В. П. Семеренко, В. А. Каплун

Системне програмування

мовою Асемблера

частина 2

Міністерство освіти і науки України
Вінницький національний технічний університет

В. П. Семеренко, В. А. Каплун

Системне програмування мовою Асемблера

Затверджено Вченою радою Вінницького національного технічного університету як лабораторний практикум (частина 2) для студентів напрямів підготовки 0915 - "Комп'ютерна інженерія" та 1601 – "Інформаційна безпека" всіх спеціальностей. Протокол № 4 від 25 листопада 2004 р.

Вінниця ВНТУ 2004

УДК 681.3.06.(075)
С31

Рецензенти:

С.В. Юхимчук, доктор технічних наук, професор
А.М. Петух, доктор технічних наук, професор
А.І. Кузьмичов, кандидат технічних наук, доцент

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Семеренко В. П., Каплун В. А.

С31 **Системне програмування мовою Асемблера.** Лабораторний практикум. Частина 2. - Вінниця: ВНТУ, 2004.- 93 с.

В практикумі розглянута методика складання програм мовою Асемблера для мікропроцесорів фірми Intel підвищеної складності. Вивчаються функції транслятора з мови Асемблера та компонувача, аналізується структура їх лістингів. Велика увага приділена вивченню принципів програмного керування дисплеєм, математичним співпроцесором та пристроєм "миша". Розглянуті також макрозасоби в мові Асемблера. Практикум призначений для студентів напрямів підготовки 0915 - "Комп'ютерна інженерія" та 1601 – "Інформаційна безпека" для вивчення дисциплін "Системне програмування", "Системне програмне забезпечення", "Операційні системи" а також може бути рекомендований студентам інших спеціальностей, пов'язаних з вивченням сучасного програмного забезпечення.

УДК 681.3.06.(075)

ЗМІСТ

Вступ

Друга частина лабораторного практикуму орієнтована на продовження практичного вивчення програмування мовою Асемблера для мікропроцесорів фірми Intel.

В першій лабораторній роботі розглядаються макроси в мові Асемблера, які є особливим різновидом підпрограм, що дозволяють скоротити обсяг асемблерних програм.

Наступні дві роботи присвячені вивченню двох основних системних програм обробки – транслятора Turbo Assembler (TASM) і компонувача TLINK. Знання особливостей їх функціонування дозволяє створювати ефективні багатомодульні програми і здійснювати їх швидке налагоджування.

Опанувавши програмування математичного співпроцесора можна використовувати арифметичні операції зі всіма типами даних, включаючи числа із плаваючою комою. В результаті значно розширюється сфера застосування Асемблера.

Дві лабораторні роботи пов'язані із програмним керуванням двох важливих периферійних пристроїв: дисплея та "миші". Вивчивши графіку на Асемблері в текстовому та графічному режимах роботи дисплея можна легко формувати графічні зображення будь-якої складності з максимально можливою швидкодією. Це дасть змогу створювати програми з динамічними зображеннями, які часто використовуються в різноманітних ігрових та навчальних програмах. Використання в таких програмах пристрою "миша" є обов'язковим.

Велику користь програмісту надасть також вміння програмувати роботу з файлами в різних режимах.

Практично засвоївши лабораторні роботи, які розглянуті в посібнику, можна створювати достатньо складні асемблерні програми на професійному рівні.

ЛАБОРАТОРНА РОБОТА № 5
Макрозасоби мови Асемблера

Тема: Макрозасоби мови Асемблера. Директиви повторення, умовні директиви. Макробібліотеки.

Теоретичні відомості

2.5.1 Призначення макрозасобів

Програмування мовою Асемблер з використанням виключно машинних команд являє собою програмування мовою низького рівня, оскільки кожна команда виконує досить елементарні дії.

За допомогою макрозасобів можна ввести в програму свої власні команди (макрокоманди), які за своїми можливостями можуть бути порівняні з командами мов високого рівня. Програма в цьому випадку стає коротшою і зрозумілішою.

Машиноорієнтована мова, в якій використовуються макрозасоби (макровизначення, макрокоманди, макробібліотеки і таке інше), називається макромовою або Макроасемблером.

2.5.2 Макровизначення і макрокоманди

Припустимо, що ми маємо два фрагменти програми:

Фрагмент 1:

```
MOV   AX, A
ADD   AX, B
SUB   AX, C
```

Фрагмент 2:

```
MOV   AX, M
ADD   AX, N
SUB   AX, P
```

Якщо ввести формальні параметри ARG1, ARG2, ARG3, тоді можна написати одну узагальнену програму:

```
MOV   AX, ARG1
ADD   AX, ARG2
SUB   AX, ARG3
```

Така узагальнена програма може бути оформлена у вигляді макровизначення.

Макровизначення – це записана за чітко визначеними правилами послідовність машинних команд Асемблера, якій присвоєно ім'я.

Загальний формат макровизначення:

```
Ім'я макровизначення MACRO [Список формальних параметрів]
        Тіло макровизначення
ENDM
```

Для нашого прикладу макровизначення буде мати такий вигляд:

```
CALC    MACRO   ARG1, ARG2, ARG3
        MOV     AX, ARG1
        ADD     AX, ARG2
        SUB     AX, ARG3
ENDM
```

У тому місці програми, де повинна виконуватись послідовність команд, яка розміщена у макровизначенні, записується макрокоманда. Її формат:

Ім'я макровизначення [Список фактичних параметрів]

Для нашої програми приклади макрокоманд:

```
CALC    A, B, C
CALC    M, N, P
```

Ім'я макрокоманди повинно збігатись з іменем макровизначення, до якого звертається макрокоманда.

У макрокоманді фактичні параметри записуються у тому самому порядку, в якому перераховані формальні параметри у відповідному макровизначенні.

У середині макровизначення може бути розташоване звернення до іншої макрокоманди, яка в цьому випадку носить назву вкладеної макрокоманди.

Під час трансляції програми макрокоманда заміняється командами з відповідного макровизначення. Цей процес називається макророзширенням.

2.5.3 Приклад програми з використанням макровизначень

Наведена нижче програма демонструє використання макросів, які зменшують обсяг програми і надають їй більшої наглядності.

```
NAME    MYPROG

KARETKA MACRO   ARG ; Макровизначення для роботи з кареткою
        MOV     AH, 02h
        MOV     DL, ARG
        INT     21h
ENDM

OUT_NUMB MACRO      ; Макровизначення для виведення цифри
        MOV     AH, 06h
        ADD     AL, 48
        MOV     DL, AL
        INT     21h
ENDM
```

```
OUT_STR  MACRO          ; Макровизначення для виведення рядка
         MOV      AH, 09h
         INT      21h
ENDM
STACKSG  SEGMENT STACK
         DB   128 DUP(?)        ; резервування пам'яті для стеку
STACKSG  ENDS
DATE_SG  SEGMENT
     A   DB   2
     B   DB   3
     C   DB   4
     str0 DB  "Data:  A=2; B=3; C=4.$"
     str1 DB  "Rezult 1: Y = A+B+C = $"
     str2 DB  "Rezult 2: Y = A*B-C = $"
DATE_SG ENDS
CODE_SG  SEGMENT
     ASSUME  CS:CODE_SG, DS:DATE_SG, SS:STACKSG
START    PROC      FAR
     PUSH     DS
     SUB      AX,AX
     PUSH     AX
     MOV      AX,DATE_SG
     MOV      DS,AX
     CALL     MAIN
     RET
START    ENDP
MAIN     PROC
     KARETKA 0Ah  ; Переведення каретки
     MOV      DX, OFFSET str0
     OUT_STR
     KARETKA 0Ah  ; Переведення каретки
     KARETKA 0Dh  ; Повернення каретки
     MOV      DX, OFFSET str1
     OUT_STR
     MOV      AL, A
     MOV      BL, B
     ADD      AL, BL
     ADD      AL, C
     OUT_NUMB
     KARETKA 0Ah  ; Переведення каретки
     KARETKA 0Dh  ; Повернення каретки
     MOV      DX, OFFSET str2
```

```
        OUT_STR
        MOV     AL, A
        CBW
        IMUL    B
        SUB     AL, C
        OUT_NUMB
        KARETKA 0Ah   ; Переведення каретки
        RET
MAIN    ENDP
CODE_SG ENDS
END     START
```

У наведеній програмі створено три макроси. Макрос OUT_NUMB використовується для виведення на екран однієї цифри, значення якої знаходиться у регістрі AL. Макрос OUT_STR призначений для виведення на екран рядка символів, причому перед викликом цього макросу необхідно занести в регістр DX адресу символьного рядка. Макрос KARETKA використовується для роботи з кареткою (переведення на новий рядок, повернення каретки на початок рядка). У ньому використовується параметр ARG, який дозволяє змінювати призначення цього макроса. Так, якщо значення аргументу дорівнюватиме 0Ah, то відбудеться переведення каретки на новий рядок, 0Dh – повернення каретки, 08h – переведення на один символ назад, 07h – подача звукового сигналу тощо.

2.5.4 Проблема міток у макровизначеннях

В одній програмі одне й те саме макровизначення може бути використано декілька разів. Якщо макровизначення містить мітку, то при його багаторазовому використанні у відповідних макророзширеннях з'являються оператори з однаковими мітками. З точки зору мови це є синтаксичною помилкою.

Для запобігання дублювання міток вони описуються за допомогою директиви (оператора) LOCAL, наприклад:

```
LOCAL   MET1, MET2
```

У цьому випадку транслятор замість вказаної мітки записує число, а при кожному зверненні до макровизначення це число збільшується на 1.

2.5.5 Циклічні макроси (директиви повторення)

Директиви повторення вимагають від асемблера повторити блок операторів, що завершується директивою ENDM. Ці директиви не обов'язково повинні знаходитись в макровизначенні, але якщо вони там знаходяться, то

одна директива ENDM потрібна для завершення блоку, який повторюється, а друга директива ENDM – для завершення макровизначення.

Директива REPT приводить до повторення блока операторів (команд або інших директив) до директиви ENDM у відповідності з кількістю повторень, вказаних у виразі

REPT вираз

Наприклад, якщо записати

```
REPT    3
INC     CX
ENDM
```

то команда INC CX виконається тричі:

```
INC  CX
INC  CX
INC  CX
```

За допомогою директиви IRP можна при кожному повторенні використовувати різні параметри:

IRP формальний параметр, <список>.

У списку параметри можуть бути символами, рядками, числовими або арифметичними константами.

Наприклад, якщо макрокомандою

PUSH_REGS <AX, BX, CX, DX>

викликається макровизначення

```
PUSH_REGS   MACRO   LIST
            IRP     REG, <LIST>
            PUSH    REG
            ENDM
ENDM
```

це означатиме, що виконуються такі команди:

```
PUSH    AX
PUSH    BX
PUSH    CX
PUSH    DX
```

Як видно з наведеного прикладу, асемблер здійснює стільки проходів тіла макровизначення, скільки вказано елементів у списку. І при кожному такому проході асемблер підставляє замість формального параметра наступний за порядком елемент у списку.

Якщо список параметрів містить символи, тоді можна застосувати директиву повторення IRPC, формат якої такий:

IRPC формальний параметр, рядок символів

Наприклад, якщо записати

```
IRPC    CHAR, ABCD
ADD     AX, CHAR&X
ENDM
```

то виконається така група команд

```
ADD     AX, AX
ADD     AX, BX
ADD     AX, CX
ADD     AX, DX
```

В даному прикладі використовується символ "&" (амперсанд), який вказує асемблеру на необхідність конкатенації символів.

2.5.6 Умовні макроси (умовні директиви)

Виконання макровизначення може бути поставлено у залежність від виконання певних умов, що задаються директивою IFxx:

```
IFxx  вираз умови
...

ELSE
...

ENDIF
```

Якщо задана умова виконується, то транслюються лише оператори, розташовані до директиви ELSE, у протилежному випадку – ті оператори, що записані після директиви ELSE.

У середині умови макровизначення можна використовувати також директиву EXITM, яка достроково завершує виконання макровизначення.

Так само, як і директиви повторення, умовні директиви не обов'язково повинні знаходитися у макровизначенні.

У таблиці 1 наведено перелік різних умовних директив.

Нижче наведено приклад умовної директиви, яка викликає макровизначення ERROR у випадку, коли опущено деякий параметр m:

```
IFB     <m>
ERROR
EXITM
ENDIF
```

Як видно з цього прикладу, директива ELSE може бути відсутньою.

Таблиця 1 – Умовні директиви

Директива	Умова виконання директиви
IF вираз	Якщо вираз не дорівнює 0
IFE вираз	Якщо вираз дорівнює 0
IFDEF ім'я	Якщо ім'я визначено в програмі або об'явлено в директиві EXTRN
IFNDEF ім'я	Якщо ім'я не визначено в програмі або не об'явлено в директиві EXTRN
IFB <arg>	Якщо *arg* дорівнює пропуску (аргумент повинен бути взятий у кутові дужки)
IFNB <arg>	Якщо *arg* не дорівнює пропуску (аргумент повинен бути взятий у кутові дужки)
IFIDN <arg1>, <arg2>	Якщо рядки *arg1* та *arg2* збігаються (аргументи повинні бути взяті у кутові дужки)
IFDIF <arg1>, <arg2>	Якщо рядки *arg1* та *arg2* не збігаються (аргументи повинні бути взяті у кутові дужки)
IF1 (немає виразу)	Якщо виконується перший прохід транслятора
IF2 (немає виразу)	Якщо виконується другий прохід транслятора

2.5.7 Використання макробібліотек

Найбільшої ефективності і зручності у роботі з макросами можна досягти лише при використанні макробібліотек.

За допомогою будь-якого текстового редактора можна створити свою особисту бібліотеку, куди можуть бути розміщені макровизначення.

Якщо файл макробібліотеки має ім'я MACRO.LIB, то для її підключення на початку програми слід використати таку директиву:

```
INCLUDE  MACRO.LIB
```

Слід зауважити, що при підключенні макробібліотеки асемблер переписує в програму всі наявні в ній макровизначення, у тому числі і ті, до яких може і не бути звернень у даній програмі. Тому доцільніше мати декілька невеликих макробібліотек і підключати їх при необхідності.

Наприклад, якщо в наведеному вище прикладі вилучити макроси KARETKA, OUT_STR та OUT_NUMB і записати їх в окремий файл з іменем "MyLib.lib", то в основній програмі замість текстів коду цих макросів слід написати лише одну директиву:

```
INCLUDE MyLib.lib.
```

2.5.8 Директиви управління лістингом програми

Існує ряд директив, за допомогою яких можна керувати кількістю інформації, яка видається у лістинг програми, а отже, отримати або короткі, або більш детальні лістинги програми.

За допомогою директиви .LALL можна отримати найповнішій лістинг, включаючи макророзширення з коментарями.

Макровизначення може містити декілька коментарів (нагадаємо, що ознакою коментаря є наявність символу ";" на початку рядка), причому деякі з них можуть виводитися у лістингу, а інші – не будуть виводитись. Для того, щоб не виводити коментарі в лістингу, необхідно перед таким коментарем записати символи ";;".

За замовчуванням в асемблері діє директива .XALL, яка виводить у лістинг тільки ті рядки макровизначення, які генерують об'єктний код.

За допомогою директиви .SALL можна виключити з лістинга всі макророзширення, тобто отримати найкоротший лістинг.

Порядок виконання роботи

- вивчити теоретичну частину даної лабораторної роботи;
- для заданого прикладу скласти макровизначення;
- виконати програму з макровизначенням у тексті програми;
- створити власну макробібліотеку і виконати програму з підключенням цієї макробібліотеки;
- отримати лістинг програми і оформити звіт з лабораторної роботи.

ЛАБОРАТОРНА РОБОТА № 6

Транслятор з мови Асемблера

Тема: Вивчення роботи транслятора з мови Асемблера. Структура лістинга Асемблера. Об'єктні бібліотеки.

Теоретичні відомості

2.6.1 Призначення та основні функції транслятора у мові Асемблер

Транслятором з мови Асемблер, або, коротше кажучи, асемблером, називається системна програма, яка переводить текст програми з мови Асемблера на машинну мову.

Для зручності розрізнення однойменних назв будемо назву транслятора писати з маленької літери, а назву мови – з великої літери.

У загальному випадку програма мовою Асемблер може містити команди мови Асемблер, макрокоманди і директиви мови Асемблер (псевдокоманди). Тому Асемблер мікропроцесорів Intel 80I86 є по суті макромовою, і процес трансляції складається з двох стадій.

На першій стадії всі макрокоманди заміняються командами Асемблера. На другій стадії команди мови Асемблер переводяться в машинні коди, тобто здійснюється власне асемблювання.

На другій стадії в процесі трансляції вхідної програми асемблер виконує два перегляди (надалі проходи) вхідного тексту. Однією з основних причин цього є посилання вперед. Це здійснюється у тому випадку, коли в деякій команді є перехід на мітку, значення (величина зміщення) якої ще не визначено асемблером.

Під час першого проходу асемблер переглядає всю вхідну програму і будує таблицю символічних імен, які використовуються в програмі, тобто таблицю змінних, констант та міток програми і їх відносних адрес (зміщень).

Під час другого проходу, маючи в наявності всю необхідну інформацію, генерується об'єктна програма.

2.6.2 Види трансляторів з мови Асемблера

В наш час існує декілька різних трансляторів з мови Асемблера. Найбільш розповсюдженими з них є такі асемблери:

- Макроасемблер MASM (розробник – фірма Microsoft);
- Турбо-Асемблер TASM (розробник – фірма Borland International);
- асемблер OPTASM (розробник – фірма SLR System);
- асемблер 386/ASM (розробник – фірма PHAR Zab Software).

Найбільш повний діапазон можливостей мають асемблери MASM та TASM. Якщо в процесі роботи з прикладними програмами будуть використовуватись інші системні програми фірми Borland, тоді перевагу у цьому випадку слід надати TASM. Відмінною особливістю асемблера Tasm є те, що від генерує налагоджувальну інформацію для програми-налагоджувальника TURBO-DEBUGGER.

В подальшому розглядатиметься функціонування асемблера TASM.

2.6.3 Запуск на виконання транслятора TASM

Перед зверненням до транслятора необхідно створити файл з розширенням ".ASM", в якому буде знаходитись код вхідної програми мовою Асемблер.

В результаті трансляції одного вхідного файла може бути створено до трьох вихідних файлів:

- об'єктний файл, який буде містити об'єктний модуль (тип файла - ".OBJ");

- файл лістингу (тип файла - ".LST");
- файл перехресних посилань, що містить таблицю перехресних посилань (тип файла - ".CRF").

У деяких випадках таблиця перехресних посилань може знаходитись безпосередньо у файлі лістингу.

Отримання необхідних вихідних файлів визначається способом запуску транслятора на виконання. Тому розглянемо формат команди запуску TASM на виконання:

TASM [options] source[,object][,listing][,xref],

де options - опції, що визначають додаткові можливості трансляції,
source - ім'я вхідного файла,
object - ім'я об'єктного файла,
listing – ім'я файла лістингу,
xref – ім'я файла перехресних посилань.

Нехай, наприклад, файл "PROG.ASM" містить код вхідної програми мовою Асемблер. Тоді найпростіший варіант виклику асемблера буде такий:

TASM PROG

В результаті трансляції буде сформовано тільки один файл - PROG.OBJ.

Якщо ж необхідно отримати також і лістинг програми, то в командному рядку слід набрати команду:

TASM PROG, PROG, PROG

або

TASM PROG , ,

Для отримання всіх трьох вихідних файлів слід використати команду:

TASM PROG, PROG, PROG, PROG

або

TASM PROG , , ,

В результаті будуть створені такі файли: PROG.OBJ, PROG.LST, PROG.XRF. Імена цим файлам можна задати довільно, але початковий файл повинен обов'язково мати ім'я PROG.

