.

Функції (процедури) обробки переривань можуть бути переписані і переназначені за потребою програмістом.

Ділянки з логічними адресами функцій обробки програмних переривань також називають векторами. Кожний вектор має довжину 4 байта. У першому слові (2 байта) зберігається значення ефективної адреси початку функції обробки переривання (IP), у другому адреса сегменту - CS. Молодші 1024 байт ОП містять вектори переривань, у тому числі апаратні. Таким чином зарезервовано і використовується 256 векторів переривань. В цілому утворюється таблиця векторів переривань МПС, що завантажується з ОС. Вектор для переривання 0 починається з адреси 0000:0000 и закінчується адресою 0000:0003, переривання 1 починається з адреси 0000:0004 і закінчується 0000:0008 тощо. Адреса F000:FEA5. відповідає стартовій адресі ПЗУ, яка відповідає перериванню 8H (BIOS).

Для управління апаратними перериваннями IBM PC сумісні комп'ютери використовують мікросхему контролера переривань Intel 8259 або більш сучасні аналоги. Мікросхема Intel 8259 для упорядкування роботи має 8

апаратних входів і 8 рівнів пріоритетів апаратних переривань. Кожний апаратний вхід контролера переривань Intel 8259 являє собою одиночний провідник. На цей провідник подається відповідний сигнал з пристрою МПС. Наприклад, на апаратні входи контролера переривань Intel 8259 подаються: переривання від клавіатури, таймеру, COM порту, тощо. Контролер переривань Intel 8259 має 8 апаратних входів, що нумеруються IRQ0 - IRQ7. Дві поєднані мікросхеми Intel 8259 утворюють розширення діапазону переривань IRQ0 - IRQ15. Максимальний пріоритет апаратного переривання відповідає рівню 0. Звичайно кожному апаратному входу (IRQ0 - IRQ7) відповідає свій вектор переривань у таблиці переривань ОП. Апаратним входам контролера Intel 8259 (IRQ0 - IRQ7) відповідають вектора 8H-0FH у таблиці переривань. Далі надані приклади векторів та їх пріоритети:

IRQ 0 таймер;

IRQ 1 клавіатура;

IRQ 2 канал вводу/вивода;

IRQ 8 годинник реального часу (тільки AT) ;

IRQ 9 програмно переводяться в IRQ2 (AT) ;

IRQ 10 резерв;

IRQ 11 резерв;

IRQ 12 резерв;

IRQ 3 COM1 (COM2 для AT) ;

IRQ 4 COM2 (модем для PCjr, COM1 для AT) ;

Мікросхема 8259 має три однобайтних регістра, що керують апаратними перериваннями. Вони доступні програмно через вихідний код Асемблеру.

Розробник звичайно перед викликом переривання перевіряє регістр дозволу переривання, або маски переривання (IMR). Це здійснюється щоб узнати, чи дозволено відповідне переривання. Ці переривання називають переривання за маскою, і можуть заборонятися або дозволятися програмно.

Звичайно розробники звертаються до регістру маски переривань (IMR) через порт 21H і. До командного регістру Intel 8259 через порт 20H.

Програми на Асемблері шляхом звернення до регістру маски можуть забороняти переривання. Це апаратні переривання називаються перериваннями за маскою. Інші апаратні переривання не можуть бути забороненими. Переривання за маскою забороняються в тому випадку, коли вони можуть мати вплив на критичні ділянки коду.

Регістр маски переривань другої мікросхеми 8259 (IRQ8-15) має адресу A1H. В коді далі приведений приклад заборони переривань.

```
;--- заборона переривання 6.
  MOV   AL,01000000B   ;підготуємо регістр, заборонівші 6 переривання
  OUT   21H,AL         ;записуємо до регістру маски
;повернення у вихідний стан
  MOV   AL,0           ;
  OUT   21H,AL         ;
```

Відомо декілька причин написання власного обробника переривань. По-перше для зручності і зменшення обсягу коду. По-друге для реалізації функції обробки нового апаратного переривання для управління нестандартним обладнанням МПС. По-третє доробити або повністю переписати процедуру обробки переривання, що запрограмована у ОС.

Розглянемо простий приклад розробки переривання з використанням функції DOS. Вихідний код апаратного переривання наданий далі. Відзначимо, що функція 25H автоматично забороняє переривання, що налаштовані на цей вектор під час її роботи. Тому не потрібно робити додаткових команд для захисту критичних ділянок коду.

```
;--- встановлення нової процедури для вектора переривання  60H
  PUSH  DS
  MOV   DX,OFFSET ROUT ; Ефективну адресу нової процедури до  DX
  MOV   AX,SEG ROUT    ;Сегмент процедури до AX
  MOV   DS,AX          ;Такі розміщаємо до DS
  MOV   AH,25H         ;Функція DOS
  MOV   AL,60H         ;номер вектора
```

```
  INT   21H                    ;Виклик функції DOS
  POP   DS          ;Відновлення DS

;--- процедура обробки переривання 60H
ROUT  PROC  FAR
    PUSH  AX
                              ; Функція корисної роботи
    POP   AX
    MOV   AL,20H     ;Ці два рядка для апаратних переривань
    OUT   20H,AL
    IRET
ROUT  ENDP
```

Два рядка, що надані далі дають можливість мікросхемі 8259 очистити регістри обслуговування для дозволу переривань з іншими пріоритетами.

```
MOV   AL,20H
OUT   20H,AL
```