Як вже відмічалось у першій частині даного посібника формуванням вказаних вихідних файлів можна керувати також за допомогою опцій команди TASM.

Найчастіше використовуються такі опції транслятора:
TASM /L - отримання об'єктного модуля і лістинга;

TASM /L/C - те саме, але лістинг містить також таблицю перехресних посилань;

TASM /L/N - виключення з лістингу таблиці символічних імен;

TASM /ZI - отримання об'єктного модуля разом з необхідною довідковою інформацією для налагоджувач TURBO DEBUGGER.

Повний список опцій транслятора можна отримати, якщо у командному рядку набрати тільки команду TASM (без параметрів). Тоді на екран буде виведено довідкову інформацію про опції транслятора

Всі наведені вище команди є правильними у тому випадку, коли всі програмні файли знаходяться у поточному каталозі. У випадку, коли деякі файли знаходяться у інших каталогах, необхідно вказувати повний шлях до них.

2.6.4 Структура лістинга програми

Після асемблювання лістинг програми містить таку інформацію:

- вхідну та об'єктну програми;
- таблицю символічних імен;
- таблицю перехресних посилань (вона може бути і відсутньою);
- повідомлення про помилки.

Мова Асемблер має у своєму арсеналі ряд директив, які дозволяють керувати процесом асемблювання і формування лістингу. Особливість директив у тому, що вони діють лише в процесі асемблювання і не генерують машинних кодів.

Директива LIST передбачає видачу повного лістингу (дії за замовчуванням). Директива XLIST забороняє видачу вхідної та об'єктної програм у лістингу.

Директива NAME у форматі

NAME ім'я_модуля

присвоює "внутрішнє" ім'я об'єктному модулю.

Для того, щоб зверху на кожній сторінці лістингу виводився заголовок (титул) програми, використовується директива TITLE у такому форматі:

TITLE текст.

Перші 6 знаків тексту використовуються як ім'я модуля, якщо немає директиви NAME. В разі відсутності директив NAME та TITLE ім'я вхідного файла вважається іменем об'єктного модуля.

Директиви CREF та XREF означають відповідно видачу і заборону таблиці перехресних посилань.

Для встановлення ширини та довжини сторінки використовується директива PAGE у такому форматі:

PAGE операнд_1, операнд_2,

де операнд_1 – кількість рядків на сторінку (число від 10 до 255), операнд_2 – кількість символів у рідку (число від 60 до 132). За замовчуванням в асемблері встановлено PAGE 66, 80.

2.6.5 Структура об'єктної програми

Основну частину лістинга займають вхідна програма (праворуч) і об'єктна програма (ліворуч).

У крайній лівій колонці вказані номери рядків програми.

У наступній колонці міститься значення лічильника адреси в шістнадцятковому форматі для кожного рядка програми. Кожний сегмент програми (стека, даних, коду) починається з відносної адреси 0000. В сегменті даних значення лічильника команд збільшується на кількість числа байтів у відповідності з типом даних (DB – 1, DW – 2, DD – 4). В сегменті коду значення лічильника команд збільшується на число байтів у відповідності з форматом команди. Директиви у програмі не змінюють значення лічильника команд.

Розглянемо більш детально процес трансляції команд Асемблера в машинний код.

Мова Асемблер є по суті майже машинною мовою, оскільки кожна її команда відповідає одній машинній команді, але записаній у символічній формі (у мнемокоді). Тому процес трансляції зводиться до заміни мнемокоду, зручного для людини, на машинний код, зручний для апаратної реалізації.

Для отримання машинного коду будь-якої команди необхідно знати її формат та режим адресації мікропроцесора.

Наприклад, команда

MOV *reg1, reg2*

використовує *регістрову адресацію* для обох операндів, тому її формат такий, як показано на рис.1.

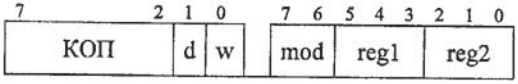


Рисунок 1 – Формат команди для режиму регістрової адресації

Перший байт команди містить код операції (КОП) і два однобітових поля – направлення передачі *d* (якщо *d*=1,то в напрямі мікропроцесора, а якщо *d*=0, то, навпаки, з мікропроцесора) та формат даних *w* (якщо *w*=1, то команда оперує словом, а якщо *w*=0, то байтом).

Другий байт команди (постбайт) вказує на режим адресації *mod* і коди регістрів *reg1* та *reg2*. Для регістрової адресації поле *mod* дорівнює 11. Спосіб кодування регістрів мікропроцесора наведений в табл.2.

Таблиця 2 – Адресація регістрів

Код (поле reg)	8-бітові регістри	16-бітові регістри	32-бітові регістри
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Таким чином, команді Асемблера

MOV BX, SI

відповідає такий машинний код:

10001011 11011110

Розглянемо більш загальні випадки, коли один з операндів знаходиться у комірці пам'яті. Таким випадкам відповідають наступні режими адресації, тобто, методи визначення ефективної (виконавчої) адреси в команді: пряма, індексна, базово-індексна, опосередкована регістрова.

Найбільш загальний формат двоопрандної команди для вказаних режимів адресації наведено на рис.2.

Другий байт команди складається з трьох полів:

mod – режим адресації;

reg – регістр;

r/m – регістр/пам'ять.

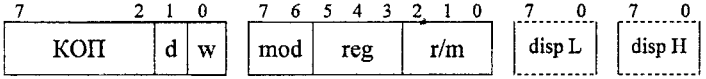


Рисунок 2 – Загальний формат двоопрандної команди

Якщо *mod* = 11, це означає, що операнд знаходиться в регістрі, в решті випадків операнд знаходиться в пам'яті, і тоді поле mod визначає, як використовується (необов'язкове) зміщення *disp*, яке знаходиться в третьому та четвертому байтах команди, а саме:

mod=
- 00, *disp*=0 – зміщення відсутнє,
- 01, *disp=disp L* – команда містить 8-бітне зміщення, яке розширюється зі знаком до 16 бітів
- 10, *disp=disp H, disp L* – команді містить два байти зміщення

В тих самих випадках (коли mod ≠ 11) поле *r/m* визначає, яким чином формується ефективна адреса операнда. Кодування поля *r/m* наведено в таблиці 3.

Таблиця 3 – Формування ефективної адреси пам'яті

Поле *r/m*	Ефективна адреса
000	EA = (BX) + (SI) + disp
001	EA = (BX) + (DI) + disp
010	EA = (BP) + (SI) + disp
011	EA = (BP) + (DI) + disp
100	EA = (SI) + disp
101	EA = (DI) + disp
110	EA = (BP) + disp
111	EA = (BX) + disp

Наведені правила формування ефективної адреси мають один виняток: якщо *mod* = 00 та *r/m* = 110, то EA = *disp H, disp L*. Тут в команді міститься абсолютна адреса пам'яті.

Як приклад розглянемо асемблювання команди додавання до регістру BX елемента масива MAS:

ADD BX, MAS[SI]

В цій команді використовується індексна адресація, при якій ефективна адреса операнда обчислюється як сума зміщення, що знаходиться в команді, так і вмісту регістрів SI і DI.

При асемблюванні величина зміщення *disp* визначається транслятором із сегменту даних, де описується масив MAS. Нехай, наприклад, елементи масива MAS задані у форматі слова і починаються з шостого байта від початку сегмента даних. Код операції даної команди додавання становить 000 000.

Таким чином, отримуємо такий машинний код команди, яку ми асемблюємо:

0000 0011 1001 1100 0000 0110 0000 0000

або в шістнадцятковому форматі:

03 9C 06 00.

Існує ряд особливостей асемблювання команд з безпосередньою адресацією. Формат двоопераднної команди з безпосередньою адресацією наведено на рис.3.

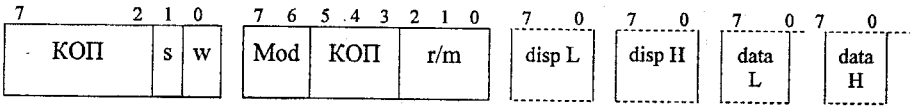


Рисунок 3 – Формат двоопераднної команди з безпосередньою адресацією

В таких командах другий операнд знаходиться в самій команді і його адресувати немає потреби, тому поле *reg* другого байта команди використовується як розширення коду операції. Крім того, тут не потрібний біт *d*, оскільки результат можна розмістити тільки на місце першого операнда. Але і в цьому форматі необхідно визначити розмір (тип) безпосереднього операнда. Для цієї мети призначені біти *s* та *w*.

sw= { x0, один байт даних *data L*;
01, два байти (слово) даних *data H, data L*;
10, один байт даних, який розширюється зі знаком до 16 бітів

Поля *mod* та *r/m* інтерпретуються так само, як і в попередньому форматі.

В Асемблері є також велика кількість інших форматів команд, інформацію про які можна знайти в спеціальній літературі.

2.6.6 Приклад ручного асемблювання програми

Розглянемо для прикладу просту програму мовою Асемблер, яка призначення для знаходження найбільшого елемента масива.

```
NAME    PROG
STACKSG SEGMENT STACK
    DB  128 DUP(?) ; резервування пам'яті для стеку
STACKSG ENDS
DATE    SEGMENT
    A   DB  1,2,33,4,2,0,5,6,7,8,5,-3,9,127,11,12,2,2
        DB  9,127,11,12,2,2,13,14,15,16,-10,1
    N   DB  4
    M   DB  6
    MAX DB  0
    BUF DB  32 DUP(0)
    mess db "REZULT: $"
DATE    ENDS
CODE_SG SEGMENT
    ASSUME  CS:CODE_SG, DS:DATE_SG, SS:STACKSG
START   PROC    FAR
        PUSH    DS
        SUB     AX, AX
        PUSH    AX
        MOV     AX, DATE_SG
        MOV     DS, AX
        CALL    MAIN
        CALL    VYVOD
```

```
        RET
START   ENDP
MAIN    PROC
        MOV     AL, N
        CBW
        MUL     M
        MOV     CX, AX      ; загальна кількість елементів у масиві
        SUB     AX, AX
        MOV     SI, 0
        MOV     AL, A[SI]   ; завантажуємо перший елемент масива
    M1:
        INC     SI
        CMP     A[SI], AL
        JGE     M2          ; вибір більшого елемента
        MOV     AL, A[SI]
    M2:
        LOOP    M1
        CBW
        RET
MAIN    ENDP

VYVOD PROC
    ; тіло процедури для виведення на екран десяткового числа
VYVOD   ENDP
CODE_SG ENDS
    END START
```

Побудуємо машинний код для фрагмента програми, яка власне здійснює обчислення найбільшого елементу, тобто процедуру MAIN. Результати нашої роботи для зручності занесемо у табл. 4. Для того, щоб мати адреси даних, що використовуються, наведемо і результати I проходу для сегмента даних.

2.6.7 Таблиця символічних імен

За об'єктною та вхідною програмами розташована таблиця символічних імен.

Перша частина таблиці містить довідкову інформацію про ім'я файла, куди розміщено вхідну програму, дату та час трансляції і т.д.

У другій частині таблиці символічних імен зібрані дані про використані в програмі символічні імена, для змінних вказується їх тип (BYTE, WORD, DWORD), зміщення та ім'я сегмента. Якщо в програмі

використані зовнішні змінні та глобальні імена, тоді всі вони перераховуються у таблиці символічних імен.

Таблиця 4 – Побудова машинного коду фрагменту програми

Команди програми	I прохід	II прохід				Шістнадцятковий код команди
... DATE SEGMENT A DB 1,2,33,4,2,0 DB 5,6,7,8,5,-3 DB 9,7,11,1,2,2 DB 4,5,6,10,1,0 N DB 4 M DB 6 ...	... 7 0000 8 0000 9 0006 10 000C 11 0012 12 0018 13 0019 ...					
MOV AL, N	39 0010	10100000	00000000	00011000		A0 00 18
CBW	40 0013	10011000				98
MUL M	41 0014	11110110	00100110	00000000	00011001	F6 26 00 19
MOV CX, AX	42 0018	10001011	11001000			8B C8
SUB AX, AX	43 001A	00101011	11000000			2B C0
MOV SI, 0	44 001C	10111110	00000000	00000000		BE 00 00
MOV AL, A[SI]	45 001F	10001010	10000100	00000000	00000000	8A 84 00 00
M1:	46 0023					
INC SI	47 0023	01000110				46
CMP A[SI], AL	48 0024	00111000	10000100	00000000	00000000	38 84 00 00
JGE M2	49 0028	01111101	00000100			7D 04
MOV AL, A[SI]	50 002A	10001010	10000100	00000000	00000000	8A 84 00 00
M2:	51 002E					
LOOP M1	52 002E	11100010	11110011			E2 F3
CBW	53 0030	10011000				98
RET ...	54 0031 ...	11000011				C3

Для всіх сегментів і груп, що використовуються в програмі, наводяться їх імена, розмір в байтах, типи об'єднання та вирівнювання.

Нижче показано таблицю символічних імен для програми, яку ми навели вище.

Symbol Table

Symbol Name	Type	Value
??DATE	Text	"16/02/04"

```
??FILENAME              Text "lab4  "
??TIME                  Text "08:36:34"
??VERSION               Number 040A
@CPU                    Text 0101H
@CURSEG                 Text CODE_SG
@FILENAME               Text LAB4
@WORDSIZE               Text 2
A                   Byte DATE:0000
ADDRMES                 Dword     DATE_SG:0045
BUF                 Byte DATE:001B
M                   Byte DATE:0019
M1                  Near CODE:0023
M2                  Near CODE:002E
MAIN                    Near CODE_SG:0010
MAX                 Byte DATE:001A
MESS                    Byte DATE:003B
MET1                    Near CODE_SG:0049
MET2                    Near CODE_SG:0055
N                   Byte DATE_SG:0018
START                   Far  CODE_SG:0000
VYVOD                   Near CODE_SG:0032

Groups & Segments       Bit Size Align  Combine Class

CODE_SG                 16  0063 Para     none
DATE_SG                 16  0049 Para     none
STACKSG                 16  0080 Para     Stack
```

2.6.8 Таблиці перехресних посилань

Перехресні посилання використовуються для допомоги при відлагоджуванні програми.

У таблиці перехресних посилань для кожного символічного імені вказуються номери рядків програми, в яких вони з'являються. Якщо за номером рядка знаходиться символ "#", це означає, що ім'я було визначено саме в цьому рядку; в противному разі у цьому рядку міститься тільки посилання на вказане ім'я. Така інформація дозволяє легко визначити всі команди, що використовують дане ім'я.

Якщо транслятор формує окремий файл перехресних посилань типу .CRF, то для використання цього файла, необхідно перетворити його в текстовий (формат ASCII). Для цього необхідно запустити програму CREF або TCREF. В результаті буде отримано необхідний вихідний файл типу .REF.

2.6.9 Об'єктні бібліотеки

При налагоджуванні багатомодульних програм є сенс об'єднувати окремі об'єктні модулі в одну об'єктну бібліотеку. Формат команди запуску об'єктної бібліотеки TLIB на виконання має вигляд:

TLIB libname [/c] [/e] commands, listfile

де libname – ім'я бібліотеки;

commands – операції з бібліотекою, наприклад:

+ означає додавання модуля в бібліотеку;
- означає вилучення модуля з бібліотеки;
* означає добування модуля з бібліотеки без його вилучення;

/c – режим роботи з урахуванням регістра, тобто режим, при якому розрізняються великі і малі літери;

/e – створення розширеного словника.

Наприклад, для створення об'єктної бібліотеки MYLIB.LIB і поміщення туди об'єктних модулів PROG1.OBJ та PROG2.OBJ необхідно виконати команду:

TLIB MYLIB + PROG1, PROG2

Порядок виконання роботи

- вивчити теоретичну частину лабораторної роботи;
- для заданого прикладу скласти текст програми мовою Асемблер і виконати трансляцію з отриманням лістинга програми;
- вивчити різні варіанти отримання лістинга програми;
- отримати окремий файл перехресних посилань в коді ASCII.
- виконати вручну асемблювання вхідної програми для першого і другого проходів транслятора;
- оформити звіт по лабораторній роботі.

ЛАБОРАТОРНА РОБОТА № 7

Компонувачі

Тема: Вивчення роботи компонувача (редактора зв'язків). Структура карти завантаження. Розбиття програми на Асемблері на зовнішні процедури (модулі).

Теоретичні відомості

2.7.1 Призначення та основні функції компонувача (редактора зв'язків)

Об'єктна програма, яку ми отримуємо після трансляції, не може бути відразу виконана.

По-перше, всі адреси в об'єктній програмі є відносними, оскільки в кожному сегменті вони починаються з нульової адреси. У випадку, коли вхідна програма складається з окремих модулів, кожний з яких транслювався окремо, їх об'єктні модулі необхідно об'єднати в один, виконавши при цьому настроювання всіх адрес.

Таким чином, необхідна спеціальна системна програма, що називається компонувачом, або редактором зв'язків, для виконання таких функцій:

- розподілення пам'яті для всіх програмних модулів (функція розподілення);
- зв'язування в одне ціле програмних модулів шляхом визначення значень зовнішніх символічних імен між ними (функція зв'язування);
- настроювання в програмі всіх величин, що залежать від конкретних фізичних адрес (функція переміщення).

Результатом роботи компонувача є завантажувальний модуль.

Для того, щоб виконати завантажувальний модуль, його необхідно розмістити в потрібну область оперативної пам'яті, змінивши попередньо всі відносні адреси на абсолютні, тобто фізичні. Цю останню задачу виконує системна програма завантажувач.

Загальна схема виконання найпростішої одномодульної прикладної програми в комп'ютері наведена на рис.4.

Головною властивістю компонувача є можливість з його допомогою реалізувати принцип модульного програмування. Розробка складних прикладних програм як набору вхідних модулів має певні переваги, а саме:

- значно спрощується процес налагоджування програм;
- є можливість використовувати в програмі модулі, які налагоджені раніше;
- окремі модулі можна писати на різних мовах програмування.

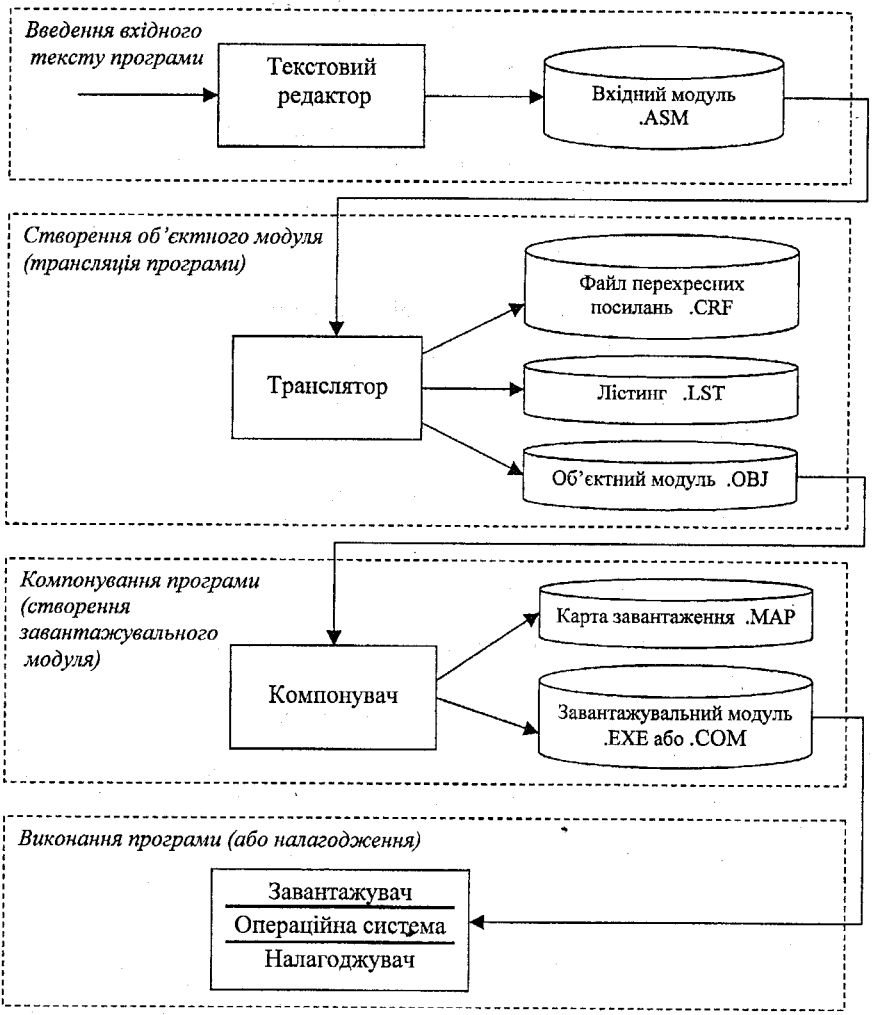


Рисунок 4 – Загальна схема виконання найпростішої одномодульної програми

2.7.2 Типи компонувачів

Існує декілька типів компонувачів. Найбільше розповсюджені з них такі:

- компонувач LINK (розробник – фірма Microsoft);
- компонувач MSLINK (розробник – фірма Microsoft);
- компонувач TLINK (розробник – фірма Borland International).

Компонувач TLINK серед наведених є найшвидшим, хоча і має ряд обмежень:

- не підтримуються оверлейні структури;
- сегменти з одним і тим самим іменем і класом повинні бути всі здатні до об'єднання, або всі не здатні;
- не можна зв'язати разом об'єктні коди, відтрансльовані компіляторами Microsoft C та Microsoft Fortran, оскільки вони мають різні формати об'єктних модулів.

Надалі розглядатимемо функціонування компонувачів TLINK та LINK, які за виконуваними функціями близькі один до одного.

2.7.3 Запуск компонувача на виконання

Вхідною інформацією для компонувача є один або декілька об'єктних модулів, що містяться у файлах типу ".OBJ". Об'єктні модулі можуть також знаходитись в об'єктних бібліотеках.

В результаті компонування може бути отримано два файли:

- файл, що містить завантажувальний модуль (типу ".EXE" або ".COM");
- файл, що містить карту завантаження (тип файла ".MAP").

Кількість вхідних файлів та їх структура визначаються способом запуску компонувача на виконання.

Формат команди запуску компонувача TLINK на виконання має такий вигляд:

TLINK [options] objfiles, mapfiles, libfiles

де options – опції, що визначають додаткові можливості компонувача;
objfiles – ім'я об'єктного файла;
mapfiles – ім'я файла, що містить карту завантаження;
libfiles – ім'я об'єктної бібліотеки.

Нехай, наприклад, об'єктний модуль знаходиться у файлі "PROG.OBJ". Тоді для отримання завантажувального модуля і карти завантаження необхідно виконати команду

TLINK PROG.

В результаті компонування будуть сформовані файли "PROG.EXE" та "PROG.MAP".

При об'єднані декількох об'єктних модулів, наприклад, "PROG1.OBJ" та "PROG2.OBJ", команда запуску TLINK може бути такою:

TLINK PROG1+PROG2, PR, PR

В результаті компонування будуть створені єдиний завантажувальний модуль у файлі "PR.EXE" і об'єднана карта завантаження у файлі "PR.MAP".

Якщо деякі об'єктні модулі знаходяться в об'єктній бібліотеці "MYLIB.OBJ", то команда запуску TLINK буде такою:

TLINK PROG1, PR, PR, MYLIB

Якщо в командному рядку набрати тільки TLINK, тоді можна отримати довідкову інформацію про опції компонувача. Опції, що використовуються найчастіше, такі:

TLINK /m PROC – видача карти завантаження з таблицею зовнішніх імен;

TLINK /s PROC – видача розширеної карти завантаження (по сегментах);

TLINK /x PROC – заборона на видачу карти завантаження;

TLINK /t PROC – створення завантажувального модуля у файлі типу ".COM";

TLINK /v PROC – отримання додаткової довідкової інформації для налагоджувача TURBO DEBUGGER.

2.7.4 Карта завантаження

Карта завантаження (карта зв'язків), що знаходиться у файлі ".MAP", містить результат роботи компонувача, тобто являє собою по суті лістинг компонувача.

У цій карті для кожного сегмента вказано його довжину в байтах, а також початкову та кінцеву адреси у тому вигляді, в якому вони будуть завантажені в оперативну пам'ять. Всі ці адреси є 20-бітовими і розпочинаються від нульової комірки.

Приклад найпростішої карти завантаження:

START	STOP	LENGTH	NAME
00000H	0007FH	00080H	SSTACK
00080H	00085H	00006H	DATA
00090H	000B8H	00029H	CODE

Оскільки завантаження програми в оперативну пам'ять здійснюється операційною системою, то завантажувач змінить значення цих адрес. Але ж відносно один одного їх значення залишаться такими самими.

Зазвичай сегменти розташовуються на границях параграфів для того, щоб адреси зміщення зберігали вірні значення. При цьому між окремими сегментами можуть знаходитись невикористані байти (до 15 байтів), як це видно з попереднього прикладу. Але в Асемблері передбачено засоби, що забезпечують більш щільне розміщення сегментів. За допомогою директиви SEGMENT можна задавати тип вирівнювання на границях слів або байтів.

Наприкінці карти завантаження вказується вхідна точка виконуваного файла. Ця адреса, як і інші, розраховується по відношенню до

початку завантажувального модуля і настроюється в пам'яті завантажувача. Існує декілька способів вказування стартової адреси для завантажувального модуля типу ".EXE".

При одному із способів програма виконується, починаючи з першого байта. При цьому необхідно слідкувати за тим, щоб в першому байті першого сегмента виконуваного файла містилася саме та команда, з якої і треба починати виконання.

Переважає спосіб задання вхідної точки в директиві END головного модуля програми:

END START,

де START – мітка першої виконуваної команди у програмі.

У випадку, коли в програмі задано більше, ніж одна точка входу, компонувач вибирає ту, яка вказана останньою.

Все це справедливо для завантажувального модуля типу ".EXE". Модулі типу ".COM" завжди починають виконуватись зі зміщення 100H сегмента коду.

2.7.5 Директиви зв'язку модулів та сегментів

При модульному програмуванні виникають специфічні проблеми об'єднання окремо відтрансльованих модулів у єдиний завантажувальний модуль.

Головна проблема полягає у тому, що з даного модуля можливі звернення (посилання) до символічних імен (в першу чергу до змінних і міток) в інших модулях, а ці інші модулі можуть звертатись до таких самих імен даного модуля. Саме за допомогою таких модульних звернень окремі модулі об'єднуються в єдину програму.

Припустимо, що програма складається з двох модулів – MOD1 та MOD2. Нехай у модулі MOD1 визначена змінна VAR1, а в модулі MOD2 – змінна VAR2, причому обидві змінні використовуються в обох модулях.

Для того, щоб використовувати вказані змінні в обох модулях, їх імена повинні бути вказані в директивах EXTRN та PUBLIC.

В директиві EXTRN перераховуються всі імена, до яких є звернення в даному модулі, але які визначені в інших модулях. Такі імена є зовнішніми по відношенню до даного модуля. Дана директива має такий формат:

EXTRN ім'я: тип [, ім'я: тип, . . .].

Іменем в директиві EXTRN може бути ім'я змінної, мітки або константи, визначної директивою EQU. Типом може бути BYTE, WORD або DWORD для змінної, NEAR або FAR для мітки, ABS для константи, причому ABS означає абсолютне число, а не змінну і не мітку.

В директиві PUBLIC перераховуються всі імена, які визначені в даному модулі і є доступними для звернення з інших модулів. Такі імена є

глобальними по відношенню до даного модуля. Дана директива має такий формат:

PUBLIC ім'я [, ім'я, . . .]

Іменами тут можуть бути такі самі імена, як і в директиві EXTRN.

На перший погляд здається зручним перераховувати в директиві PUBLIC імена всіх змінних, що використовуються у модулі. Але ж це означало би, що в будь-якому модулі, які повинні бути зв'язані між собою, всі імена мають бути унікальними. Тому доцільно в директиві PUBLIC перераховувати лише ті імена, які необхідні для зв'язування модулів, тоді решта імен в різних модулях можуть бути однаковими.

Таким чином, для нашого прикладу змінні VAR1 і VAR2 мають бути описані так, як це показано на рис. 5.

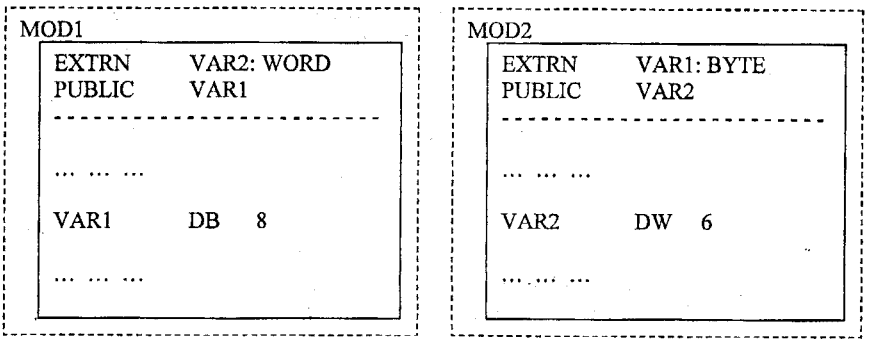


Рисунок 5 – Приклад взаємозв'язків між зовнішніми модулями

Припустимо, що модулі MOD1 і MOD2 містять сегменти стеку, даних та коду. Тоді при об'єднанні цих модулів можна отримати шість сегментів. Оскільки сегменти коду і даних в обох модулях адресуються за допомогою одних і тих самих сегментних регістрів (CS, DS), то при кожній передачі управління між модулями необхідно змінювати вміст сегментних регістрів. Крім того, об'єднаній програмі потрібен один стек, а отже, одна з областей стеку виявляється непотрібною, а розмір іншої повинен відповідати вимогам до стеку обох модулів.

Якщо об'єднаний розмір сегментів коду менший за 64К, то сегменти кодів обох модулів можна об'єднати і немає необхідності витрачати час на модифікацію регістра CS. Аналогічним чином можна розв'язати проблеми при об'єднанні сегментів даних і сегментів стека модулів.

Варіанти об'єднання логічних сегментів задаються в директиві SEGMENT, повний формат якої має вигляд:

ім'я SEGMENT [тип вирівнювання] [тип об'єднання] ['ім'я класу']

В полі "тип вирівнювання" вказуються атрибути, що визначають границю, на якій повинен бути розміщений логічний сегмент:

PARA – на границі параграфа;

WORD – на границі слова;

BYTE – на границі байта;

PAGE – на границі сторінки.

В полі "тип об'єднання" вказуються атрибути, що визначають спосіб об'єднання даного логічного сегмента з іншим логічним сегментом, що має таке саме ім'я: PUBLIC, STACK, COMMON, MEMORY, AT.

Коли логічному сегменту призначається атрибут "ім'я класу", то компонувач збирає разом всі області з однаковими іменами класів.

2.7.6 Приклад розбиття програми на зовнішні процедури

Нижче наведено код програми на Асемблері для обчислення суми елементів вектора.

```
NAME    PROG
SSTACK  SEGMENT
        DB      128 DUP (?)
SSTACK  ENDS
DATA    SEGMENT
MAS     DW      1,3,1,2,1
N       DW      5
Y       DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA, SS:SSTACK
START   PROC    FAR
        PUSH
        SUB     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        CALL    MAIN
        CALL    OUTPUT
        RET
START   ENDP
MAIN    PROC
        SUB     AX, AX
        MOV     CX, 5
        LEA     SI, MAS
  MET:  ADD     AX, [SI]
```

```
            ADD     SI, 2
            LOOP    MET
            MOV     Y,AX
            RET
MAIN        ENDP
OUTPUT      PROC
            MOV     AX, Y
            ADD     AL, 48
            MOV     DL, AL
            MOV     AH, 06h
            INT     21h
            RET
OUTPUT      ENDP
CODE        ENDS
            END     START
```

Розіб'ємо цю програму на три процедури: головну процедуру START, процедуру MAIN - для визначення суми, процедуру OUTPUT - для виведення результату обчислень. В процедурі START будуть визначатися всі змінні і здійснюватись виклики решти процедур. Процедура MAIN отримує від процедури START початкові дані для обчислень і повертає їй результат обчислень. Процедура OUTPUT отримує від процедури START значення результату для виведення його на екран дисплея.

З урахуванням вказаних вище умов головна процедура буде мати такий вигляд:

```
            NAME    PROG1
            EXTRN   MAIN:NEAR, OUTPUT:NEAR
            PUBLIC  MAS, N, Y
SSTACK      SEGMENT
            DB      128 DUP (?).
SSTACK      ENDS
DATA        SEGMENT
    MAS         DW      1,3,1,2,1
    N           DW      5
    Y           DW      ?
DATA        ENDS
CODE        SEGMENT
            ASSUME  CS:CODE, DS:DATA, SS:SSTACK
START       PROC    FAR
            PUSH
            SUB     AX, AX
            PUSH    AX
```

```
        MOV     AX, DATA
        MOV     DS, AX
        CALL    MAIN
        CALL    OUTPUT
        RET
START   ENDP
CODE    ENDS
        END     START
```

Зовнішня процедура MAIN матиме вигляд:

```
        NAME    PROG2
        EXTRN   MAS:WORD, N:WORD, Y:WORD
        PUBLIC  MAIN

SSTACK  SEGMENT
        DB      128 DUP (?)
SSTACK  ENDS

CODE    SEGMENT PUBLIC
        ASSUME  CS:CODE, SS:SSTACK
MAIN    PROC    NEAR
        SUB     AX, AX
        MOV     CX, 5
        LEA     SI, MAS
  MET:  ADD     AX, [SI]
        ADD     SI, 2
        LOOP    MET
        MOV     Y,AX
        RET
MAIN    ENDP
CODE    ENDS
        END
```

Зовнішня процедура OUTPUT буде такою:

```
        NAME    PROG3
        EXTRN   Y:WORD
        PUBLIC  OUTPUT

SSTACK  SEGMENT
        DB      128 DUP (?)
SSTACK  ENDS

CODE    SEGMENT PUBLIC
        ASSUME  CS:CODE, SS:SSTACK
OUTPUT  PROC
        MOV     AX, Y
        ADD     AL, 48
```

```
        MOV     DL, AL
        MOV     AH, 06h
        INT     21h
        RET
OUTPUT  ENDP
CODE    ENDS
        END
```

Порядок виконання роботи

1. Вивчити теоретичну частину лабораторної роботи.
2. Для заданого приклада розбити вхідну програму на зовнішні процедури і виконати трансляцію окремо для кожної процедури.
3. За допомогою компонувача TLINK створити єдиний завантажувальний модуль. Дослідити різні рівні роботи компонувача.
4. Сформувати розширену карту завантаження і здійснити її аналіз з урахуванням типів об'єднання і типів вирівнювання логічних сегментів.

ЛАБОРАТОРНА РОБОТА № 8

Математичний співпроцесор

Тема: Знайомство з архітектурою математичного співпроцесора. Вивчення форматів даних та системи команд для управління математичним співпроцесором. Програмування мовою Асемблер із використанням математичного співпроцесора.

Теоретичні відомості

2.8.1 Архітектура математичного співпроцесора

Математичний співпроцесор (МС) призначений для виконання операцій над числами з фіксованою та плаваючою комою. Завдяки математичному співпроцесору можна досягти високої точності і швидкодії при різноманітних математичних обчисленнях, при роботі з машинною графікою, при виконанні статистичних та інженерних розрахунків, при використанні засобів мультимедіа і т.д.

Математичний співпроцесор може працювати лише разом із головним процесором, оскільки в ньому відсутній механізм вибірки команд.

Перший математичний співпроцесор 8087 працював у перших моделях IBM PC. Згодом були розроблені математичні співпроцесори 80287 та 80387. У процесорах Intel 80486 та Pentium наявний вбудований

співпроцесор, що є одним з модулів головного процесора, але з програмної точки зору вони залишились повністю сумісними з 80387.

В загальному випадку МС x87 можна розглядати як архітектурне розширення центрального процесора, тобто до загальних регістрів процесора додаються вісім 80-розрядних арифметичних регістри стека і блок регістрів стану та управління.

Блок регістрів стану та управління складається з таких 16-розрядних регістрів:

- слово стану SWR (Status Word Register);
- слово управління CWR (Control Word Register);
- ознаки;
- покажчик команд;
- покажчик даних.

Група арифметичних регістрів частіше використовується в режимі стеку, тобто операнди зчитуються у порядку, оберненому до порядку запису. Але підтримуються всі форми команд, що виконують запис і зчитування для конкретних регістрів.

З регістрами стеку пов'язаний 3-бітовий покажчик стеку ST, що визначає регістр, який в даний момент є вершиною стеку (рис.10). Нумерація регістрів завжди починається від вершини стеку.

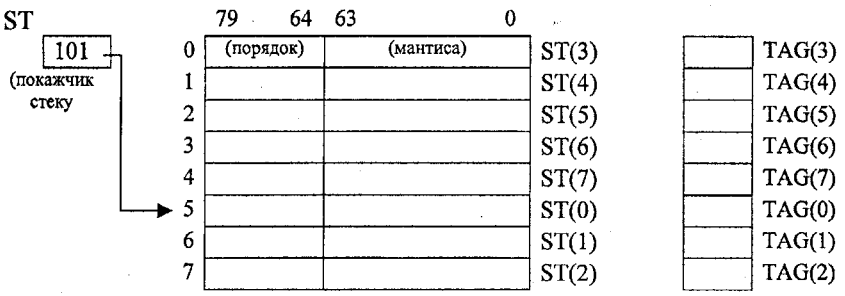


Рисунок 10 – Схема зв'язку покажчика стеку з регістрами співпроцесора

При завантаженні даних в арифметичний регістр вміст покажчика стеку попередньо зменшується на одиницю (ST ← ST-1) і вказує номер регістра, в який здійснюється завантаження. При добуванні даних з регістру вміст покажчика стеку автоматично збільшується на одиницю (ST ← ST+1).

З кожним з арифметичних регістрів пов'язаний регістр тегів, в якому є 2-бітове поле, значення якого залежить від вмісту регістра ST(i) (рис.11).

Математичний співпроцесор має у своєму складі:

- блок обробки мантиси;
- блок обробки порядку;

- ПЗП констант, який зберігає 7 відомих констант: +0.0; +1.0; π; $ln2$; $lg2$; log_210; log_2e.

Код поля TAG(i)	Вміст регістра ST(i)
00	Скінчене число, що не дорівнює 0
01	Нуль
10	NAN, $\pm\infty$
11	Регістр пустий (позначається e)

Рисунок 11 – Значення полів тегів

2.8.2 Формати даних співпроцесора 8087

Математичний співпроцесор X87 підтримує 7 форматів:
- 3 формати цілих чисел;
- 3 формати дійсних чисел з плаваючою комою;
- 1 формат цілих двійково-десяткових чисел.

Формати цілих чисел

На рис. 12 наведені формати цілих чисел математичного співпроцесора X87.

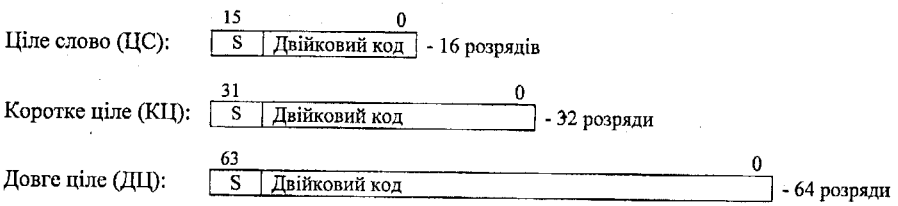


Рисунок 12 – Формати цілих чисел.