Після завершення програми доцільно повернути вектора переривань у вихідний стан. Це особливо важливо для векторів переривань ОС. Якщо цього не зробити, то операційна система буде працювати не так як вона задумана, що може привести до краху ОС. Приклад повернення таблиці векторів переривань і відповідних їм функцій у вихідний стан показаний у фрагменті коду далі.

```
;--- оголошуємо у сегменті даних дві змінні для збереження адреси сегменту і
;ефективної адреси:
  KEEP_CS  DW    0       ;
  KEEP_IP  DW    0
;--на початку програми отримаємо значення вектору і зберігаємо їх.
        MOV   AH,25H     ;функція отримання вектору
        MOV   AL,1CH     ;номер вектору
         INT   21H       ;виклик функції, після її роботи в ES сегмент функції обробки
переривання,
    У BX ефективна адреса функції обробки переривання
        MOV   KEEP_IP,BX ;запам'ятовуємо все у змінні
        MOV   KEEP_CS,ES ;
; --- приведення таблиці векторів і відповідних функцій у вихідний стан
        CLI          ; Заборона переривань
```

```
PUSH  DS        ;
MOV   DX,KEEP_IP ;підготовка до відновлення
MOV   AX,KEEP_CS ;
MOV   DS,AX      ;сегмент даних
MOV   AH,25H     ;функція відновлення
MOV   AL,1CH     ;номер вектору, що відновлюється
INT   21H        ;виклик функції, що відновлюється
POP   DS
STI              ; Дозвіл переривань
```

# ЛАБОРАТОРНА РОБОТА 7

# ПІДПРОГРАМИ АРХІТЕКТУРИ IA-32 (X86) У REAL ADRESS MODE

Завдання на лабораторну роботу. Лабораторна робота передбачає проведення експерименту, що дозволяє розкрити механізм виклику підпрограм і описати використання команд call, ret. Визначення стану регістрів при виконанні підпрограм, стану пам'яті, стеку. Подальша звірка фрагментів лістингу програми і поточного стану МПС з метою розкриття механізму виклику підпрограм і використання пам'яті. Для цього необхідно написати програму, що реалізує текстовий інтерфейс і підпрограми. Завдання.

Таблиця 2.

Сполучення букв для розробки інтерфейсу користувача

| Функції | Номер варіанту | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Літера для виклику функції обчислення виразу | q | r | y | i | a | G | l | c | n | . | 2 | 5 | 8 | e | t | u | i | o |
| Літера для виклику функції включення звуку (тривалість звучання с) | W 1 | T 2 | U 3 | O 4 | S 5 | H 4 | Z 3 | V 2 | M 1 | / 2 | 3 4 | 6 7 | 9 8 | D 9 | H 1 | J 3 | K 4 | L 5 |
| Літера для виходу з програми | e | y | i | p | d | k | x | b | , | 1 | 4 | 7 | 0 | c | n | m | , | . |
| Літера для пошуку найбільшого значення *(парний варіант)* | | a | | b | | c | | d | | f | | g | | h | | r | | o |
| Літера для пошуку найменьшого значення *(непарний варіант)* | q | | w | | e | | t | | r | | y | | u | | o | | p | |