У всіх цих форматах від'ємні числа записуються у доповняльному коді (якщо s=0, число додатне, якщо s=1, - число від'ємне).

Формати дійсних чисел

Дійсні числа можуть бути представлені в одному з трьох форматів (рис.13).
Мантиса числа завжди записується в нормалізованому вигляді:

$$1, m_1m_2m_3...$$

Порядок дійсного числа завжди записується в зміщеному вигляді, тому реальний порядок дорівнює числу в полі порядку мінус значення зміщення. Це дозволяє спростити порівняння чисел.

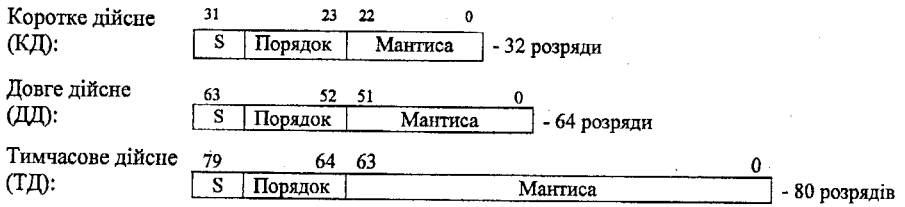


Рисунок 13 – Формати дійсних чисел

Для обчислення реальних значень коротких дійсних чисел використовують таку формулу:

$$(-1)^S \cdot (1, m_1m_2...m_{23}) \cdot 2^{E-127}.$$

Наприклад, нехай задано:

S	E	$m_1m_2m_3m_4m_5...m_{23}$
1	01111101	1 0 1 1 0 ... 0

Обчислюємо реальне значення числа:

$$(-1)^1 \cdot (1,1011)_2 \cdot 2^{125-127} = -(1,6775)_{10} \cdot 2^{-2} = -0,419375.$$

Число +1.0 представляється у вигляді:

S	E	$m_1m_2m_3m_4m_5...m_{23}$
0	01111111	0 0 0 0 0 ... 0

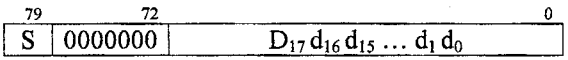
У форматах короткого та довгого дійсних чисел такі величини, як +∞, -∞ та NAN (Not a Number, тобто не числа) представляються таким чином (рис. 14).

Знак S	Порядок	Мантиса	Значення	
0	11 ... 11	00 ... 00	+∞	
1	11 ... 11	00 ... 00	-∞	
0	11 ... 11	11 ... 11	NAN	тобто, коли в полі мантиси будь-яке ненульове число
1	11 ... 11	11 ... 01	NAN	
...	...	...	NAN	
0	11 ... 11	00 ... 11	NAN	
1	11 ... 11	00 ... 01	NAN	

Рисунок 14 – Представлення величин +∞, -∞ та NAN

Формат десяткового числа

Двійково-десяткове число має такий формат:



де d – тетрада (4 двійкових розряди).

2.8.3 Система команд 8087

Повна система команд 8087 включає в себе 6 команд і побудована на основі команди ESC. Використовується 2 формати команди ESC.

а) Формат команди з постбайтом – застосовується у тих випадках, коли операнд-джерело знаходиться в пам'яті або в регістрі ST(i), або коли результат необхідно переслати в пам'ять або в регістр ST(i) (рис. 15).

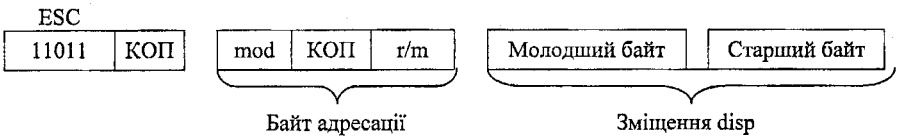


Рисунок 15 – Формат команди з постбайтом

Зміщення disp може дорівнювати:

1 байт (у цьому випадку mod = 01);

2 байти (у цьому випадку mod = 10);

бути відсутнім (тоді mod = 00 або mod = 11).

У випадку mod=11 операнд розміщується в одному з арифметичних регістрів. Виняток складають випадки, коли mod=11 і r/m=110, - тоді зміщення складає 2 байти і повністю визначає виконувану адресу.

Б) Формат команди без постбайта – застосовується у тих випадках, коли операнд розташований в регістрі ST(i) або в командах управління (рис. 16).

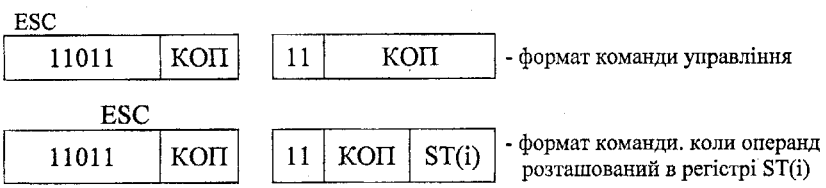


Рисунок 16 – Формат команди без постбайта

Команди керування

До команд керування відносяться 15 команд, які використовуються не для обчислень. а для виконання системних функцій. Більшість із цих команд має два варіанти (два мнемокоди). Другий мнемокод утворюється шляхом додавання до мнемокоду букви N, яка вказує, що при асемблюванні даного мнемокоду перед ним не треба розміщувати команду WAIT. Другий мнемокод використовується у програмах, які є критичними за часом виконання. До основних команд керування належать:

- команда ініціалізації співпроцесора – FINIT або FNINIT;
- команда очікування – FWAIT;
- команда збільшення покажчика стека – FINCSTR
  (якщо ST=7, то після виконання даної команди ST=0);
- команда зменшення покажчика стека – FDECSTR
  (якщо ST=0, то після виконання команди ST=7).

Команди передавання даних

Команди передавання даних включають 4 типи команд:

- команди завантаження;
- команди збереження (запам'ятовування);
- команда обміну;
- команди завантаження констант.

До основних команд завантаження належать:

- команда завантаження дійсного числа – FLD src або FLD ST(0),src (передача даних здійснюється від джерела src у вершину стека ST(0));
- команда завантаження цілого числа – FILD src (число завантажується у вершину стека);
- команда завантаження десяткового числа – FBLD src (число завантажується у вершину стека).

До основних команд збереження належать:

- команда збереження дійсного числа – FST dst або FST ST(i) (Число завантажується з вершини стеку в пам'ять за адресою dst або в інший регістр ST(i));
- команда збереження цілого числа – *FIST dst*;

• команда збереження дійсного числа з одночасним виштовхуванням зі стека – FSTP dst;
• команда збереження цілого числа з одночасним виштовхуванням зі стека – FISTP dst;
• команда збереження десяткового числа з одночасним виштовхуванням зі стека – FSTP dst.

До команди обміну належить:
• команда обміну – FXCH dst
(обмінюється вміст регістра-приймача dst з вершиною стека. За замовчуванням (команда FXCH) для обміну з вершиною стека ST(0) береться dst=ST(1).

До основних команд завантаження констант належать:
• команда завантаження числа +0.0 – FLDZ
(число завантажується у вершину стека);
• команда завантаження числа 1.0 – FLD1
(число завантажується у вершину стека);
• команда завантаження числа π FBLPI
(число завантажується у вершину стека);
• команда завантаження числа $\log_2 10$ – FLDL2T
(число завантажується у вершину стека).

Арифметичні команди

Арифметичні команди включають 6 типів команд:
- додавання;
- віднімання;
- множення;
- ділення;
- порівняння;
- аналізу.

До основних арифметичних команд належать:

• команда додавання дійсних чисел – FADD dst, src

(Як правило, dst=ST(0), src=ST(i) або dst=ST(i), src=ST(0). За замовчуванням (команда FADD) вважається, що dst=ST(0), src=ST(1). Результат операції в ST(0));

- команда віднімання дійсних чисел – FSUB dst, src;
- команда множення дійсних чисел – FMUL dst, src;
- команда ділення дійсних чисел – FDIV dst, src;
- Команда порівняння з нулем вершини стека – FTST;

- команда порівняння дійсних чисел – FCOM src

Остання команда виконує віднімання вмісту src від вмісту вершини стека ST(0), формуючи значення прапорців C3 та C0 (рис. 17).

Відношення	C3 C0
ST(0) > (src)	0 0
ST(0) < (src)	0 1
ST(0) = (src)	1 0
ST(0) не може бути порівняно з (src)	1 1

Рисунок 17 – значення прапорців C3 та C0

Спеціальні команди для обчислень

До спеціальних команд належать:

- команда обчислення кореня квадратного – FSQRT;
- команда обчислення функції sin – FSIN;
- команда обчислення функції cos – FCOS;
- команда обчислення функції arctg – FRATAN.

2.8.4 Приклади програм із використанням математичного співпроцесора

Оскільки введення та виведення даних мовою Асемблер має деякі труднощі, то операції введення з клавіатури та виведення результатів обчислень на екран реалізуємо в програмі мовою Сі, а в неї вбудуємо асемблерний код, в якому власне і проведемо математичні розрахунки.

Приклад 1. Програма для обчислення виразу: result = a·x/b.

```
#include <stdio.h>
#include <conio.h>
void main()
{
      float x,a,b,result,buf;
      int cycle;  b=3.0;   a=4.0;
        clrscr();
      puts("Input x");
```

```
        scanf("%f",&x);
        _asm{
              FLD       B
              FLD       X
              FLD       A
              FLD1
              FMUL      ST(0),ST(1)
              FMUL      ST(0),ST(2)
              FDIV      ST(0),ST(3)
              FST       RESULT
        }
        printf("%f\n",result);
        getch();
}
```

Приклад 2. Програма для обчислення значення функції $y=e^x$, використовуючи розклад функції в ряд Тейлора.

```
#include <stdio.h>
#include <conio.h>
void main()
{
            float x,a,y,i,result;
            int cycle;
            i=0.0;  a=1.0;  y=1.0;
              clrscr();
            puts("Input x");
            scanf("%f",&x);
            puts("Input cycle");
            scanf("%d",&cycle);
            _asm     MOV    CX,CYCLE

            NEXT:
            _asm{
                  FLD1
                  FLD       I
                  FADD      ST(0),ST(1)
                  FST       I
                  FINIT
                  FLD       Y
                  FLD       I
                  FLD       A
                  FLD       X
                  FMUL      ST(0),ST(1)
                  FDIV      ST(0),ST(2)
```

```
            FST     A
            FADD    ST(0),ST(3)
            FST     Y
            LOOP    NEXT
            FLD     Y
            FST     RESULT
        }
        printf("%f\n",result);
        getch();
}
```

Порядок виконання роботи

1. Вивчити теоретичну частину даної лабораторної роботи.
2. Скласти програму мовою Асемблер для обчислення значення заданого виразу згідно з варіантом індивідуального завдання.
3. Скласти програму мовою Сі для обчислення цього виразу і порівняти результати обчислень.

ЛАБОРАТОРНА РОБОТА № 9
Керування дисплеєм

Тема: Знайомство з текстовим та графічним режимами роботи дисплеїв. Програмування графіки мовою Асемблер. Керування дисплеєм шляхом прямого доступу до відеопам'яті диплея та за допомогою функцій BIOS.

Теоретичні відомості

2.9.1 Технічні особливості дисплеїв

Дисплей включає в себе екран з електронно-променевою трубкою, а також комплекс технічних засобів, що забезпечують появу на екрані зображення. В основі його роботи є багато спільного з телевізором: в результаті зіткнення пучка електронів з поверхнею екрана, покритою люмінофором, який світиться, утворюється точка - піксел, що світиться, причому інтенсивність світла залежить від енергії електронного пучка. Змінюючи енергію пучка, можна отримати різну яскравість певного піксела – від темної до максимальної. Електронний промінь "оббігає" екран зліва направо і згори до низу 25 разів у секунду, послідовно формуючи множину близько розташованих пікселів, які, зливаючись один з одним, сприймаються оком як єдине ціле.

Найбільш важливою відмінністю описаного механізму від способу створення зображення на екранах звичайних телевізорів є та обставина, що програміст може керувати світінням екрана в будь-якому його місці, і навіть кожним окремим пікселем. Така можливість досягається ціною значного ускладнення електронних компонентів дисплея у порівнянні з побутовим телевізором.

Через велику кількість пікселів на екрані давати окрему команду для кожного піксела буде досить неефективно, оскільки центральний процесор буде практично постійно зайнятий регенерацією зображення.

Тому між програмою і схемою електронної розгортки зображення розміщується буферна пам'ять (відеопам'ять) на два входи (порти). Програма може звертатись до свого порту, щоб розмістити потрібне значення у відеопам'ять або прочитати з неї раніше встановлене значення так само, як це робиться із звичайною оперативною пам'яттю ЕОМ. Одночасно до свого порту звертаються за зчитуванням інформації з відеопам'яті схеми розгортки зображення. Відносно повільна програма не поспішаючи "кладе" у відеопам'ять окремі фрагменти зображення, в той час як швидкі схеми розгортки безперервно формують з цих фрагментів цільне зображення.

Найбільш важливі електронні компоненти дисплея:

- контролер (схема керування) електронно-променевої трубки;
- порти введення-виведення, що програмуються;
- постійний запам'ятовуючий пристрій (ПЗП), який включає генератор символів та буферна відеопам'ять.

Ці компоненти розташовані на одній друкованій платі, яка називається відеоадаптером.

Існують такі загальноприйняті стандарти відеоадаптерів:

- монохромний дисплейний адаптер MDA (Monochrome Display Adapter);
- кольоровий графічний адаптер CGA (Color Graphics Adapter);
- монохромний графічний адаптер MGA (Monochrome Graphics Adapter) або графічний адаптер "Геркулес" (Hercules Graphics Adapter);
- покращений графічний адаптер EGA (Enhanced Graphics Adapter);
- відеографічна матриця VGA (Video Graphics Array).
- супервідеографічна матриця SVGA (Super Video Graphics Array).

2.9.2 Текстовий режим роботи дисплея

Існує два види інформації, що з'являється на екрані дисплея: текстова, тобто така інформація, яка складається зі знаків алфавіту, цифр та спеціальних символів; і графічна – креслення, рисунки, графіки. Хоча описаний вище механізм формування зображень – загальний для обох видів інформації, є суттєві відмінності у тому, яким чином використовується

відеопам'ять. Ці відмінності настільки істотні, що говорять про два режими роботи дисплеїв – текстовий та графічний. Розглянемо спочатку текстовий режим при відтворенні чорно-білої інформації.

Для найбільш розповсюджених типів дисплеїв на екран може бути виведено 25 рядків по 80 символів в кожному, тобто всього 2000 символів. Кожний символ являє собою один з 256 заздалегідь обумовлених знаків. Для розміщення символу на екрані використовується прямокутна матриця розміром 8×8 = 64 пікселів, яка називається знакомісцем.

На рис.18 показано формат екрана 25×80, а на рис.19 – позиції символів на екрані, пронумеровані цілими числами

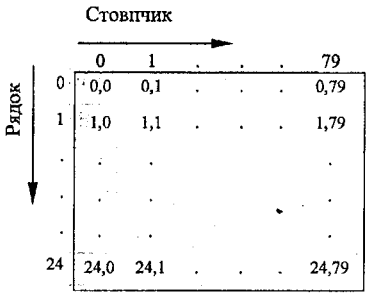


Рисунок 18 – Формат екрана 25×80

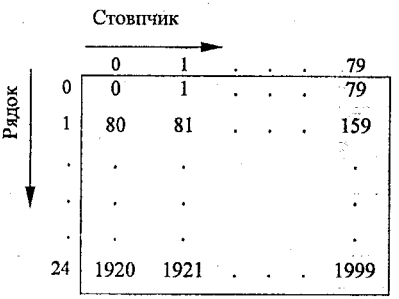


Рисунок 19 – Позиції символів на екрані

Оскільки вигляд будь-якого символу, що виводиться, заздалегідь визначений, для зберігання інформації про нього немає необхідності використовувати матрицю 8×8, тобто 8 байтів відеопам'яті. Досить лише вказати номер символу, тобто помістити в пам'ять лише 1 байт. Використовуючи цей байт як ключ, електронні схеми розгортки відшукають в ПЗП одну з 256 матриць 8×8 розрядів і, відповідно до неї, викреслять символ на екрані.

Насправді ж у відеопам'яті приходиться зберігати не один, а два байти інформації про кожне знакомісце – додатковий байт використовується для зберігання атрибутів символу. Атрибути забезпечують індикацію світлого символу на темному фоні (нормальне зображення) або темного символу на світлому фоні (інверсне зображення), виведення символу з підвищеною яскравістю, виведення символу, що мерехтить, виведення підкресленого символу. Формат байта атрибутів наведений на рис.20 і в табл.5.

Таким чином, для зберігання однієї "карти" (тобто, однієї сторінки) екрана в текстовому режимі при використанні двох кольорів (чорного і білого) необхідно лише 80×25×2 = 4 Кб відеопам'яті. Таким є розмір відеопам'яті адаптера MDA.

Значно більший розмір відеопамяті для відеоадаптера CGA: 4 текстові сторінки в режимі 80×25 символів та 8 сторінок в режимі 40×25 символів, тобто 16 Кб.

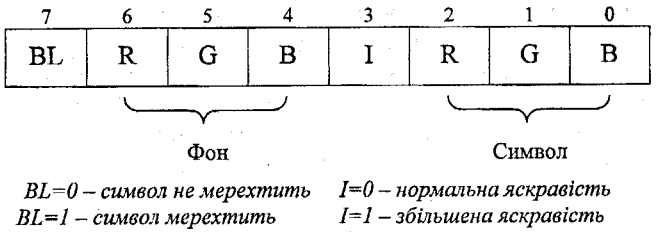


Рисунок 20 – Формат байта атрибутів для відеоадаптера MDA

Таблиця 5 – Атрибути символів для відеоадаптера MDA

Вміст поля RGB для фону	Вміст поля RGB для символу	Атрибут
0 0 0	0 0 0	Не відображається: чорний по чорному (можна використовувати для паролів)
1 1 1	1 1 1	Білий прямокутник (білий по білому)
0 0 0	1 1 1	Нормальне зображення (білий по чорному)
1 1 1	0 0 0	Інверсне зображення (чорний по білому)

Адаптер CGA дозволяє також відтворювати текстову інформацію в кольорі. Колір окремого символу задається за допомогою його атрибута, формат якого наведений на рис.9 та в табл.6.

Рисунок 21 – Формат байта атрибутів для відеоадаптера CGA

У відеопам'яті код символу завжди міститься у байті з парною адресою, а його атрибут – у байті з більшою непарною адресою. Отже, слово з парною адресою дає всю інформацію для однієї символьної позиції.

Хоча відеопам'ять не схожа на основну оперативну пам'ять комп'ютера, до неї можна звертатись з програми за допомогою адрес. Для

відеопам'яті в комп'ютері зарезервовані адреси з A000 по DFFF, тобто 256 Кб.