1. $((a_1+a_2)*a_3/a_4+a_5)$ $a_1=-7, a_2=3, a_3=2, a_4=4, a_5=1$

2. $((a_1+a_2)*a_3/a_4+a_5)$ $a_1=-7, a_2=3, a_3=2, a_4=4, a_5=2$

3. $((a_1+a_2)*a_3/a_4+a_5)$ $a_1=-7, a_2=3, a_3=2, a_4=4, a_5=3$

4. $((a_1-a_2)*a_3*a_4+a_5)$ $a_1=-1, a_2=1, a_3=2, a_4=2, a_5=3$

5. $((a_1-a_2)*a_3*a_4+a_5)$ $a_1=-1, a_2=2, a_3=1, a_4=2, a_5=3$

6. $((a_1-a_2)*a_3*a_4+a_5)$ $a_1=-1, a_2=1, a_3=1, a_4=2, a_5=3$

7. $(((a_1-a_2)+a_3)/a_4*a_5)$ $a_1=-2, a_2=3, a_3=1, a_4=2, a_5=3$

8. $(((a_1-a_2)+a_3)/a_4*a_5)$ $a_1=-2, a_2=3, a_3=1, a_4=2, a_5=3$

9. $(((a_1-a_2)+a_3)/a_4*a_5)$ $a_1=-2, a_2=3, a_3=1, a_4=2, a_5=3$

10. $(a_1-a_2*a_3/a_4+a_5)$ $a_1=-6, a_2=3, a_3=2, a_4=2, a_5=1$

11. $(a_1-a_2*a_3/a_4+a_5)$ $a_1=-6, a_2=3, a_3=2, a_4=2, a_5=1$

12. $(a_1-a_2*a_3/a_4+a_5)$ $a_1=-6, a_2=3, a_3=2, a_4=2, a_5=1$

13. $(((a_1-a_2)/a_3 - a_4)*a_5)$ $a_1=-3, a_2=3, a_3=2, a_4=1, a_5=2$

14. $(((a_1-a_2)/a_3 - a_4)*a_5)$ $a_1=-3, a_2=3, a_3=2, a_4=1, a_5=2$

15. $(((a_1-a_2)/a_3 - a_4)*a_5)$ $a_1=-3, a_2=3, a_3=2, a_4=1, a_5=2$

16. $(((a_1+a_2)/a_3 + a_4) - a_5)$ $a_1=-8, a_2=4, a_3=2, a_4=1, a_5=1$

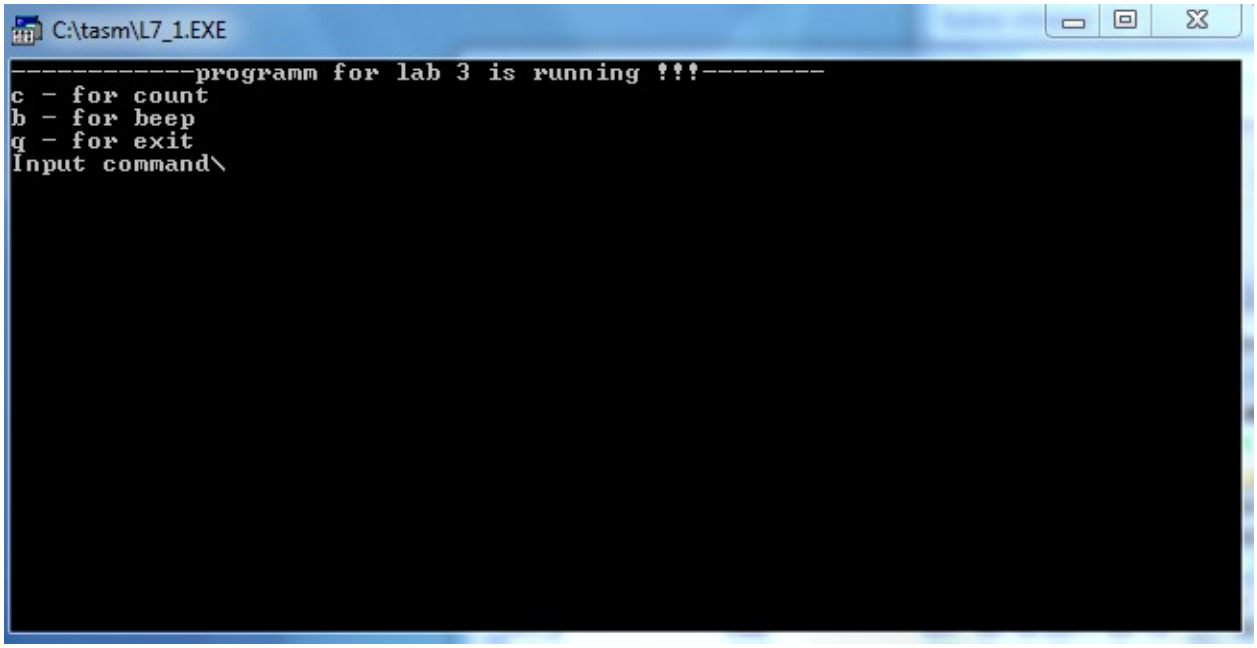


Рисунок - 18 Інтерфейс програми.

Сполучення букв у інтерфейсі користувача потрібно змінити, відповідно до варіантів завдань, що надані у таблиці. Написати функцію для розрахунку виразу, що викликається при натискуванні відповідної клавіші на клавіатурі. Вирази надані вище. Методика і програма проведення експерименту надана у відповідних підрозділах. Студент повинен читати і пояснювати свій розроблений код програми.

Програма проведення експерименту для першої частини лабораторної роботи. Реалізується у декілька етапів, що надані далі. Програма реалізована для Асемблеру TASM. У якості спрощеного прототипу для виконання лабораторної роботи пропонується вихідний код, що наданий нижче. Він дає можливість сформувати консольну програму з текстовим інтерфейсом користувача.

```
IDEAL
MODEL   small
STACK   256

DATASEG
      string  db      254     ;string variable def. There is max len ,
      str_len db      0       ;под запись реально
                              ; числа символов: str_len
      db 254 dup ('*')        ; Буфер


      system_message_1   DB "Input somthing\  " ,'$'

      display_message_0  DB "------------menu bagin--------", 13, 10, '$'
      display_message_1  DB "c - for count",   13, 10,  '$'
      display_message_3  DB "q - for exit",    13, 10,  '$'
      display_message_4  DB "----------programm for lab is END !!! - for exit", 13, 10,  '$'
      display_message_5  DB "------------menu end--------", 13, 10, '$'
      message  DB ?

      test_message_1  DB "COUNT",         13, 10,  '$'
      test_message_3  DB "EXIT",          13, 10,  '$'

CODESEG

Start:
    mov ax, @data
    mov ds, ax

Main_cycle:
```

```
        call display

        call input_foo

        cmp ax, 063h ; c ascii =63h
        je Count
        cmp ax, 071h ; q ascii =71h
        je  Exit
        jmp Main_cycle

    Count:
        mov dx, offset test_message_1
        call display_foo  ; any foo
        jmp Main_cycle


    Exit:
      mov dx, offset display_message_4
      call display_foo
      mov ax,04C00h         ;
      int 21h               ; прерывания DOS
     ;-------------------------------------------------
    PROC display
        mov dx, offset display_message_0
        call display_foo
        mov dx, offset display_message_1
        call display_foo
        mov dx, offset display_message_3
        call display_foo
        mov dx, offset system_message_1
        call display_foo
        mov dx, offset display_message_5
        call display_foo
        ret
    ENDP display_foo_main
    ;-------------------------------------------------
    PROC display_foo; input dx is offset
        mov ah,9
        int 21h
        xor dx, dx
        ret
    ENDP display_foo
    ;-------------------------------------------------
    PROC input_foo              ; input string  out ax
       mov ah, 0ah              ; ah <- 0ah input
       mov dx, offset string    ; dx <- offset string
       int 21h                  ; call  0ah function DOS int 21h

        xor ax, ax
        mov bx, offset string
        mov ax, [bx+1]
        shr ax, 8
        ret
    ENDP input_foo

END Start
```

У головному циклі програми відбувається відображення текстового меню програми. Далі викликається процедура, що виводить до регістру AX код натиснутої клавіші. Після цього у конструкції вибору визначається логіка програми.