Таблиця 6 – Атрибути символів для відеоадаптера CGA

Фон		Символ			
RGB	Колір	IRGB	Колір	IRGB	Колір
0 0 0	чорний	0 0 0 0	чорний	1 0 0 0	сірий
0 0 1	синій	0 0 0 1	синій	1 0 0 1	яскраво-блакитний
0 1 0	зелений	0 0 1 0	зелений	1 0 1 0	яскраво-зелений
0 1 1	синьо-зелений	0 0 1 1	синьо-зелений	1 0 1 1	яскраво-синьо-зелений
1 0 0	червоний	0 1 0 0	червоний	1 1 0 0	яскраво-червоний
1 0 1	малиновий	0 1 0 1	малиновий	1 1 0 1	яскраво-малиновий
1 1 0	коричневий	0 1 1 0	коричневий	1 1 1 0	яскраво-жовтий
1 1 1	білий	0 1 1 1	білий	1 1 1 1	яскраво-білий

Відеопам'ять адаптера MDA починається з адреси B000 і займає 4 Кб, для адаптера CGA відеопам'ять починається з адреси B800 і займає 16 Кб. У табл.7 наведені адреси початку текстових сторінок для CGA-дисплеїв.

Таблиця 7 – Початкові адреси текстових сторінок

Номер сторінки	1	2	3	4	5	6	7	8
Режим роботи 80×25	B800	B900	BA00	BB00	-	-	-	-
Режим роботи 40×25	B800	B880	B900	B980	BA00	BA80	BB00	BB80

2.9.3 Приклад програми прямого доступу до відеопам'яті

Код програми, наведеної нижче, демонструє роботу дисплея у текстовому режимі у випадку безпосереднього доступу до сторінок відеопам'яті. У програмі екран диплея спочатку очищається символами "пропуск" на малиновому фоні, а потім за допомогою символів псевдографіки формується зображення рамки яскраво-блакитного кольору.

```
NAME    PROG
STACKSG SEGMENT STACK
        DB   128 DUP(?)
STACKSG ENDS
DATE    SEGMENT
DATE    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATE, SS:STACKSG
START   PROC    FAR
```

```
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        MOV     AX,DATE
        MOV     DS,AX
        CALL    MAIN
        RET
START   ENDP
MAIN    PROC
        MOV     AH,0
        MOV     AL,03h
        INT     10h
        MOV     AX, 0B800h ; початкова адреса текстової сторінки
        MOV     ES,AX
        MOV     DI,0
        MOV     AL,' '     ; задаємо символ для відображення
        MOV     AH,059h    ; задаємо колір: 01011001b
        MOV     CX,2000    ; задаємо кількість повторень символу
        CLD                ; скидаємо прапорець напрямку (інкремент)
        REP  STOSW         ; повторюємо ланцюжок символів (слово AX)
; рисуємо лівий верхній кут
        MOV     DI,660
        MOV     AL,213
        MOV     AH,059h
        MOV     ES:[DI],AX ; розміщуємо AX за адресою DI
                           ; відносно початку ES сторінки
; рисуємо верх рамки (40 символів в рядок зліва направо)
        MOV     CX,40
        MOV     AL,205
M1:     ADD     DI,2
        MOV     ES:[DI],AX
        LOOP    M1
; рисуємо правий верхній кут
        ADD     DI,2
        MOV     AL,184
        MOV     ES:[DI],AX
; рисуємо праву сторону рамки (6 символів у стовпчик згори донизу)
        MOV     AL,179
        MOV     CX,6
M2:     ADD     DI,160
        MOV     ES:[DI],AX
        LOOP    M2
; рисуємо правий нижній кут
```

```
        ADD     DI,160
        MOV     AL,217
        MOV     ES:[DI],AX
; рисуємо низ рамки (40 символів справа наліво)
        MOV     AL,196
        MOV     CX,40
M3:     SUB     DI,2
        MOV     ES:[DI],AX
        LOOP    M3
; рисуємо нижній лівий кут
        SUB     DI,2
        MOV     AL,192
        MOV     ES:[DI],AX
; рисуємо ліву сторону рамки (6 символів знизу вгору)
        MOV     AL,179
        MOV     CX,6
M4:     SUB     DI,160
        MOV     ES:[DI],AX
        LOOP    M4
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

2.9.4 Графічний режим роботи дисплеїв

Як відомо, в ЕОМ прийнято растровий спосіб формування зображень, тобто, будь-яка інформація на екрані дисплея являє собою сукупність точок - пікселів, що світяться. Кожний піксел визначається своїми координатами – розташуванням відносно лівого верхнього кута екрана, який має координати (0,0). Програміст може керувати яскравістю та/або кольором будь-якого піксела, що дозволяє формувати на екрані будь-які зображення, у тому числі рисунки, графіки, креслення, символи.

Як і в текстовому режимі, в графічному режимі використовується буферна відеопам'ять, вміст якої являє собою "карту" екрана. Розмір пам'яті адаптера CGA (16 Кб) достатній для відображення 640×200 пікселів, якщо з кожним пікселем зв'язувати лише один розряд (біт) відеопам'яті. Зрозуміло, що інформаційна цінність одного біта, а отже, і можливості керування пікселем в цьому випадку надзвичайно обмежені: адже один розряд може мати або нульове, або одиничне значення. В комп'ютері прийнято вважати, що нульовий стан будь-якого розряду відеопам'яті означає сигнал про відсутність світіння відповідного піксела, і, навпаки, одиничний стан – сигнал висвітлити відповідний піксел.

Таким чином, об'єм 16 Кб відеопам'яті дозволяє отримати на екрані розміром 640×200 пікселів тільки двоколірне зображення – чорний колір відповідає стану 0 біта відеопам'яті, а стан 1 – виведенню піксела будь-яким заздалегідь вибраним кольором. В рамках CGA-адаптера такий режим називається режимом високої роздільної здатності.

В іншому режимі адаптера CGA – режимі середньої роздільної здатності – на кожний піксел виділяється 2 біти відеопам'яті, але досягається це ціною подвійного погіршення роздільної здатності екрана: 320×200 пікселів.

Вказані два біти відеопам'яті використовуються для вибору кольору піксела.

На перший погляд, за допомогою двох бітів можна вказати лише 2 кольори. Але розробники дисплеїв прийняли інше рішення.

Одне зі значень, що запам'ятовується у парі бітів (значення 00), виділено окремо: воно вказує на те, що піксел має той самий колір, що і фон. Така точка сприймається як така, що не світиться.

Як раніше згадувалось, вісім основних кольорів отримуються в результаті накладання променів червоного, синього та зеленого кольорів (решта вісім кольорів – це ті ж самі основні, але підвищеної яскравості). Таким чином, кольори можна розділити на дві групи, які відрізняються наявністю або відсутністю, наприклад, синьої складової.

У першій групі (параметр "палітра" дорівнює 0) синій промінь не присутній, і всі кольори утворюються сумішшю червоного і зеленого. Якщо пара бітів встановлена в стан 01, то точка загорається червоним кольором, 10 – зеленим, 11 – їх сумішшю, тобто жовтим (табл.8).

Таблиця 8 – Кольори зображення

Колір піксела	Код кольору	Суміш променів	Параметр "палітра"
Червоний	01	червоний	0
Зелений	10	зелений	0
Жовтий	11	червоний + зелений	0
Фіолетовий	01	червоний + синій	1
Бірюзовий	10	зелений + синій	1
Білий	11	зелений + червоний + синій	1

У другій групі кольорів (параметр "палітра" дорівнює 1) синя складова обов'язково присутня. Тому стану 01 відповідає фіолетовий колір, стану 10 – бірюзовий і стану 11 – білий колір (табл.8).

Пари бітів не вистачає для того, щоб запам'ятати яскравість піксела, і тому в режимі графіки середньої роздільної здатності кількість доступних кольорів обмежено числом шість.

В більш сучасних адаптерах EGA та VGA розмір відеопам'яті збільшено і він складає від 64 до 256 Кб. Це дозволяє отримати на екрані з

роздільною здатністю 640×350 пікселів або й більше і до 16 кольорів одночасно.

До відеопам'яті мають можливість одночасного доступу як програма для переустановлення стану будь-якого піксела, так і електронні схеми розгортки зображення на екрані. Ці схеми розглядають відеопам'ять як один довгий рядок бітів, що зберігає команди щодо встановлення яскравості і кольору певного піксела. Електронні схеми опитують відеопам'ять послідовно біт за бітом у темпі електронного променя, що "оббігає" екран.

Зображення стандартного телевізійного кадру формується з двох півкадрів. У першому півкадрі промінь оббігає екран по всіх парних рядках екрана, у другому – по всіх непарних. Відповідним чином зберігається і інформація у відеопам'яті: в першій її половині (з меншими адресами) повинна міститися інформація про парні рядки, у другій половині (адреса початку цієї частини повинні бути кратною 1024) – про непарні. Вбудовані процедури і функції, що підтримують графіку, враховують цю особливість. При прямому зверненні до відеопам'яті ця функція покладається на програміста.

У графічному режимі існує можливість виведення символьної інформації. Символи будь-якої конфігурації користувач може формувати самостійно і представляти їх просто як деякі піктограми, що займають певну частину екрана (за аналогією з текстовим режимом будемо називати частину екрана, зайняту зображенням символу, знакомісцем).

Інший підхід полягає в імітації засобів, реалізованих в текстовому режимі. Нагадаємо, що у текстовому режимі використовується розташована в ПЗП таблиця графічних образів усіх символів, яка керує роботою схем генерації тексту. У графічному режимі при виведенні символів також використовується таблиця знакогенератора, але розташована в ПЗП таблиця підключається лише при виведенні першої половини символів (з кодами від 0 по 127), при виведенні другої половини символів (з кодами від 128 по 256) використовується частина таблиці, що завантажується в оперативну пам'ять. Зроблено це не випадково, оскільки друга половина ASCII-кодів використовується для формування символів національного алфавіту, у тому числі кирилиці. Реалізація цієї частини кодів за допомогою завантажуваної в пам'ять таблиці знакогенератора дуже спрощує проблему "перенавчання" комп'ютера під будь-який національний алфавіт. На жаль, для текстового режиму можливість завантажування таблиці знакогенератора є лише на удосконалених відеоадаптерах EGA, VGA та більш сучасних. При роботі з адаптером CGA "перенавчання" комп'ютера в текстовому режимі роботи дисплея досягається лише методом перепрограмування мікросхеми ПЗП.

У графічному ж режимі роботи адаптера CGA "заставити" комп'ютер виводити текстові повідомлення українською мовою не складає проблеми. Для цього треба лише завантажити в оперативну пам'ять шрифт у вигляді

таблиці з 128 матриць розміром 8×8 бітів і повідомити адресу початку цієї таблиці схемам розгортки. Кожна матриця 8×8 бітів (8 байтів) містить графічний образ символу, що виводиться, причому перші 8 байтів використовуються при виведенні символу з кодом 128, наступні 8 байтів – символу з кодом 129 і т.д. Кожний байт матриці кодує один рядок розгортки з 8 суміжних пікселів: якщо перший біт у байті має значення 1, перший піксель цього рядка буде світитися, якщо ж 0 – не буде. Стан другого пікселя визначається вмістом другого біта і т.д.

2.9.5 Керування дисплеєм за допомогою функцій BIOS

Керувати дисплеєм можна також на основі загальних принципів керування периферійними пристроями.

На початку свого розвитку мікропроцесорні системи мали у своєму складі програму, що називається "монітор". Ця програма зазвичай знаходилась в ПЗП і обслуговувала пристрої введення-виведення – клавіатуру, дисплей, касетний накопичувач на магнітній стрічці та ін. Діалогова частина монітора дозволяла виконувати деякі операторські функції – завантаження і запуск програми, відлагоджування в покроковому режимі, друк текстів, перегляд і редагування вмісту пам'яті і т.д. Але найголовніше – прикладні програми, що були складені для цих систем, могли користуватись модулями монітора для роботи з периферійною апаратурою і виконувати ці операторські функції. Програма вже не містила в собі всі необхідні для її роботи модулі, а користувалась "стандартними" послугами програми-монітора. Така організація програми не тільки зменшувала розмір її завантажувального модуля, а й дозволяла програмістам зосередити свої зусилля на розв'язанні основної задачі.

Механізм взаємодії програми користувача і монітора був реалізований по-різному у різних системах. У найгіршому випадку програма користувалась відомими абсолютними адресами модулів монітора, у найкращому – використовувала спеціальні таблиці адрес програмних модулів. На жаль, різні системи були несумісні за складом модулів монітора і механізмом їх виклику, що сильно утруднювало їх програмну сумісність, а іноді і цілком її виключало.

У першому масовому комп'ютері IBM PC модулі обслуговування стандартної периферії були записані в ПЗП. Сукупність цих модулів (плюс програма початкової ініціалізації і тестування) називається базовою системою введення-виведення – Basic Input/Output System (BIOS).

Комп'ютери, сумісні з IBM PC, що випускаються різними фірмами, можуть дещо відрізнятися за типом периферійного обладнання, але для досягнення сумісності з IBM PC модулі BIOS знімають ці відмінності, надаючи в розпорядження програми користувача стандартний набір модулів для роботи з пристроями введення-виведення.

Не буде перебільшенням сказати, що одна з причин такої популярності комп'ютера IBM PC на ринку персональних комп'ютерів – це наявність добре продуманого стандартного інтерфейсу модулів BIOS і прикладних програм. Саме завдяки цьому інтерфейсу досягається майже стовідсоткова сумісність персональних комп'ютерів даного типу, що випускаються різними фірмами, за їх програмним забезпеченням.

Для виклику відповідного модуля BIOS прикладна програма використовує команду переривання INT <n> з відповідним номером n. Програма передає параметри модулям BIOS через регістри процесора, результат роботи модуля повертається також в регістри.

Для роботи з дисплейним адаптером використовується команда переривання INT 10h. Під час виклику цього переривання, як і під час виклику багатьох інших переривань, регістр AH повинен містити номер функції, яку слід виконати. Для виконання більшості функцій необхідно мати деякі додаткові дані, які повинні бути розміщені в регістрах AL, BX, CX, DX. Надалі ми конкретно розглянемо всі функції, необхідні для керування CGA-дисплеєм. Варто зауважити, що, у порівнянні з безпосереднім доступом до відеопам'яті, використання звернень до BIOS є набагато простішим інструментом, але більшість функцій BIOS характеризуються невисокою швидкодією.

2.9.6 Основні функції BIOS для керування дисплеєм

Дисплей може працювати у декількох режимах. Перелік можливих режимів роботи дисплея наведено в табл.9.

Таблиця 9 – Режими роботи дисплеїв

Номер режиму	Текстовий або графічний	Колір є/немає і скільки	Формат	Тип адаптера
0	текстовий	немає	40 зн. х 80 ряд.	MDA, CGA, EGA
1	текстовий	8 кольорів	40 зн. х 80 ряд.	CGA, EGA, VGA
2	текстовий	немає	80 зн. х 80 ряд.	CGA, EGA, VGA
3	текстовий	8 кольорів	80 зн. х 80 ряд.	CGA, EGA, VGA
4	графічний	6 кольорів	320 х 200 точ.	CGA, EGA, VGA
5	графічний	немає	320 х 200 точ.	CGA, EGA, VGA
6	графічний	немає	640 х 200 точ.	CGA, EGA, VGA
7	текстовий	немає	80 зн. х 80 ряд.	Hercules,EGA,VGA
0Dh	графічний	16 кольорів	320 х 200 точ.	EGA, VGA
0Eh	графічний	16 кольорів	640 х 200 точ.	EGA, VGA
0Fh	графічний	немає	640 х 350 точ.	EGA, VGA
10h	графічний	4 або 16 кольорів	640 х 350 точ.	EGA, VGA
11h	графічний	немає	640 х 480 точ.	VGA
12h	графічний	16 кольорів	640 х 480 точ.	VGA
13h	графічний	256 кольорів	320 х 200 точ.	VGA

Для зручності зведемо всі функції BIOS для керування роботою дисплеїв у таблицю 10.

Таблиця 10 – Функції переривання INT 10h для керування роботою дисплеїв

Номер функції	Призначення	Вміст регістрів	Пояснення
00h	Задання режиму дисплея	*На вході:* AH = 00h AL = номер режиму (табл.9)	Функція використовується для вибору режиму роботи дисплея.
01h	Задання типу курсора	*На вході:* AH = 01h CH = номер початкової лінії AL = номер кінцевої лінії. *На виході:* немає	Функція дозволяє встановити розмір курсора. Передбачено визначення першої та останньої лінії (лінії розгортки), що обмежують область розташування символу (установлення початкової лінії вгорі області, а установлення кінцевої лінії внизу області не обов'язкове, тобто курсор може бути встановлено посередині описаної області). Слід зауважити, що в графічному режимі курсор відсутній. Курсор може бути відключений шляхом установлення початкової і кінцевої лінії нижче рівня розташування курсора.
02h	Задання позиції курсора	*На вході:* AH = 02h DH = номер рядка DL = номер стовпчика BH = номер сторінки *На виході:* немає	Функція видає координати для позиціонування курсора па екрані. Рядок 0 розташований вгорі, а стовпчик 0 – в лівій частині екрана.
03h	Зчитування позиції курсора	*На вході:* AH = 03h BH = номер сторінки *На виході:* DH =поточний рядок DL = поточний стовпчик CH = початкова лінія курсора CL = кінцева ліні *На виході:* немає	Функція дозволяє отримати координати позиції курсора на екрані та його тип.
05h	Задання активної сторінки екрана, тобто, який з екранів буде	*На вході:* AH = 05h AL = номер сторінки, що виводиться на	Адаптер дозволяє використовувати декілька сторінок (екранів) інформації в пам'яті. Видимою в певний момент може бути лише одна – активна

Продовження таблиці10

05h	Задання активної сторінки екрана, тобто, який з екранів буде виведено на дисплей (за замовчуванням – 0)	*На вході:* AH = 05h AL = номер сторінки, що виводиться на екран дисплея *На виході:* немає	Адаптер дозволяє використовувати декілька сторінок (екранів) інформації в пам'яті. Видимою в певний момент може бути лише одна – активна сторінка. Більшість функцій можуть модифікувати екран (запис символу, побудова точки і т.д.) і вибирати сторінку, а отже вносити зміни в невидиму сторінку. Тобто на екрані може бути одне зображення, в той час як ми модифікуємо іншу сторінку (зручно при мультиплікації і виведенні слайдів).
06h	Прокручування (скролінг) активної сторінки вгору	*На вході:* AH = 06h AL = число рядків, на яке слід "прокрутити" (0 – для очистки екрана) BH = символ атрибута нових рядків CH = верхній рядок вікна CL = лівий стовпчик вікна DH = нижній рядок вікна DL = правий стовпчик вікна *На виході:* немає	За допомогою цієї функції виконується прокручування тексту на екрані: рядки переміщуються знизу вгору, а внизу здійснюється вставка пустих рядків. Можуть бути визначені координати вікна, яке прокручується, оскільки лише частина екрана може підлягати прокручуванню.

Продовження таблиці10

07h	Прокручування (скролінг) активної сторінки вниз	*На вході:* AH = 07h AL = число рядків, на яке слід "прокрутити" (0 – для очистки екрана) BH = символ атрибута нових рядків CH = верхній рядок вікна CL = лівий стовпчик вікна DH = нижній рядок вікна DL = правий стовпчик вікна *На виході:* немає	Виконується прокручування тексту на екрані: рядки переміщуються згори донизу, а вгорі здійснюється вставка пустих рядків. Функція працює таким самим чином, як і 05h.
08h	Зчитування атрибута і сим-волу в поточній позиції курсора	*На вході:* AH = 08h BH = номер сторінки *На виході:* AL = ASCII-код символу AH = атрибут символу	Функція може бути використана для зчитування символу з будь-якої сторінки. У результаті повертається символ, розташований в поточній позиції курсора вказаної сторінки і його атрибут. Атрибут являє собою один байт згідно з рис.8 і табл.6. Байт атрибута має сенс лише в текстових режимах.
09h	Установлення атрибута символу і запис символу в поточну позицію курсора	*На вході:* AH = 09h AL = ASCII-код символу BH = номер сторінки BL = атрибут символу CX = кількість записуваних символів *На виході:* немає	В графічних режимах байт атрибуту використовується для установлення кольору символу. Установлення 7-го біта байта атрибуту приводить до того, що символ накладається на поточний вміст екрана. В текстових режимах виведення більшого, ніж може поміститися, числа копій символу призведе до перенесення на наступний рядок. У графічному режимі всі символи повинні розміщатися в одному рядку. Позиціонування символу повинно виконуватись засобами програми. Виведення керуючих символів виконується як виведення звичайних символів.

Продовження таблиці 10

0Ah	Запис символу у поточну позицію курсора	*На вході:* AH = 0Ah AL = ASCII-код символу BH = номер сторінки CX = кількість символів, що записуються *На виході:* немає	Звернення до цієї функції ідентичне зверненню до функції 09h, за винятком того, що атрибут не може бути установлений (існуючі атрибути не змінюються).
0Bh	Задання палітри кольорів	*На вході:* AH = 0Bh BH = ідентифікатор фону або палітри кольорів BL = номер кольору *На виході:* немає	При BH = 0 у графічному режимі регістр BL визначає колір фону (значення 0-15), а у текстових режимах – колір границі (значення 0-31).
0Ch	Запис точки	*На вході:* AH = 0Ch AL = номер кольору BH = номер сторінки (лише для EGA та VGA) CX = номер стовпчика піксела (0-319 або 0-639) DX = номер рядка (0-199, 0-349 або 0479) *На виході:* немає	Звернення до цієї функції застосовується для побудови точки на будь-якій сторінці в графічних режимах. Стовпчик 0 розташований зліва, а рядок 0 – вгорі екрана. Якщо в регістрі AL встановлений біт 7, то новий піксел накладається на поточний вміст екрана з використанням функції "виключне АБО". Регістр BH (номер сторінки) CGA не підтримує.
0Dh	Зчитування точки	*На вході:* AH = 0Dh BH = номер сторінки (лише для EGA та VGA) CX = номер стовпчика піксела (0-319 або 0-639) DX = номер рядка (0-199, 0-349 або 0-479) *На виході:* AL = значення кольору точки *На виході:* немає	Звернення до цієї функції використовується для отримання кольору точки, розташованої на будь-якій сторінці у графічному режимі (для CGA лише одна сторінка). Стовпчик 0 розташований зліва, а рядок 0 – вгорі екрана.

2.9.7 Приклади програми для виведення зображення у текстовому режимі з використанням функцій BIOS

Наведена нижче програма демонструє прийоми роботи з дисплеями у текстовому режимі за допомогою функцій BIOS. На екрані дисплея на

чорному фоні за допомогою символів псевдографіки виводиться вікно синього кольору, в середині якого виведено рядок "Приклад 2".