Створюємо програму, користуючись вихідним кодом, що наданий далі.

```
;-----------------------------------------------------------------
;Програма під час початку роботи виводить повідомлення.
;В залежності від обраного повідомлення вона :
;1. Викликає функцію для видачі звуку.
;2. Викликає функцію для обчислення результату виразу
; (в програмі це не реалізовано, потрібно доопрацювати)
;3. Забезпечує вихід з програми
; Під час реалізації функцій
; Програма читає зі стандартного вводу (клавіатури) строку.
; Максимальна довжина строки 254 символу, строку можна зберігати у ;файлі (в програмі це не реалізовано).
; Розроблено на кафедрі АУТС 3.10.2011 року.
;-----------------------------------------------------------------

        IDEAL
        MODEL   small
        STACK   256

        DATASEG

    string  db      254     ;змінна для строки - string,
    str_len db      0       ;
    db 254 dup ('*')        ; Буфер заповнюється '*' для ;кращого налаштування
;----- Змінні для виводу системних команд
system_message_1   DB "Input command and press enter\  "  ,'$'
system_message_2   DB "Program end"       ,'$'
;----- Змінні для виводу команд під час управління програмою
display_message_0  DB "------------programm for lab 3 is running !!!--------", 13, 10, '$'
display_message_1  DB "c - for count",   13, 10,  '$'
display_message_2  DB "b - for beep",    13, 10,  '$'
display_message_3  DB "q - for exit",    13, 10,  '$'
display_message_4  DB "----------programm for lab is END !!! --------------", 13, 10,  '$'
display_message_5  DB "Press any key for beep --------------", 13, 10,  '$'
;--- Змінні що використовувалися під час налаштування програми
message  DB ?
  ;test_message_1  DB "!!! count DISPLAY",      13, 10,  '$'
  ;test_message_2  DB "!!! beep DISPLAY",       13, 10,  '$'
  ;test_message_3  DB "q - for exit",           13, 10,  '$'
;------Константи  для функції звуку
     NUMBER_CYCLES  EQU   2000
     FREQUENCY      EQU   600
     PORT_B                 EQU   61H
     COMMAND_REG    EQU   43H      ; Адреса командного регістру
```

```
    CHANNEL_2     EQU   42H           ; Адреса каналу 2
    simvol  db ?

  CODESEG
   Start:
    mov ax, @data           ;
    mov ds, ax
Main_cikle: ; Основний цикл програми для інтерфейсу користувача
;---------------------------------------------
call display_foo_main
;---------------------------------------------
    mov ah, 0ah             ; ah <- 0ah
    mov dx, offset string   ; пересилка в dx начала буфера
    int 21h
xor ax, ax
 mov bx, offset string      ;пересилка в bx начала буфера для
                           ;реалізації адресації зі зміщенням
 mov ax, [bx+1]             ;занесення в ax чисельного значення
                           ;символу ASCII, що відповідає
                           ;знаку,
                           ;який введено з клавіатури

 shr ax, 8                 ;зсув в  регістрі ax для виконання
                           ;cmp
cmp ax, 063h ; c ascii =63h   ; Вибір відповідної функції
je Count                   ; На лекції 3!!!
cmp ax, 062h ; b ascii =62h
je Beep
cmp ax, 071h ; q ascii =71h
je  Exit
jmp Main_cikle
;---------------------------------------------------------
Count:
   ; mov dx, offset test_message_1 ; Закоментовані повідомлення
                           ; у ході налаштування
   ; call display_foo           ; тут повинна викликатися
                           ; функція для обчислення
                           ; виразу і виведення
                           ; результату на консоль
   ; any foo for counte
jmp Main_cikle
;---------------------------------------------------------
Beep:
   ; any foo for sound          ; виклик функції звуку
  mov dx, offset display_message_5
  call display_foo
  call  zvukF1
  jmp Main_cikle
;---------------------------------------------------------
; Стандартний вихід з програми
  Exit:
    mov dx, offset display_message_4
    call display_foo
  mov ah,04Ch         ;
  int 21h             ;
;------------SUB-1 display_foo_main-------------------
```

```
PROC display_foo_main
    mov ah, 0               ;Функція відображає інтерфейс
                        ;користувача
    mov al, 3
    int 10h
    mov dx, offset display_message_0
    call display_foo
    mov dx, offset display_message_1
    call display_foo
    mov dx, offset display_message_2
    call display_foo
    mov dx, offset display_message_3
    call display_foo
    mov dx, offset system_message_1
    call display_foo
  ret
ENDP display_foo_main
;-------------SUB-2 display_foo---------------------
PROC display_foo
    mov ah,9
    int 21h
    xor dx, dx
  ret
ENDP display_foo
;-------------SUB-3 zvukF1---------------------
PROC  zvukF1

lab2:

        int 16h             ; Зберігає отримане значення з клавіатури  в змінній
        mov [simvol],al       ; simvol
        cmp [simvol],'e'      ; Перевірка на відповідність і встановлення прапору ознаки 0
        jz Exit             ;
                        ; Перехід на Exit: у випадку відповідності

;Встановлення частоти 440 гц

      ;--- дозвіл каналу 2 встановлення порту  В мікросхеми 8255
                IN   AL,PORT_B     ;Читання
                OR   AL,3               ;Встановлення двох молодших бітів
                OUT  PORT_B,AL     ;пересилка байта в порт В мікросхеми 8255

     ;--- встановлення регістрів порту вводу-виводу
                 MOV  AL,10110110B ;біти для каналу 2
                 OUT  COMMAND_REG,AL  ;байт в порт командний регістр

     ;--- встановлення лічильника
                MOV  AX,2705                ;лічильник = 1190000/440
                OUT  CHANNEL_2,AL    ;відправка AL
                MOV  AL,AH       ;відправка старшого байту в AL
                OUT  CHANNEL_2,AL    ;відправка старшого байту

     ;--- виклик преривання з клавіатури для зупинки
                MOV  AH,8         ;номер функції преривання 8
                INT  21H              ;виклик преривання
```

```
    ;--- виключення звуку
            IN   AL,PORT_B     ;отримуємо байт з порту B
            AND  AL,11111100B  ;скидання двох молодших бітів
            OUT  PORT_B,AL     ;пересилка байтів в зворотному напрямку
        ret
    ENDP zvukF1

    END Start
```