```
        NAME   PROG
CURSOR  MACRO  ; макрос для встановлення курсора в певну позицію
        MOV    AH,02H
        MOV    BH,0
        INT    10h
ENDM
ATTR    MACRO  ; макрос для виведення символу
        MOV    BH, 0
        MOV    AH, 09H
        INT    10h
ENDM
STACKSG SEGMENT STACK
   DB   128 DUP(?)
STACKSG ENDS
DATE    SEGMENT
TEXT    DB   "ПРИКЛАД", 0
DATE    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATE, SS:STACKSG
START   PROC   FAR
        PUSH   DS
        SUB    AX,AX
        PUSH   AX
        MOV    AX,DATE
        MOV    DS,AX
        CALL   MAIN
        CALL   STROKA
        RET
START   ENDP
MAIN    PROC
        MOV    AH,0 ; задаємо режим дисплея
        MOV    AL,3
        INT    10h
        MOV    DH,5 ; позиція курсора
        MOV    DL,10
 MET2:
        CURSOR          ; встановлюємо курсор в задану позицію
```

```
            MOV     CX,60   ; кількість повторень
            MOV     AL,219  ; символ псевдографіки
            MOV     BL,059h ; задаємо колір
            ATTR            ; виводимо символ із заданими параметрами
            INC     DH
            CMP     DH,10
            JNE     MET2
            RET
MAIN        ENDP

    ; процедура для виведення рядка посимвольно
STROKA      PROC    NEAR
            MOV     DH,7        ; позиція курсора
            MOV     DL,30
            CURSOR              ; встановлюємо курсор
            LEA     SI,TEXT-1   ; завантажуємо адресу рядка
 MET3:
            INC     SI
            MOV     AL,[SI]
            MOV     BL,07Ch     ; атрибути символу
            MOV     CX,1
            ATTR                ; виводимо символ
            INC     DL
            CURSOR
            MOV     AL,[SI]
            CMP     AL,0        ; перевірка на кінець рядка
            JNE     MET3
            MOV     DL,'2'
            MOV     AH, 06H
            INT     21h
            MOV     DH,22
            MOV     DL,1
            CURSOR
            RET
STROKA      ENDP
            ENDS
END         START
```

2.9.8 Приклад програми для виведення зображення у графічному режимі з використанням функцій BIOS

Наведена нижче програма демонструє прийоми роботи з дисплеями у графічному режимі за допомогою функцій BIOS. На екрані дисплея на

чорному фоні за допомогою пікселів виводиться пряма лінія синього кольору.

```
NAME    PROG1
STACKSG SEGMENT STACK
        DB   128 DUP(?) ; резервування пам'яті для стека
STACKSG ENDS
DATE_SG SEGMENT
        CON1    DW  100
        CON2    DW  50
DATE_SG ENDS
CODE_SG SEGMENT
        ASSUME  CS:CODE_SG, DS:DATE_SG, SS:STACKSG
START   PROC    FAR
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        CALL    MAIN
        RET
START   ENDP
MAIN PROC
        MOV     AH,00h      ; установлення режиму
        MOV     AL,4h       ;
        INT     10h
        MOV     AH,0Bh      ; установлення палітри кольорів
        MOV     BH,1        ; для фону зображення
        MOV     BL,1        ; номер кольору

        INT     10h
        MOV     AH,0Bh      ; установлення режиму
        MOV     BH,0        ; для виведення рисунка
        MOV     BL,6        ; номер кольору
        INT     10h
        MOV     SI,CON2     ; кінцева координата
        MOV     CX,CON1     ; початкова координата (№ стовпчика)
        MOV     DX,CON2     ; номер рядка
MET1:   MOV     AH,0Ch      ; запис точки
        MOV     AL,2        ; номер кольору

        INC     CX
        INT     10h
        DEC     SI
        CMP     SI,0
```

```
        JNE     MET1
        RET
MAIN    ENDP
CODE_SG ENDS
        END     START
```

Порядок виконання роботи

1. Вивчити теоретичну частину даної лабораторної роботи.
2. Скласти програму для виведення зображення на екран дисплея згідно з індивідуальним завданням, використовуючи прямий доступ до відеопам'яті.
3. Скласти програму для виведення зображення на екран дисплея згідно з індивідуальним завданням, використовуючи функції BIOS.

ЛАБОРАТОРНА РОБОТА № 10

Робота з файлами

Тема: Програмування основних операцій над файлами мовою Асемблер: створення, відкриття, закриття файлів. Введення даних у файл (пристрій) і виведення даних з файла (пристрою). Програмування операцій з атрибутами файлів. Робота з дисками, каталогами, пошук файлів.

Теоретичні відомості

2.10.1 Класи функцій для роботи з файлами

В операційній системі MS DOS існують два класи функцій для роботи з файлами:

- функції, що використовують блок керування файлом (FCB);
- функції, що використовують програмний канал (Handle).

Перший клас функцій (функції 0Fh – 24h) застосовувався в MS DOS версій 1.x і на даний час має чисто історичний інтерес.

Другий клас функцій (функції 39h – 62h) з'явився вперше в MS DOS версії 2.0 і широко використовується дотепер.

Сутність каналу (інші назви – дескриптор файла, файловий індекс, логічний номер) полягає у наступному. Для того, щоб почати роботу з файлом або пристроєм, програма повинна відкрити цей файл або пристрій. Процес відкриття файла або пристрою полягає у присвоєнні йому певного номера каналу й виконання деяких інших ініціалізуючих дій.

Після закінчення роботи з файлом або пристроєм його необхідно закрити, при цьому канал, що йому відповідає, стає вільним.

Перші п'ять номерів каналів зарезервовані операційною системою для таких стандартних пристроїв:

0 – Stdin – стандартний пристрій введення (клавіатура);
1 – Stdout – стандартний пристрій виведення (екран диплея);
2 – Stderr – стандартний пристрій для виведення повідомлень про помилки (екран диплея);
3 – Stdaux – стандартний пристрій зв'язку (послідовний адаптер COM1);
4 – Stdptr – стандартний пристрій друку (перший паралельний порт LPT1).

Ці пристрої доступні прикладним програмам, їх не треба відкривати, але програми можуть закривати їх. Пристрій Stdaux може використовуватись як для введення, так і для виведення даних.

Наведені вище пристрої при початковому завантаженні операційної системи знаходяться у символьному режимі.

2.10.2 Створення файлів

Створюється файл за допомогою функції 3Ch переривання INT 21h. для виклику цієї функції необхідно помістити в регістри таку інформацію:

На вході:

AH = 3Ch.

CX = Атрибути файла, який буде створено, а саме:

00 – звичайний файл;
01 – тільки для читання;
02 – прихований файл;
04 – системний файл.

DS:DX = адреса ASCII-рядка, що містить шлях по каталогу для файла, що створюється.

На виході:

після завершення роботи функції в регістрі AX знаходиться номер каналу цього файла або вказується код помилки, якщо файл не було створено.

Додатково ця ж функція виконує також операцію відкриття щойно створеного файла. Варто зауважити, що якщо файл із вказаним ім'ям вже існує, він обрізається до нульової довжини. І тому, щоб випадково не знищити вміст файла з таким самим ім'ям, як і той, що створюється, можна використовувати функцію 5Bh переривання INT 21, яка виконує перевірку наявності файла з таким іменем.

Можна для відкриття або створення файла з розширеними можливостями використовувати більш сучасну функцію 6Ch, яка з'явилася

в останніх версіях MS DOS (DOS 4.0++). З її появою зникає необхідність відслідковувати існування файла, що створюється. Для коректної роботи цієї функції слід лише задати потрібні значення у відповідних регістрах.

2.10.3 Відкриття файла

Відкриття існуючого файла виконується за допомогою функції 3Dh переривання INT 21h. Для виклику цієї функції необхідно розмістити в регістрах таку інформацію:

На вході:

AH = 3Dh.

AL = байт режиму доступу.

DS:DX = адреса ASCII-рядок, що містить шлях по каталогу файла, який відкривається.

На виході:

AX містить номер каналу відкритого файла або код помилки, якщо файл не був відкритий.

Байт режиму доступу визначає правила доступу до файла. Цей байт складається з чотирьох полів, і побітово поля в ньому розподіляються таким чином, як це наведено у таблиці 11.

Таблиця 11 – Вміст байта режиму доступу

Режим доступу	Біти	Призначення	Можливі значення
<I>	7	прапорець успадкування	0 – файл передається породженому процесу; 1 – файл являє собою власність даного процесу.
<S>	6,5,4	поле режиму розподілу	000 – спільне використання файла; 001 – захист від читання і запису; 010 – захист від запису; 011 – захист від читання; 100 – без захисту.
<R>	3	резервне поле	0
<A>	2,1,0	вид доступу	000 – доступ на читання; 001 – доступ на запис; 010 – доступ на читання і запис.

Усі інші комбінації заборонені.

Розглянемо детальніше сутність режимів розділення й видів доступу.

Режим розділення інформує операційну систему про те, які операції над файлом є доступними іншим процесам (програмам). За замовчування

(режим спільного використання) забороняється всім іншим комп'ютерам мережі звертатись до цього файла.

Іноді виникає потреба дозволити іншим програмам виконувати читання з файла, поки наша програма працює з ним. У цьому випадку необхідно встановити захист від запису, який забороняє записувати у файл іншим процесам, але дозволяє зчитувати з нього.

Важливо також вказати, які операції програма збирається здійснювати з файлом (вид доступу). Встановлений за замовчуванням вид доступу (доступ на зчитування і запис) приводить до того, що запит на відкриття файла не дозволяється, якщо інший процес на цьому або іншому комп'ютері мережі відкрив даний файл в режимі розділення, відмінному від режиму "без захисту". Але якщо необхідно лише читати з файла, то можна відкрити файл, який вже відкритий іншим процесом з режимом розділення з "захистом від запису", встановивши йому "доступ на читання" (отже, підвищується доступність файла).

Питання про використання файла декількома процесами залежить від режиму розділення та виду доступу.

2.10.4 Закриття файла

Закриття файла виконується за допомогою функції 3Eh переривання INT 21h. Для виклику цієї функції необхідно помістити у регістри таку інформацію:

На вході:

AH = 3Eh;

BX = номер каналу файла;

DS:DX = адреса ASCII-рядка, що містить шлях по каталогу файла, що закривається.

На виході:

AX містить код помилки.

Після виконання цієї функції файл закривається, елемент каталога оновлюється, а всі системні буфери, що відведені для цього файла, звільняються.

2.10.5 Читання даних з файла або від пристрою

Для пересилання заданої кількості байтів у буфер із файла або пристрою символьної обробки (клавіатури) використовується функція 3Fh переривання INT 21h. Для виклику цієї функції необхідно помістити у регістри таку інформацію:

На вході:

AH = 3Fh;

BX = номер каналу відкритого файла;

CX = кількість байтів, що передаються;
DS:DX = адреса буфера для даних.

На виході:

AX = кількість переданих байтів або код помилки, якщо передача даних не відбулася.

2.10.6 Запис даних у файл або на пристрій

Для пересилання заданої кількості байтів із буфера у файл або на пристрій символьної обробки (дисплей) використовується функція INT 40h переривання 21h. Для виклику цієї функції необхідно помістити таку інформацію у регістри:

На вході:

AH = 40h;
BX = номер каналу відкритого файла;
CX = кількість байтів, що передаються;
DS:DX = адреса буфера для даних.

На виході:

AX = кількість переданих байтів або код помилки, якщо передача даних не відбулася.

2.10.7 Вилучення файла

Файл може бути вилучено функцією 41h переривання 21h. Специфікація цієї функції наведена нижче.

На вході:

AH = 41h;
DS:DX = ASCII-рядок з іменем файла;
CL = атрибути файла, що вилучається.

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:
AX = 2 – файл не знайдено;
AX = 3 – невірно задано шлях;
AX = 5 – відказано в доступі.

Функція 41h не дозволяє вилучати файли з атрибутом "тільки для читання". В цьому випадку слід змінити атрибути файла, що вилучається, за допомогою функції 43h.

2.10.8 Приклад програми роботи з файлом

Нехай, наприклад, необхідно створити файл FILE3.DAT у підкаталозі E:\TOM і записати в нього п'ять одноразрядних чисел, розділених пропусками.

Для розв'язання вказаної задачі необхідно спочатку створити файл, потім переслати в нього 9 символів (цифри разом із пропусками), а після цього закрити файл. Така програма буде мати вигляд, наведений нижче.

```
        NAME    PROG2
SSTACK  SEGMENT STACK
        DB      128 DUP (?)
SSTACK  ENDS
DATA    SEGMENT
ADDR    DB      "E:\TOM\FILE3.DAT",0
BUF     DB      "3 5 7 9 1"
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA, SS:SSTACK
START   PROC    FAR
        PUSH    DS
        SUB     AX,AX
        PUSH        AX
        MOV     AX, DATA
        MOV     DS, AX
        CALL    MAIN
        RET
START   ENDP
MAIN    PROC
        MOV     AH, 03Ch
        MOV     CX, 0
        MOV     DX, OFFSET ADDR
        INT     21h
        MOV     BX, AX
        MOV     AH, 40h
        MOV     CX, 9
        MOV     DX, OFFSET BUF
        INT     21h
        MOV     AH, 03Fh
        INT     21h
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

2.10.9 Методи доступу до файлів

Існує два основних методи доступу до файлів: послідовний і прямий.

У файлах із послідовним методом доступу (їх називають також послідовними файлами) елементи даних (рядки) можуть бути змінної довжини і розділятися між собою парою символів: спочатку символом CR повернення каретки (ASCII-код - 13), а потім символом LF переведення рядка (ASCII-код – 11).

У файлах з прямим доступом заздалегідь відводять фіксоване місце під кожний елемент даних. Якщо деякий елемент не займає все відведене під нього місце, то залишок заповнюється пропусками. У цьому випадку, знаючи розмір, який відводиться під елемент даних, досить легко визначити місце розташування будь-якого елемента даних за його номером.

Для пошуку потрібного елемента даних використовується файловий покажчик, який може бути встановлений у потрібну позицію за допомогою функції 42h переривання INT 21h. Для виклику цієї функції необхідно заповнити регістри такою інформацією:

На вході:

AH = 42h;

AL = вид доступу:

0 – абсолютне зміщення від початку файла;

1 – зміщення від поточної позиції;

2 – зміщення відносно кінця файла;

CX = старший байт зміщення;

DX = молодший байт зміщення;

BX = номер каналу файла.

На виході:

DX = перший байт поточної позиції покажчика,

AX = молодший байт покажчика або код помилки при невиконанні функції.

За допомогою цієї функції можна визначити розмір файла, що досить корисно для послідовних файлів.

2.10.10 Алгоритм запису даних у послідовний файл

Для запису даних у послідовний файл слід виконати такі дії:

1) Створити файл за допомогою функції 3Ch.
2) Ввести дані (з клавіатури або з іншого файла) в область буфера:

   BUF DB 30 DUP (?), 13, 10

   (тут під елемент відводиться 30 байтів).
3) Виконати запис даних у файл за допомогою функції 40h.
4) Закрити файл за допомогою функції 3Eh.

Для додавання даних в існуючий файл необхідно виконати такі дії:

1) Відкрити файл за допомогою функції 3Dh.
2) Установити покажчик файла в його кінець за допомогою функції 42h;
3) Ввести дані в область буфера.
4) Виконати запис даних у файл за допомогою функції 40h.
5) Закрити файл за допомогою функції 3Eh.

2.10.11 Алгоритм читання даних із послідовного файла

При читанні даних із послідовного файла необхідно виконати таку послідовність дій:

1) Відкрити файл за допомогою функції 3Dh.
2) Визначити довжину файла за допомогою функції 42h шляхом встановлення покажчика на його кінець.
3) Встановити покажчик файла на його початок за допомогою функції 42h.
4) Переслати дані з файла в область буфера за допомогою функції 3Fh.
5) Вивести дані з буфера на пристрій або в інший файл за допомогою функції 40h.
6) Закрити файл за допомогою функції 3Eh.

Якщо потрібно здійснити операцію введення-виведення в середину послідовного файла, тоді n-й елемент даних знаходиться методом підрахунку n розділювачів CR/LF.

2.10.12 Операції читання та запису даних у файлах прямого доступу

Операції читання та запису даних у файлах прямого доступу відрізняється від аналогічних операцій з послідовними файлами лише способом пошуку потрібного елемента даних. Позиція n-го елемента даних дорівнює добутку розміру місця, що відводиться під елемент даних, на число (n-1).

2.10.13 Приклад програми читання даних із послідовного файла

Нехай, наприклад, необхідно вивести на екран дисплея вміст файла D:\TOM\PRIM.ASM. Така програма буде мати вигляд, наведений нижче.

```
        NAME    PROG3
SSTACK  SEGMENT STACK
        DB      128 DUP (?)
SSTACK  ENDS

DATA    SEGMENT
ADDR    DB      "D:\TOM\PRIM.ASM",0
BUF     DB      1000 DUP (?)
HANDLE  DW      ?
```

```
SIZE1   DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA, SS:SSTACK
START   PROC    FAR
        PUSH    DS
        SUB     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS,AX
        CALL    MAIN
        RET
START   ENDP
MAIN    PROC
    ; відкриття файла
        MOV     AH, 03Dh
        MOV     AL, 0C2h
        MOV     DX, OFFSET  ADDR
        INT     21h
        MOV     HANDLE, AX
    ; визначення розміру файла
        MOV     BX, AX
        MOV     AH, 42h
        MOV     AL, 2
        MOV     CX, 0
        MOV     DX, 0
        INT     21h
        MOV     SIZE1, AX
    ; установлення покажчика на початок файла
        MOV     AH, 42h
        MOV     AL, 0
        MOV     CX, 0
        MOV     DX, 0
        INT     21h
    ; пересилання даних із файла у буфер
        MOV     AH, 03Fh
        MOV     CX, SIZE1
        MOV     SX, OFFSET  BUF
        INT     21h
    ; виведення даних із буфера на екран дисплея
        MOV     BX, 1
        MOV     AH, 040h
        MOV     CX, SIZE1
```

```
        MOV     DX, OFFSET BUF
        INT     21h
    ; закриття файла
        MOV     BX, HANDLE
        MOV     AH, 03Fh
        INT     21h
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

Дана програма по суті виконує команду DOS, яка видає вміст файла PRIM.ASM на екран дисплея:

```
TYPE D:\TOM\PRIM.ASM
```

2.10.14 Робота з атрибутами файлів

Операційна система дозволяє отримати для аналізу і при необхідності змінити ім'я файла, атрибути файла, дату і час його останньої модифікації. Для цього призначені функції 43h, 56h, 57h переривання 21h. При цьому для функції 43h підфункція 00 призначена для отримання, а підфункція 01 – для встановлення слова атрибутів файла. Аналогічно для функції 57h підфункції 00 та 01 дозволяють отримати та змінити дату і час відповідно.

Отримання атрибутів файла

На вході:

AX = 4300h;

DS:DX = ASCII-рядок з повним іменем файла.

На виході:

якщо прапорець CF = 0, тоді в регістрі CX слово атрибутів файла, причому його формат такий:

7-й біт – розділяється в NovellNetWare;
6-й біт – не використовується;
5-й біт – архівний;
4-й біт – каталог;
3-й біт – мітка тома;
2-й біт – системний;
1-й біт – прихований;
0-й біт – тільки для читання.

якщо CF = 1, тоді AX містить код помилки:

AX = 1 – невірне значення в AL;
AX = 2 – файл не знайдено;
AX = 3 – вказаний шлях не існує;
AX = 5 – доступ заборонено.

Наведемо фрагмент коду, що демонструє отримання атрибутів файла.

```
. . .
.DATA
FNAME    DB   "MYFILE.ASM"
ADDRESS  DD   FNAME
. . .
.CODE
. . .
LDS        DX, ADDRESS
MOV        AX,4300H          ;номер функції DOS
INT  21H
JC   EXIT                    ;Перехід у випадку помилки
. . .
; Тепер в CX – атрибути файла
. . .
EXIT:
. . .
```

Встановлення атрибутів файла

На вході:

AX = 4301h;
CX = нове слово атрибутів файла;
DS:DX = ASCII-рядок з повним іменем файла.

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:
AX = 1 – невірне значення в AL;
AX = 2 – файл не знайдений;
AX = 3 – вказаний шлях не існує;
AX = 5 – доступ заборонений.

Перейменування файла

На вході:

AH = 56h;
DS:DX = ASCII-рядок з іменем існуючого файла;
ES:DI = ASCII-рядок з іменем нового файла;

На виході:

якщо прапорець CF = 0, тоді перейменування файла виконано успішно;

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 1 – невірне знaчення в AL;

AX = 2 – файл не знайдений;

AX = 3 – вказаний шлях не існує;

AX = 5 – доступ заборонений;

AX = 11h – у разі, якщо пристрої для нового і старого файлів не збігаються.

Дана функція дозволяє здійснити переміщення між каталогами, не змінюючи пристрою. Наведемо фрагмент коду, що демонструє переміщення між каталогами, не змінюючи пристрою.

```
...
.DATA
OLDNAME     DB   "MYFILE1.ASM",0
ADDROLD     DD   OLDNAME
NEWNAME     DB   "E:\MYFILE1.ASM",0
ADDRNEW     DD   NEWNAME
...
.CODE
...
    LDS   DX, ADDROLD
    LES   DI,  ADDRNEW
    MOV   AH,56H
    INT   21H
    JC    EXIT        ;Перехід у випадку помилки
...
EXIT:
...
```

Отримання дати і часу створення або останньої модифікації файла

На вході:

AX = 5700h;

BX = дескриптор файла.

На виході:

якщо прапорець CF = 0, тоді в CX і DX інформація про час і дату (табл.12):

CX = час;

DX = дата.

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 1 – заборонений номер підфункції в AL;

AX = 6 – заборонений дескриптор.

Таблиця12 – Формат часу і дати створення або модифікації файла

Час		Дата	
Біти	Опис	Біти	Опис
15-11	Години (0-23)	15-9	Рік
10-5	Хвилини	8-5	Місяць
4-0	Секунди	4-0	День

Встановлення дату і час створення або останньої модифікації файла

На вході:

AX = 5701h;
BX = дескриптор файла;
CX = новий час;
DX = нова дата.

На виході:

якщо прапорець CF = 0, тоді в CX і DX інформація про встановлені час і дату:

CX = час;
DX = дата.

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 1 – недопустимий номер підфункції в AL;
AX = 6 – недопустимий дескриптор.

2.10.15 Робота з дисками, каталогами та організація пошуку файла

Під час роботи з попередніми програмами ми при вказанні імен файлів практично не вказували імен дисководів і шляхів до файлів. Операційна система має у своєму арсеналі засоби для встановлення поточного диску та каталогу, в якому виконуватимуться всі поточні операції з файлами. Та й задача пошуку потрібних файлів за їх маскою є завжди актуальною.

Отримання номера заданого за замовчуванням дисководу

На вході:

AH = 19h.

На виході:

AL = номер дисководу (00h – A:, 01h – B: і т.д.).

Фрагмент коду, що отримує номер поточного дисководу, може виглядати так:

```
. . .
.CODE
. . .
MOV     AH,19H    ;номер функції DOS
INT     21H
```

```
JC          EXIT        ;перехід у випадку помилки
; в al - номер поточного диска
EXIT:
...
```

Установлення номера дисководу

На вході:

AH = 0Eh.

DL = номер дисковода (00h – A:, 01h – B: і т.д.).

На виході:

AL = максимально можливий в даній системі номер дисковода (00h – A:, 01h – B: і т.д.), визначається на основі параметра LASTDRIVE у файлі CONFIG.SYS.

Отримання інформації про вільний дисковий простір

На вході:

AH = 36h;

DL = номер дисковода (00h – A:, 01h – B: і т.д.).

На виході:

AX = FFFFh - неправильний номер пристрою в DL;

Інакше (якщо дисковод вказано правильно):

AX = число секторів в одному кластері;

BX = число вільних кластерів;

CX = розмір сектора (в байтах);

DX = загальне число секторів на диску.

Таким чином, використовуючи інформацію, яку повертає функція 36h, можна підрахувати такі характеристики:

- вільний простір на диску – добуток AX·BX·CX,
- повний об'єм диску – добуток AX·CX·DX.

Створення нового каталога

На вході:

AH = 39h;

DS:DX = ASCII-рядок, вказує шлях до каталогу, що створюється.

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 3 – вказаний шлях не існує;

AX = 5 – доступ заборонений.

Шлях до каталога повинен містити послідовність усіх каталогів. починаючи від кореневого (причому вони повинні існувати). Останнє ім'я каталога – ім'я каталога, що створюється.

Для прикладу наведемо фрагмент коду програми, що демонструє створення каталога.