Опишемо програму більш докладно. На початку роботи програма викликає функцію *display_foo_main*. Вона призначена для виводу повідомлення на консоль для користувача, див. код нижче.

```
CODESEG
  Start:
    mov ax, @data          ;
    mov ds, ax

Main_cikle: ; Основний цикл програми для інтерфейса користувача
;---------------------------------------------
call display_foo_main
;---------------------------------------------
```

Функція display_foo_main показана нижче.

```
;------------SUB-1 display_foo_main------------------

PROC display_foo_main
    mov ah, 0              ;Функція відображає інтерфейс
                           ;користувача
    mov al, 3
    int 10h
    mov dx, offset display_message_0
    call display_foo
    mov dx, offset display_message_1
    call display_foo
    mov dx, offset display_message_2
    call display_foo
    mov dx, offset display_message_3
    call display_foo
    mov dx, offset system_message_1
    call display_foo
  ret
ENDP display_foo_main
```

Як можна побачити з наданого вище коду, на першому етапі здійснюється очищення екрану, далі послідовно викликається функція *display_foo*, що дає можливість вивести на екран відповідне повідомлення, передача параметрів в функцію здійснюється через регістр *dx*.

Код функції *display_foo* наданий нижче.

```
;-------------SUB-2 display_foo---------------------
    PROC display_foo
    mov ah,9
    int 21h
    xor dx, dx
  ret
  ENDP display_foo.
```

Після виведення повідомлення виконується основний цикл програми. Його код наданий далі.

```
Main_cikle: ; Основний цикл програми для інтерфейса користувача
;-----------------------------------------
call display_foo_main
;-----------------------------------------
    mov ah, 0ah             ; ah <- 0ah
    mov dx, offset string    ; пересилка в dx начала буфера
    int 21h

 xor ax, ax
 mov bx, offset string       ;пересилка в bx начала буфера для
                             ;реалізації адресації зі зміщенням
 mov ax, [bx+1]              ;занесення в ax чисельного значення
                             ;символу ASCII, що відповідає
                             ;знаку,
                            ;який введено з клавіатури

 shr ax, 8                  ;зсув в  регістрі ax для виконання
                            ;cmp

cmp ax, 063h ; c ascii =63h     ; Вибір відповідної функції
je Count                        ; cmp ax, 062h ; b ascii =62h
je Beep
cmp ax, 071h ; q ascii =71h
je  Exit
jmp Main_cikle
;-------------------------------------------------------
Count:
       ; any foo for counte
jmp Main_cikle
;-------------------------------------------------------
Beep:
      mov dx, offset display_message_5
      call display_foo
     call  zvukF1
     jmp Main_cikle
;-------------------------------------------------------
; Стандартний вихід з програми
  Exit:
```

```
 mov dx, offset display_message_4
call display_foo
mov ah,04Ch
int 21h
```

Аналізуючи вихідний код можна зробити висновок, що в залежності введеного з клавіатури знаку (c, b або q) викликаються відповідні функції. При наборі на клавіатурі і натискуванні Enter – здійснюється розрахунок виразу (в шаблоні програми його немає, його потрібно розробити відповідно завданню лабораторної роботи). При наборі b і натискуванні Enter – викликається звук динаміка, при наборі і натискуванні Enter q здійснюється вихід з програми.

Для управління звуком потрібно використати засоби управління звуком. Для цього використовується діапазон адрес управління пристроями у МПС.

Після описанного виникає запитання, як здійснюється управління пристроями МПС, наприклад перефінійними пристроями, контролером переривань, годинником реального часу, тощо. Саме для цього крім оговореної вище памяті у МПС існує окрема ділянка і діапазон, що не належить до ОЗП. Фізична реалізація цей ділянки Вона призназивається простором портів вводу-виводу. Наведена нижче таблиця містить розподіл адресного простору портів вводу-виводу для IBM PC/XT. При цьому:

- порти 0-FFh відведені для системної плати;
- порти 100h-3FFh відведені для контролерів пристроїв;
- порти, починаючи з 400h, недоступні для системної шини.

Таблиця 3

| AT/PS-2 | PC/XT | Опис |
|---|---|---|
| 000-01F | 000-00F | Контролер ПДП N 1, 8237A-5 |
| 020-03F | 020-021 | Контролер переривань N 1, 8259A |
| 040-05F | 040-043 | Таймер (PC / XT: 8253-5, AT: 8254-2) |
| | 060-063 | Програмований інтерфейс периферії 8255 |
| 060-06F | | Контролер клавіатури AT 8042 |
| 070-07F | | Пам'ять CMOS і маска NMI |
| 080 | | Діагностичний регістр |

| | | |
|---|---|---|
| 080-08F | 080-083 | Регістри сторінок ПДП 74LS612 |
| 090-097 | | Блок управління каналами PS/2 |
| | 0A0 | Маска NMI |
| 0A0-0BF | | Контролер переривань N 2, 8259A |
| 0C0-0DF | | Контролер ПДП N 2, 8237A-5 |
| 0F0-0FF | | (N2 при MGA) |
| 380-38F | 380-38F | Синхронні адаптери: SDLC або BSC N 2 |
| 3A0-3AF | 3A0-3A9 | Синхронний адаптер BSC N 1 |
| 3B0-3BF | 3B0-3BF | Монохромний адаптер (MGA) + принтер N 1 |
| 3C0-3CF | 3C0-3CF | Розширений графічний адаптер (EGA) N 1 |
| 3D0-3DF | 3D0-3DF | Кольоровий графічний адаптер (CGA) и EGA |
| 3F0-3F7 | 3F0-3F7 | Контролер НГМД N 1 |
| 3F8-3FF | 3F8-3FF | Стик RS-232 N 1 |

Отже програмне управління пристроями МПС здійснюється через порти вводу-виводу. Доступ до кожного порта контролера може бути здійснений через свої порти вводу-виводу.

Приклад управління портом вводу-виводу і формування звуку показаний у вихідному коді, що наданий далі.

```
        IDEAL
        MODEL   small
        STACK   256

MACRO   delay     time
        local   outer
        push    cx
        mov     cx, time
outer:                  ;зовнішній цикл
        push    cx
        mov     cx, 0FFFFh
        loop    $               ;внутрішній цикл, стрибати на
                        ;       місці cx раз (9,10)
        pop     cx              ;відновлення cx (11)
        loop    outer           ;cx <-cx-1 і, якщо cx<>0, то перехід outer,
        pop     cx
ENDM
```

```
        DATASEG

_100Hz  DW     11930          ;вихідне значення лічильника ПТ
len_snd DW     400            ;тривалість звучання

        CODESEG
Start:
  ;ініціалізація регістра ds
        mov     ax,@data
        mov     ds,ax
;програмування таймера
        mov     al, 0B6h
        out     43h, al        ;РУС ПТ(порт 43h)<-al (2)
;завантаження буферного регістра 2 таймера
        mov     ax, [_100Hz]    ;пересилання Nпоч.= 11930=2E9Ah в ax (3)
        out     42h, al        ;порт 42h<-МБ11930=9Ah (3)
        mov     al, ah          ;al<-ah=СБ11930=2Eh (3)
        out     42h, al        ;порт 42h<-СБ11930=2Eh (3)
;ввімкнення каналу 2 таймера та динаміка
        in      al, 61h        ;
        or      al, 3          ;al<-(al\/03h) (4)
        out     61h, al        ;порт 61h<-al (4)

        delay   [len_snd]      ;встановлення затримки (див. macros.mac)
        and     al, 11111100b   ;маска вимикання звуку (15)
        out     61h, al        ;порт 61h<-al (15)
        mov     ax,04C00h      ;ax<-04C00h (16)
        int     21h            ;виклик переривання DOS 21h (16)
        END Start              ;(17)
```

Для переписування змісту стеку до нового масиву можна використати базову адресацію або використати команди управління стеком.

ЛАБОРАТОРНА РОБОТА 8

АРХІТЕКТУРА IA-32 (X86) У REAL ADRESS MODE

Основне завдання дослідження полягає у розробки програми засобами Асемблер, що реалізує псевдографічний інтерфейс, який показано на рис.19. Під час розробки необхідно додати у меню три додаткових кнопки.


Рисунок - 19 . Меню для вихідного коду файлу *STRMENU.asm.*

При натискуванні першої кнопки на екран виводиться номер робочої групи, ім'я учасників групи - латиницією з використанням переривань DOC.

Далі потрібно повторити функціональність лабораторної роботи 7(див.лаб.раб. 7).

Програма надається в якості одного з варіантів роботи. Можлива своя реалізація програми експерименту.

**Етап 1.** Розробляється і налагоджується програма, що реалізує простий текстовий інтерфейс – меню.

**Етап 2.** Розробляються функції що задані. Здійснюється їх тестування.

**Етап 3.** Функції додаються до програми - інтерфейсу.

Далі у якості прикладу і методичних рекомендацій дається програма, що реалізує інтерфейс наданий у завданні. Під час лабораторної роботи дається можливість реалізувати інтерфейс за власним шаблоном.

```
TITLE     STRMENU (EXE)
        .MODEL SMALL
        .STACK 64
        .DATA
;-----------------------------------------------------------------------
TOPROW    EQU   08        ;Верхній рядок меню
BOTROW    EQU   15        ;Нижній рядок меню
LEFCOL    EQU   26            ;Лівий стовпчик меню
ATTRIB    DB    ?         ; Атрибути екрану
ROW       DB    00        ;Рядок екрану
SHADOW    DB    19 DUP(0DBH);
MENU      DB    0C9H, 17 DUP(0CDH), 0BBH
        DB    0BAH, ' Add records    ',0BAH
        DB    0BAH, ' Delete records  ',0BAH
        DB    0BAH, ' Enter orders    ',0BAH
        DB    0BAH, ' Print report    ',0BAH
        DB    0BAH, ' Update accounts ',0BAH
        DB    0BAH, ' View records    ',0BAH
        DB    0C8H, 17 DUP(0CDH), 0BCH
PROMPT    DB    'To select an item, use <Up/Down Arrow>'
        DB    ' and press <Enter>.'
        DB    13, 10, 'Press <Esc> to exit.'
.386  ;---------------------------------------------------------------
        .CODE
A10MAIN   PROC  FAR
        MOV   AX,@data
        MOV   DS,AX
        MOV   ES,AX
        CALL  Q10CLEAR        ; Очистка екрану
        MOV   ROW,BOTROW+4
A20:
        CALL  B10MENU         ;Вивід меню
        MOV   ROW,TOPROW+1      ;Вибір верхнього пункту меню
                                    ;   у якості початкового значення
        MOV   ATTRIB,16H        ;Переключення зображення в інв..
        CALL  D10DISPLY         ;Відображення
```

```
        CALL   C10INPUT         ;Вибір з меню
        JMP    A20              ;
A10MAIN    ENDP
;--------------------------------------------------------------------------
;    Вивід рамки, меню і запрошення…
;--------------------------------------------------------------------------
B10MENU    PROC   NEAR
        PUSHA                   ;
       MOV    AX,1301H          ;
        MOV    BX,0060H         ;
        LEA    BP,SHADOW        ;
        MOV    CX,19            ;
        MOV    DH,TOPROW+1      ;
        MOV    DL,LEFCOL+1      ;
B20:      INT    10H
   ;;;;;
        INC    DH               ;Наступний рядок
        CMP    DH,BOTROW+2      ;
        JNE    B20              ;
        MOV    ATTRIB,71H       ;
        MOV    AX,1300H         ;
        MOVZX  BX,ATTRIB        ;
        LEA    BP,MENU          ;
        MOV    CX,19
        MOV    DH,TOPROW        ;Рядок
        MOV    DL,LEFCOL        ;Стовпчик
B30:
        INT    10H
        ADD    BP,19            ;
        INC    DH               ;
        CMP    DH,BOTROW+1      ;
        JNE    B30              ;
        MOV    AX,1301H         ;
        MOVZX  BX,ATTRIB        ;
        LEA    BP,PROMPT        ;
        MOV    CX,79            ;
        MOV    DH,BOTROW+4      ;
        MOV    DL,00            ;
        INT    10H
        POPA                    ;,
        RET
```

```
B10MENU    ENDP
;--------------------------------------------------------------------------
;  Натискування клавиш, управління через клавиші і  ENTER
;  для вибору пункту меню і клавіші ESC для виходу
;--------------------------------------------------------------------------
C10INPUT   PROC    NEAR
        PUSHA                   ;
C20:      MOV     AH,10H          ;Запитати один символ з кл.
        INT     16H             ;
        CMP     AH,50H          ;Стрілка до низу
        JE      C30
        CMP     AH,48H          ;Стрілка до гори ?
        JE      C40
        CMP     AL,0DH          ;Натистнено  ENTER?
        JE      C90
        CMP     AL,1BH          ;Натиснено  ESCAPE?
        JE      C80             ;  Вихід
        JMP     C20             ;Жодна не натиснена, повторення
C30:
        MOV     ATTRIB,71H      ;Кольор символів
        CALL    D10DISPLY       ;
        INC     ROW             ;
        CMP     ROW,BOTROW-1    ;
        JBE     C50             ;
        MOV     ROW,TOPROW+1    ;
        JMP     C50
C40:
        MOV     ATTRIB,71H      ;Кольор символів і екрану
        CALL    D10DISPLY       ;
                        ;
        DEC     ROW
        CMP     ROW,TOPROW+1    ;
        JAE     C50             ;
        MOV     ROW,BOTROW-1    ;
C50:
        MOV     ATTRIB,17H      ;Кольор символів
        CALL    D10DISPLY       ;
        JMP     C20
C80:
        MOV     AX,4C00H
        INT     21H
```

```
C90:
        POPA
        RET
C10INPUT    ENDP
;-------------------------------------------------------------------------
;  Забарвлення виділеного рядка
;-------------------------------------------------------------------------
D10DISPLY  PROC    NEAR
        PUSHA
        MOVZX   AX,ROW
        SUB     AX,TOPROW
        IMUL    AX,19
        LEA     SI,MENU+1
        ADD     SI,AX
        MOV     AX,1300H
        MOVZX   BX,ATTRIB
        MOV     BP,SI
        MOV     CX,17
        MOV     DH,ROW
        MOV     DL,LEFCOL+1
        INT     10H
        POPA
        RET
D10DISPLY  ENDP
;-------------------------------------------------------------------------
;    Очищення екрану
;-------------------------------------------------------------------------
Q10CLEAR   PROC    NEAR
        PUSHA
        MOV     AX,0600H
        MOV     BH,61H
        MOV     CX,00
        MOV     DX,184FH
        INT     10H
        POPA
        RET
Q10CLEAR   ENDP
        END     A10MAI
```

Коротко розглянемо роботу програми. Якщо натискуємо кнопку клавіатури “стрілка до гори”, курсор зсувається вверх, елемент меню

виділяється синім кольором, як показано на рисунку. Якщо кнопка не відпущена, рух курсору здійснюється з низу до гори, циклічно. При натискуванні кнопки клавіатури “стрілка до низу”, здійснюються таки самі функції, тільки циклічно з верху до низу. При натискуванні Enter здійснюється виділення відповідного елементу меню. При натискуванні Esc здійснюється вихід з програми. Після виходу з програми кольори екрану залишаються такими самими.

Скелет вихідного коду програми *STRMENU.asm.* показаний далі.

```
TITLE
        .MODEL SMALL
        .STACK 64;
.DATA
....
    .CODE
A10MAIN    PROC   FAR
          .....
          CALL   Q10CLEAR          ;очистка екрану
 A20:     ...
          CALL   B10MENU           ; виклик меню
         ...
          CALL   D10DISPLY         ;відображення елементів
        ...
          CALL   C10INPUT          ;управління меню
          JMP    A20               ;основний цикл програми
A10MAIN    ENDP
;---------------------------------------------------------------------
Відображення меню
;---------------------------------------------------------------------
B10MENU    PROC   NEAR
          PUSHA                    ;збереження регістрів
          POPA                     ;відновлення регістрів
          RET                      ; повернення з  підпрограми
B10MENU    ENDP
;---------------------------------------------------------------------
Управління програмою
;---------------------------------------------------------------------
C10INPUT   PROC    NEAR
          PUSHA                    ;
C20:       MOV     AH,10H   ;
          INT     16H              ;
          CMP     AH,50H
          JE      C30
          CMP     AH,48H           ;
          JE      C40
          CMP     AL,0DH           ; Натиснута кнопка ENTER?
```

```
        JE      C90
        CMP     AL,1BH          ; Натиснута кнопка ESCAPE?
        JE      C80             ; якщо натиснута, то вихід з підпрограми
       JMP      C20             ;якщо ні,  повернення в цикл підпрограми
C30:
.....
C40:
....
C50:
....
C80:
....
C90:
        POPA
        RET
C10INPUT    ENDP
;-------------------------------------------------------------------------
Відображення елементів меню
;-------------------------------------------------------------------------
D10DISPLY  PROC    NEAR
        RET
D10DISPLY  ENDP
;-------------------------------------------------------------------------
Очистка і налаштування екрану
;-------------------------------------------------------------------------
Q10CLEAR   PROC    NEAR

        RET
Q10CLEAR   ENDP
        END     A10MAIN
```

Основний цикл програми представлений підпрограмою *A10MAIN.* Як можна побачити з коду, в основному циклі *A10MAIN* викликаються по черзі чотири підпрограм. Перша реалізує очистку екрану.

```
        .....
        CALL   Q10CLEAR         ;очистка екрану
```

Далі зверніть увагу на мітку, що призначена для основного циклу програми *A20*. Вона визначає межі основного циклу програми, в якому реалізуються три основних підпрограми.

```
 A20:    ...
        CALL   B10MENU          ; виклик меню
       ...
        CALL   D10DISPLY        ;відображення елементів
      ...
        CALL   C10INPUT         ;управління меню
        JMP    A20              ;основний цикл програми
```

Після реалізації всіх підпрограм управління повертається на мітку A20. Вихід з програми здійснюється через C10INPUT, про що буде докладніші пояснено далі.

Підпрограми B10MEN, D10DISPLY призначені для відображення меню на екрані і їх можна вивчити з використанням файлу STRMENU.asm. Важливим елементом програми є підпрограма управління програмою C10INPUT, що викликається у головному циклі. Розглянемо її більш докладно. В ній застосовано переривання DOS INT 16H, що реалізує ввід з клавіатури сигналу (символу). Далі здійснюється по черзі перевірка введеного символу (натиснутої клавіші). В залежності від натиснутої клавіші реалізується різний код. Наприклад, якщо натиснута клавіша ESCAPE, то здійснюється перехід на C80: і вихід з програми, без зміни налаштування екрану (див. на фрагмент коду, що наданий далі).

```
Управління програмою
;-------------------------------------------------------------------------
C10INPUT   PROC    NEAR
        PUSHA                ;
C20:      MOV     AH,10H   ;
        INT     16H            ;
        CMP     AH,50H
        JE      C30
        CMP     AH,48H          ;
        JE      C40
        CMP     AL,0DH          ; Натиснута кнопка ENTER?
        JE      C90
        CMP     AL,1BH          ; Натиснута кнопка ESCAPE?
        JE      C80               ; якщо натиснута, то вихід з підпрограми
       JMP     C20             ;якщо ні,  повернення в цикл підпрограми
C30:
.....
C40:
....
C50:
....
C80:
....
```

```
C90:
        POPA
        RET
C10INPUT    ENDP
```

Дозволяється реалізовувати свій варіант реалізації програми. При власної реалізації обсяг коду повинен збільшуватися  не більше ніж на 100 % відносно прототипу.

Робота прототипу, меню, що реалізується, показано на рис.20. З цього рисунку можна побачити повідомлення, які виводяться під час нормальної роботи програми і показанні на рис.20 чорною стрілкою.


Рисунок -  20  Результати роботи програми

# 3 ОФОРМЛЕННЯ ЗВІТУ ТА ПОРЯДОК ЙОГО ПОДАННЯ

Звіт може представлятися у електронній формі або у роздрукованому вигляді. Звіт бажано подавати без затримок. Затримка не повинна перевищувати одну лабораторну роботу. У випадку більшої затримки оцінка знижується на 1 бал. Загальні вимоги щодо оформлення і захисту лабораторних робіт надані далі.

При виконанні роботи на Асемблері необхідною умовою захисту лабораторної роботи є:

наявність файлу з *вихідним кодом* ;

у вихідному коді коментарі з прізвищем, номером групи, номером лабораторної роботи;

наявність файлу з лістингом програми;

здатність студента зробити транслювання файлу вихідного коду;

здатність студента зробити компонування програми;

здатність студента здійснити покроковий запуск програми під час налаштування, при цьому студент повинен пояснити змінні в пам’яті програми під час кожного кроку; пояснити призначення кожної інструкції. У випадку необхідності до програми додаються фотографії екранів і відповідні обчислення, що обумовлено у кожній лабораторної роботі.

Код повинен містити необхідні коментарі, його дозволено використовувати під час захисту роботи. У випадку помилок при асемблюванні програми або при її запуску робота до захисту не допускається.

При виконанні роботи на С необхідною умовою захисту лабораторної роботи є:

наявність файлів з вихідним кодом, вихідний код повинен містити всі потрібні коментарі;

у основному файлі програми коментарі з прізвищем, номером групи, номером лабораторної роботи;

здатність студента зробити компіляцію проекту;

здатність студента здійснити покроковий запуск програми під час налаштування, поставити точки зупинки і налагоджувати програму;

необхідно пояснити значення змінних.

У випадку помилок при компіляції проекту або при його запуску робота до захисту не допускається.

Додаткові вимоги:

для кожної функції повинен бути прототип з коротким описом параметрів;

під час оформлення роботи забезпечити форматування коду, не допускати неналежного переносу рядків коментарів;

особливості, що виявлені під час реалізації, невідповідність виявленого функціоналу типових функцій з описом у літературі, тощо.

Оцінка знижується на 2 бала за відсутність коментарів до роботи.

## 4. СПИСОК ЛІТЕРАТУРИ

1. И. Рудаков, К.Г Фіногенов. Программируем на языке Ассемблера IBM PC. – М.: Горячая линия – Телеком, 2003. – 316 с.

2. Магда Ю.С. Ассемблер для процесоров Intel Pentium.–Спб:. Питер, 2006–410 с.

3. Т. Сван.Освоение Turbo Assembler: Пер. с англ. – 2-е изд. – К.: М.: СПб.: Диалектика, 1996.– 544 с.