```
...
.DATA
DNAME   DB   "C:\WINDOWS\MY_DIR",0
ADDR    DD   DNAME
...
.CODE
LDS     DX, ADDR
MOV     AH,39H
INT     21H
JC      EXIT
...
EXIT:
...
```

Знищення каталога

На вході:

AH = 3Ah;

DS:DX = ASCII-рядок зі шляхом до файла, що знищується.

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 3 – вказаний шлях не існує;

AX = 5 – доступ заборонено;

AX = 10h – спроба знищення поточного каталогу.

Зауважимо, що каталог, який знищується, повинен бути пустим.

Зміна поточного каталога

MS DOS дозволяє встановити поточний каталог для того, щоб не вказувати повний шлях для наступних операцій з файлами. При необхідності можна отримати повний шлях до поточного каталога у вигляді ASCII-рядка.

На вході:

AX = 3Bh;

DS:DX = покажчик на буфер. що містить повний шлях від кореневого каталога у вигляді ASCII-рядка (до 64 байт).

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 03h – шлях не знайдено.

Отримання поточного каталога

На вході:

AH = 47h;

DL = номер пристрою (00h – поточний, 01h – A: і т.д.).

DS:DX = покажчик на 64-байтовий буфер для запису повного шляху від кореневого каталога (ASCII-рядок).

На виході:

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 0Fh – недопустимий номер дисковода.

Пошук файла за заданим шаблоном

Для пошуку в каталогах використовується пара функцій 4Eh та 4Fh. В імені файла, що розшукується, можуть бути символи шаблону '*' та '?'. Спочатку викликається функція 4Eh. В якості параметрів їй передаються адреса ASCII-рядка, в якому міститься шлях до файла, та комбінація його атрибутів. Ім'я файла може бути задано у вигляді шаблону. У разі успіху (CF=0), тобто при виявленні першого підходящого до шаблону файла, функція розміщує його ім'я і розширення в область DTA зі зміщенням 1eh від її початку. Далі можна або відкрити файл, або продовжити пошук, але вже використовуючи функцію 4Fh.

При роботі з шаблоном функцію 4Fh можна викликати циклічно, до тих пір, поки в процесі перебору не будуть розглянуті імена всіх підходящих файлів. Про це можна дізнатися за станом прапорця CF (він прийме значення 1 у випадку, коли файлів, що задовольняють шаблон, в даному каталозі більше немає).

Для першого пошуку файла, що відповідає шаблону, використовується функція 4Eh.

На вході:

AH = 4Eh;

CX = атрибути файла (біти 0 та 5 ігноруються);

DS:DX = ASCII-ім'я файла (можливо, зі шляхом до нього і символами шаблона '*' та '?').

На виході:

якщо прапорець CF = 0, тоді в DTA повертається блок даних для першого знайденого файла.

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 2 – файл не знайдено;

AX = 3 – вказаний шлях не існує;

AX = 12h – більше файлів у каталозі немає.

Область DTA (Data Transfer Area) розташовується у префіксі програмного сегменту зі зсувом 80h від його початку і займає 128 байтів. При успішному завершені пошуку функція 4Eh (і 4Fh також) розміщує блок даних, що має формат, як це вказано у табл.13.

Після аналізу даної області в програмі приймається рішення щодо завершення або продовження пошуку.

Для того, щоб знайти наступний файл, що відповідає шаблону, використовується функція 4Fh.

На вході:

AH = 4Fh;

В області DTA повинен міститися блок даних, заповнений єдиним викликом функції 4Eh на початку пошуку.

На виході:

якщо прапорець CF = 0, тоді операцію виконано успішно;

якщо прапорець CF = 1, тоді AX містить код помилки:

AX = 12h – більше файлів у каталозі немає.

Таблиця 13 – Формат і вміст області DTA

Зсув	Розмір в байтах	Опис
00h	1	Буква логічного диску; якщо 7-й біт дорівнює 0, то це віддалений диск
01h	11	Шаблон пошуку
0Ch	1	Атрибути пошуку
0Dh	2	Порядковий номер файла у каталозі
0Eh	2	Номер кластера початку каталога попереднього рівня
11h	4	Резерв
15h	1	Атрибути знайденого файла
16h	2	Час створення (модифікації) файла
18h	2	Дата створення файла
1Ah	4	Розмір файла
1Eh	13	ASCII-ім'я файла з розширенням

Варто зауважити, що для виконання робіт, пов'язаних з файлами MS DOS надає користувачам можливість встановлювати свою область DTA (функція 2Fh) і встановлювати поточну область DTA (функція 1Ah). При цьому всі зміщення і дані, що формуються функціями 4Eh та 4Fh, залишаються актуальними.

2.10.16 Робота з файлами в MS DOS у випадку довгих імен

Перераховані вище функції працюють в різних версіях «чистої» системи MS DOS (включно до версії 6.22). Операційні системи Windows 95/98/Mil також підтримують свою версію MS DOS, яка має номер 7.0. Вони організують спеціальне середовище для роботи, що називається сеансом DOS, і має у своєму складі засоби для роботи з файловою системою Windows. Ця файлова система, як відомо, відрізняється тим, що повне ім'я файла може досягати 255 символів. розглянемо деякі засоби середовища MS DOS для роботи з файловою системою Windows.

Для визначення факту того, в якій системі працює програма, можна за результатами роботи таких функцій:

- функції 30h переривання 21h (отримати версію DOS);
- функції 4A33h переривання 2Fh (визначення факту роботи в середовищі MS DOS 7.0);
- функції 71A0h переривання 21h (встановлення факту того, що система підтримує довгі імена).

У Windows з'явилися додаткові можливості як самої файлової системи, так і засобів щодо керування нею. Основне нововведення – підтримка довгих імен. Будь-який файл у цій системі має два імені – довге ім'я та його псевдонім.

Довге ім'я являє собою ASCII-рядок довжиною до 255 символів. Система формує псевдонім для цього імені форматом 8.3 у відповідності до таких правил: беруться перші 6 символів довгого імені, після них додається символ тильда ('~'), за якою ставиться деякий порядковий номер. Для першого формата 8.3 це 1. Якщо такий псевдонім вже існує, то порядковий номер чергового псевдоніма буде на 1 більшим. Розширення псевдоніма формується з перших трьох символів розширення довгого імені (якщо воно існує).

Розглянуті в попередніх підрозділах функції для роботи з файлами та каталогами не підтримують довгих імен. Для цього система Windows надає програмам MS DOS аналогічні функції, які мають інші номери. Уважно розглянувши більшість з цих номерів, видно, яким зі старих функцій вони відповідають. Нові номери складаються з 4-х цифр: перші дві – 071h, останні дві – номер старої функції. Для деяких з цих функцій існують особливості у їх роботі.

У таблиці 14 наведено перелік функцій переривання 21h, що працюють з файлами, які мають довгі імена.

При цьому варто розрізняти способи використання довгих імен у програмах MS DOS та у програмах Windows. Програми MS DOS отримують доступ до довгих імен файлів через додаткові функції переривання 21h. Програми Windows використовують для цього відповідні функції API.

Порядок виконання роботи

1. Вивчити теоретичну частину даної лабораторної роботи.
2. Скласти програму створення послідовного файла або файла прямого доступу і записати в нього дані.
3. Скласти програму зчитування окремих частин файла.
4. Скласти програму введення-виведення даних із використанням стандартних пристроїв введення-виведення.

Таблиця 14 – Додаткові функції переривання 21h для роботи з файлами. що мають довгі імена

Нова функція int 21h	Стара функція int 21h	Призначення функції	Функція API Win32
5704h		Отримати дату і час останнього доступу	GetFileTime
5705h		Встановити дату і час останнього доступу	SetFileTime
5706h		Отримати дату і час створення	GetFileTime
5707h		Встановити дату і час створення	SetFileTime
7139h	39h	Створити каталог	CreateDirectory
713Ah	3Ah	Знищити каталог	RemoveDirectory
713Bh	3Bh	Змінити поточний каталог	SetCurrentDirectory
7141h	41h	Знищити файл	DeleteFile
7143h	43h	Отримати або встановити атрибути файла	GetFileAttributes SetFileAttributes
7147h	47h	Отримати поточний каталог	GetCurrentDirectory
714Eh	4Eh	Знайти перший файл	FindFirstFile
714Fh	4Fh	Знайти наступний файл	FindNextFile
7156h	56h	Перейменувати файл	MoveFile
7160h	60h	Отримати повний шлях	GetFullPathName
7160h		Отримати повний шлях з короткими іменами	GetShortPathName
7160h		Отримати повний шлях з довгими іменами	Відсутній
716Ch	3Ch 3Dh 5Bh	Створити або відкрити файл	CreateFile OpenFile
71A0h		Отримати інформацію про том	GetVolumeInformation
71A1h		Завершити пошук	FindClose

ЛАБОРАТОРНА РОБОТА № 11
Керування пристроєм "миша"

Тема: Знайомство з можливостями для керування пристроєм "миша". Керування роботою миші у графічному та текстовому режимах.

Теоретичні відомості

2.11.1 Ініціалізація та визначення поточного стану драйвера пристрою "миші"

Для керування пристроєм "миша" використовуються різноманітні функції переривання 33H. Більшість сучасних драйверів "миші" підтримують стандарт, введений фірмою Microsoft для різних функцій переривання 33H. Більшість систем програмування мають у своєму складі бібліотечні функції. що виконують звернення до будь-якої функції переривання 33H. Бібліотечні функції працюють повільніше, ніж безпосереднє звернення до переривання 33H з прикладної програми. Окрім того, програмісти часто використовують неповні або ж навіть несанкціоновані копії систем програмування, в яких бібліотеки підтримки пристрою відсутні або працюють некоректно. Тому надалі наводяться приклади безпосереднього звернення до переривання 33H, а не бібліотечні функції.

Кожна програма, що використовує для введення інформації пристрій "миша", повинна виконати ряд підготовчих дій, а саме:

1) визначити чи інстальований драйвер "миші";
2) задати вид і форму курсора пристрою;
3) описати межі переміщення курсора "миші" по екрана;
4) описати "чутливість" курсора, рівну числу "міккі", що приходиться на один піксел екрана по горизонталі та по вертикалі;
5) встановити межу "подвоєної" швидкості курсора "миші";
6) "включити" курсор пристрою (зробити його видимим на екрані);
7) встановити курсор в початкову позицію на екрані відповідно до потреб програми ініціалізації.

Нижче наводиться специфікація цієї функції.

На вході: AX = 0000h.

На виході: AX = стан пристрою "миші" (0000h – драйвер "миші" відсутній, тобто відсутня апаратура або не завантажено драйвер; FFFFh – "миша" готова до роботи).
BX = кількість кнопок пристрою (0000h – не дві кнопки;
0002h – дві кнопки, режим Microsoft mode;
0003h – три кнопки, режим Mouse Systems mode).

При виконанні функції AX=0000h драйвер пристрою приводиться в стан за замовчуванням:

- курсор пристрою встановлений в центрі екрана і виключений;
- чутливість пристрою по вертикалі становить 2 міккі/піксел, по горизонталі – 1 міккі/піксел;
- межа подвоєної швидкості встановлена рівною 64 міккі/с;
- встановлено форму курсора за замовчуванням;
- координати лівого верхнього кута області переміщення курсора пристрою відповідає координатам (0,0), а координати нижнього правого кута переміщення пристрою – максимальним координатам поточного відеорежиму мінус 1.

Якщо необхідно встановити значення, що відрізняються від значень за замовчуванням, використовуються функції установлення форми курсора, межі його переміщення, установлення значень чутливості і межі подвоєної швидкості.

При визначенні дієздатності "миші", за допомогою функції AX=0000h варто прийняти до відома такі зауваження:

- операційна система MS DOS "чіпляє" при завантаженні за вектор переривання 33h "заглушку" з єдиної інструкції IRET. Тому виконання інструкції INT 33h навіть при відсутності драйвера не породжує ніяких проблем;
- успішне завершення ініціалізації не означає, що апаратура адаптера або самого пристрою справна, а свідчить лише про те що драйвер "миші" завантажено. У ряді випадків драйвер "миші" передає в пристрій заздалегідь задану послідовність слів інформації. Ці слова викликають в пристрої генерацію відповідного відлуння (рос. "эха"), аналіз якого дозволяє драйверу одночасно розпізнати "свою" апаратуру пристрою і перевірити справність шляхів прийому інформації адаптера асинхронного послідовного зв'язку.

2.11.2 Визначення типу і форми курсора "миші" у текстовому режимі

Драйвери "миші" в текстовому режимі роботи відеоадаптера підтримують два види курсора:

1) "жорсткий" курсор, що збігається зі звичайним курсором текстового режиму у формі декількох телевізійних рядків в межах знакомісця;
2) курсор, що програмується і являє собою знакомісце зі зміненим атрибутом символу і, можливо, з символом, що специфікований користувачем.

Курсор "миші" в текстовому режимі переміщується по знакомісцях екрана. Вибір типу і параметрів курсора в текстовому режимі роботи відеоадаптера виконається за допомогою функції AX=000Ah переривання 33h, специфікація якої наведена нижче.

На вході:

AX = 000Ah – установлення курсора "миші" в текстовому режимі;

BX – вибір типу курсора (00 – курсор, що програмується;
01 – "жорсткий" курсор).

CX = AND-маска (маска екрана) для курсора, що програмується, або номер верхньої скен-лінії для "жорсткого" курсора.

DX = XOR-маска (маска курсора) для курсора, що програмується, або номер нижньої скен-лінії для "жорсткого" курсора.

На виході: функція нічого не повертає.

Якщо вибрано "жорсткий" курсор, то курсор "миші" на екрані має форму звичайного текстового. Перевагою такого курсора є те, що на екрані присутній лише один звичний курсор і при виконанні будь-якої операції введення інформації з використанням функцій MS DOS (наприклад, функцій стандартного введення-виведення) курсор "миші" автоматично переміщується в поточну текстову позицію BIOSa. Це не означає, що драйвер "миші" при переміщення курсора оновлює слово поточної позиції курсора в області даних BIOSa. Тому без додаткових зусиль з боку програміста не вдається розмістити символ на екрані в позицію, на яку вказує курсор "миші".

При формуванні текстового курсора, що програмується, використовуються:

а) слово відеопам'яті, утворене символом та атрибутом знакомісця екрана, в якому знаходиться курсор пристрою (screen_word);

б) AND-маска (AND_mask); в) XOR-маска (XOR_mask). Результуюче представлення курсора на екрані формується порозрядними логічними операціями за формулою:

screen_word = screen_word AND AND_mask XOR XOR_mask .

При переміщенні курсора в іншу позицію попередній вміст відеобуфера відновлюється. Таким чином, користувач може встановлювати різні форми програмованого текстового курсора "миші". Наприклад, якщо задати AND_mask=00FFh та XOR_mask=xx00h, то текстовий курсор буде у вигляді прямокутника, що зберігає будь-який попередній символ, атрибут якого буде xx. Молодша шістнадцяткова цифра буде задавати колір контуру символу, а старша – колір фону. Але при цьому можливе "зникнення" курсора, коли атрибут символу вже був рівний xx.

Розповсюдженим є програмований текстовий курсор із такими значеннями: AND_mask = FFFFh та XOR_mask = 7700h. У цьому випадку інвертується і колір фону, і колір атрибута, що гарантує видимість курсора і не змінює символ в позиції курсора.

Неважко добитися того, щоб курсор "миші" на екрані відображався у вигляді потрібного символу. Наприклад, для того, щоб текстовий курсор набув форму "галочки" (ASCII-код символу FBh), необхідно задати такі значення масок: AND_mask = 0000h та XOR_mask = xxFBh. де xx задає атрибут символу курсора. Загальновживаною є практика зміни форми курсора для індикації натискання тієї або іншої кнопки пристрою "миші".

2.11.3 Визначення типу і форми курсора "миші" у графічному режимі

У графічному режимі роботи відеоадаптера може бути описана будь-яка власна форма курсора в межах прямокутника 16×16 пікселів. Курсор у графічному режимі описується двома масками розміром 16×16 бітів кожна. Одна з масок називається AND-маскою (маскою екрана), інша - XOR-маскою (масою курсора). При переміщенні курсора попередній вміст екрана відновлюється драйвером "миші". Маски, комбінуючись, визначають метод обробки поточного коду кольору піксела на екрані. Результати обробки представлені в таблиці 15.

Таблиця 15 – Формування коду кольору пікселів відображення курсора "миші" у графічному режимі

Біт AND-маски	Біт XOR-маски	Колір піксела на ерані
0	0	Колір фону
0	1	Білий колір
1	0	Поточний колір піксела
1	1	Побітова інверсія поточного кольору піксела

За допомогою різних масок можна встановити будь-яку форму курсора і добитися його видимості на будь-якому фоні. Для режимів 4, 5 та 13h використовуються тільки парні стовпчики в бітових масках: в іншому випадку курсор при використанні матриці 16×16 мав би занадто великі розміри. Для інших режимів "проріджування" стовпчиків масок курсора не виконується.

У графічних режимах, окрім форми курсора, описується так звана "гаряча пляма" (hot spot) у відносних координатах, за точку відліку яких прийнято верхній лівий кут прямокутника 16×16 пікселів. "Гаряча пляма" – це той піксел, на який вказує в даний момент графічний курсор. Опис курсора "миші" у графічному режимі виконує функція AX=0009h переривання 33h. Наведемо специфікацію цієї функції.

На вході:

AX = 0009h – установлення курсора "миші" в графічному режимі.

BX = номер стовпчика "гарячої плями" курсора відносно верхнього лівого кута прямокутника курсора.

CX = номер рядка "гарячої плями" курсора відносно верхнього лівого кута прямокутника курсора.

ES:DX = покажчик на 32 слова масок: перші 16 слів утворюють AND-маску, наступні 16 слів – XOR-маску. Використання бітів цих масок ілюструє табл..14.

На виході: функція нічого не повертає.

Для отримання графічного курсора, який би було видно на будь-якому фоні, поступають таким чином. Маску екрана (AND-маску) задають такою, що складається зі слів FFFFh, а маску курсора (XOR-маску) описують так, щоб у ній стояли одиниці у тих бітах, які утворюють обриси курсора, і нулі – в усіх інших бітах маски. Як приклад наведемо створення курсора у вигляді нахиленої стрілки. Формування бітової карти цього курсора пояснюється на рис.22.

Відразу після ініціалізації екрана областю допустимих переміщень курсора за замовчуванням приймається весь екран. Але завжди є можливість обмежити переміщення курсора окремо по горизонталі та по вертикалі, для цього програма використовує функції AX=0007h та AX=0008h переривання 33h.

Бітова карта курсора	Двійкове подання	Шістнадцяткове подання
	1000 0000 0000 0000	0x8000
	1100 0000 0000 0000	0xC000
	1010 0000 0000 0000	0xA000
	1001 0000 0000 0000	0x9000
	1010 1000 0000 0000	0xA800
	1011 0100 0000 0000	0xB400
	1011 1010 0000 0000	0xBA00
	1011 1101 0000 0000	0xBD00
	1011 1110 1000 0000	0xBE80
	1011 1111 0100 0000	0xBF40
	1011 1111 1010 0000	0xBFA0
	1011 1111 1101 0000	0xBFD0
	1011 1111 1110 0000	0xBFE0
	1010 1110 0000 0000	0xAE00
	1010 0011 0000 0000	0xA300
	1110 0011 0000 0000	0xE300

Рисунок 12 – XOR-маска зображення курсора у вигляді стрілки

На вході:

AX=0007h – установлення вертикальних границь переміщення (обмежують горизонтальне переміщення курсора "миші" по екрану);

AX=0008h - установлення горизонтальних границь переміщення (обмежують вертикальне переміщення курсора "миші" по екрану);

CX = мінімальна границя переміщення курсора "миші";

DX = максимальна границя переміщення курсора "миші".

На виході: функція нічого не повертає.

При роботі у графічному режимі границі переміщення задаються у пікселах. Для текстового режиму вони задаються у віртуальних пікселах (виходять з того, що знакомісце незалежно від режиму і типу адаптера займає матрицю 8×8 умовних пікселів). Тому для текстового режиму 25×80 максимально можливими границями переміщення по вертикалі (функція AX=0008h) є значення CX=0 і DX=199, а по горизонталі (функція AX=0007h) – CX=0 і DX=639. Для текстового режиму 50×80 (VGA- та MCGA-адаптери) граничні значення такі: по вертикалі (функція AX=0008h) є значення CX=0 і DX=399, по горизонталі (функція AX=0007h) – CX=0 і DX= 639. У разі, коли мінімальна границя більша за максимальну, функції міняють місцями значення у регістрах CX та DX. Якщо в момент установлення нової границі курсор пристрою знаходиться поза межами заданого діапазону переміщення, він "стрибає" до найближчої з границь.

2.11.4 Установлення чутливості курсора "миші"

При переміщенні пристрою "миша" по столу пропорційно руху переміщається і курсор "миші". Регулювати швидкість переміщення курсора по екрану дозволяють пороги чутливості, що визначають число "міккі", яке необхідно прийняти драйверу для переміщення курсора на один піксел по горизонталі або по вертикалі. Менше значення чутливості робить курсор "миші" більш рухливим. Наприклад, при переміщенні курсора по вертикалі вниз з верхньої лінії до нижньої у графічному режимі 10h (350 рядків) і при чутливості 1 міккі/піксел, для стандартної "миші" 200 міккі/дюйм необхідно перемістити пристрій на 350/200×25 = 44 мм. Для переміщення курсора зліва направо при тих самих параметрах необхідно переміщення "миші" по столу на 640/200×25 = 80 мм. Для установлення чутливості драйвера використовується функція AX=000Fh, специфікація якої наведена нижче.

На вході:

AX = 000Fh – установлення чутливості драйвера "миші" по горизонталі та по вертикалі;

CX – чутливість драйвера "миші" по горизонталі в міккі на 8 пікселів;

DX – чутливість драйвера "миші" по вертикалі в міккі на 8 пікселів.

На виході: функція нічого не повертає.

Мінімальне можливе значення для CX та DX дорівнює 1, що відповідає просто "реактивній" швидкості. Навіть випадкові рухи на столі будуть приводити до переміщення курсора "миші". Висока чутливість драйвера спрощує переміщення курсора на великі "відстані" по екрана, але одночасно утруднює установлення курсора точно в задану позицію на екрані. Наведене протиріччя між швидкістю і точністю позиціонування реалізується драйвером "миші" за допомогою алгоритму балістичного курсора. Ідея цього алгоритму полягає у наступному. Якщо "миша" переміщується по столу з високою швидкістю, то інформаційні слова в адаптер поступають з високою частотою. При досягненні "мишею" так званого порога подвоєної швидкості драйвер починає подвоювати кожний прийнятий сигнал при переміщенні. В результаті вдвічі зростає швидкість переміщення курсора пристрою на екрані. При зменшенні швидкості переміщення пристрою нижче згаданого порогу, подвоєння припиняється і курсор по екрану пересувається зі швидкістю. що визначається параметрами чутливості. Іншими словами, для "далекого" переміщення курсора по екрану треба здійснити "мишею" по столу різкий рух у потрібному напрямку, а потім повільно підвести курсор у потрібну точку. Установлення власного значення порогу подвоєної швидкості виконує функція AX=0013h.

На вході:

AX=0013h – установлення значення порогу подвоєної швидкості

DX – поріг подвоєної швидкості курсора "миші", міккі/с.

На виході: функція нічого не повертає.

(за замовчуванням поріг подвоєної швидкості курсора дорівнює 64 міккі/с);

Для визначення поточних параметрів чутливості і порогу подвоєної швидкості курсора "миші" використовується функція AX=001Bh.

На вході:

AX = 001Bh – визначення поточних значень чутливості по горизонталі та по вертикалі;

На виході:

BX – чутливість драйвера по горизонталі (міккі на один піксел);

CX – чутливість драйвера по вертикалі (міккі на один піксел);

DX – поріг подвоєної швидкості курсора "миші", тобто кількість міккі в секунду, при досягненні якого швидкість переміщення курсора по екрану подвоюється (за за-

мовчуванням поріг подвоєної швидкості курсора дорівнює 64 міккі/с).

2.11.5 Керування станом курсора "миші" (видимий та невидимий курсор)

Після проведеної ініціалізації програма повинна "включити" курсор, що робить його видимим на екрані. Ця операція деблокує секцію обробника переривань від адаптера, яка працює з відеопам'яттю. Відразу після ініціалізації курсор виключений, стан курсора відслідковує спеціальна внутрішня змінна, що встановлюється після ініціалізації "миші" в –1. Керування внутрішнім прапорцем здійснюють дві функції переривання 33h:

- AX=0001h, що робить курсор видимим, тобто змінює на 1 прапорець видимості;
- AX=0002h, яка робить курсор невидимим, тобто зменшує на 1 прапорець видимості.

Якщо прапорець уже дорівнює 0, його подальше збільшення функцією AX=0001h не виконується. Але функція AX=0002h зменшує прапорець, навіть якщо він уже рівний –1. У зв'язку з цим відразу після ініціалізації виконується функція включення курсора і на кожну наступну операцію включення повинна приходитися рівно одна (і ніяк не більше) операція виключення. В протилежному випадку курсор "миші" не буде з'являтися на екрані після його включення.

Всі зміни інформації на екрані слід виконувати з виключеним курсором для запобігання непередбачених ускладнень. справа у тому. що зміни, проведені точно під курсором "миші", не будуть "відомі" драйверу курсора і при переміщенні у нову позицію екрана драйвер відновить не нове, а попереднє значення відеопам'яті. Після виконання змін на екрані слід знову включити курсор. Виключення курсора скидає як форму, так і установлення "гарячої плями" графічного курсора.

2.11.6 Читання позиції курсора і стану кнопок "миші"

Після того, як виконано всі необхідні підготовчі операції, програма може використовувати пристрій "миші" для введення інформації. Інформація, що вводиться, – це поточна позиція курсора і стан кнопок пристрою. Періодичне введення інформації з "миші" і попадання курсора в ті або інші області екрана дозволяє виконувати потрібні дії: відкривати підменю, виконувати заливку якимось кольором та багато чого іншого. Визначення стану кнопки (натиснута, відпущена, натиснута двічі) використовується як команда на виконання програмою додаткових дій. В наш час склався певний стандарт використання кнопок "миші".

Наприклад, натискання лівої кнопки (для "правші") означає вибір того чи іншого пункту меню або включення якогось режиму. У багатьох прикладних програмах натискання лівої кнопки аналогічне натисканню клавіші ENTER. Швидке дворазове натискання лівої або правої кнопки пристрою у деяких випадках використовується як додаткова команда на виконання будь-яких дій програми, наприклад, запуск будь-якої програми на виконання. Права кнопка, навпаки, відміняє зроблений раніше вибір, тобто часто служить аналогом клавіші клавіатури ESC.

Для визначення поточного стану пристрою використовуються функції AX=0003h; 0005h; 0006h; 000Bh.

На вході:

AX = 0003h – визначення місцеположення курсора і стану кнопок пристрою.

На виході:

BL = байт стану кнопок пристрою:

біт 0: 1 – натиснуто ліву кнопку, 0 – не натиснуто ліву кнопку);

біт 1: 1 – натиснуто праву кнопку, 0 – не натиснуто праву кнопку);

біт 2: 1 – натиснуто середню кнопку (для Mouse System), 0 – не натиснуто;

біти 3-7 – не використовуються;

CX = горизонтальна координата курсора "миші" (номер стовпчика на екрані);

DX = вертикальна координата курсора (номер рядка на екрані).

У графічному режимі повертаються піксельні координати, а у текстовому – віртуальні піксельні координати (див. опис функцій AX=0007h та AX=0008h), і для отримання номерів текстового рядка і стовпчика слід розділити значення, які повертаються у регістрах CX та DX, на число 8.

На вході:

AX = 0005h – визначення кількості натискань кнопок "миші";

AX = 0006h – визначення кількості відпускань кнопок "миші";

BX – ідентифікатор кнопки пристрою:

= 0 – запит про ліву кнопку;

= 1 – запит про праву кнопку;

= 2 – запит про середню кнопку (для Mouse Systems).

На виході:

BX = кількість натискань (відпускань) кнопок пристрою з моменту останнього звернення до функцій або з моменту ініціалізації "миші", якщо запит виконується вперше;

CX = горизонтальна координата курсора "миші" (номер стовпчика на екрані) в момент натискання вказаної кнопки;

DX = вертикальна координата курсора (номер рядка на екрані) в момент натискання вказаної кнопки.

У графічному режимі повертаються піксельні координати, а у текстовому – віртуальні піксельні координати (див. опис функцій AX=0007h та AX=0008h), і для отримання номерів текстового рядка і стовпчика необхідно розділити значення, які повертаються у регістрах CX та DX, на число 8. Лічильник кількості натискань (відпускань) зберігає значення від 0 по EFFFh. Після виклику функції AX=0005h або AX=0006h відповідний лічильник занулюється.

На вході:

AX = 000Bh – визначення значення лічильників сигналів міккі.

На виході:

CX = переміщення курсора миші по горизонталі в міккі з моменту останнього виклику даної функції або з моменту ініціалізації "миші", якщо запит виконується вперше;

DX = переміщення курсора миші по вертикалі в міккі з моменту останнього виклику даної функції або з моменту ініціалізації "миші", якщо запит виконується вперше.

Значення, що містять регістри CX та DX на виході, мають цілий тип. Від'ємні значення відповідають руху курсора вліво або донизу, додатні – вправо або вгору. Після кожного виклику функції значення лічильників занулюються. Для перерахунку значень лічильників у пікселі графічного режиму або у віртуальні пікселі текстового режиму використовується функція AX=001Bh, яка визначає поточне значення чутливості по горизонталі та по вертикалі.

2.11.7 Позиціонування курсора миші пристроєм і прикладною програмою

При переміщенні пристрою по столу драйвер переміщає курсор "миші" по екрана без будь-якої участі програми. Однак і прикладна програма має можливість керувати позицією курсора "миші", для цього використовується функція AX=0004h.

На вході:

AX = 0004h – установлення курсора "миші" і внутрішніх лічильників позиції в нове місце на екрані;

CX = горизонтальна координата курсора "миші";

DX = вертикальна координата курсора "миші".

На виході: функція нічого не повертає.

Нові координати задаються для графічного режиму в пікселах, а для текстового – у віртуальних пікселах. Тому при необхідності установлення пристрою в потрібні текстовий рядок або стовпчик їх номери множаться на

число 8. Дана функція не перетинає границі, встановлені функціями AX=0007h та AX=0008h. При запиті позиціонування на точку екрана поза діапазонами переміщення курсора останній "натикається" на найближчу границю, але не перетинає її.

Наявність функції керування позицією курсора "миші" дозволяє організувати керування єдиним курсором на екрані як самим пристроєм, так і з клавіатури. Така можливість керування рухами курсора часто використовується при побудові різноманітних текстових редакторів. Функція прийому клавіатурного введення постійно аналізує поточну позицію курсора "миші" і приводить у відповідність до неї позицію текстового курсора BIOSa. При натисненні клавіш клавіатури зі стрілками (Left, Right, Up, Down і ін.) виконується позиціонування курсора "миші". Розглянуті дії приводять до того, що поточна позиція курсора, відома BIOSy для виведення інформації на екран, керується як пристроєм, так і клавішами зі стрілками.

2.11.8 Приклад програми для керування пристроєм "миша" у текстовому режимі роботи відеоадаптера

Наведена нижче програма демонструє принципи програмування пристрою "миша" з використанням функцій переривання INT 33h.

При одночасному виконанні двох умов – попадання текстового курсора "миші" у прямокутник екрана дисплея з координатами лівого верхнього кута X_1=6, Y_1=1, координатами правого нижнього кута X_2=25, Y_2=8, та натисненні лівої клавіші "миші" – на екран дисплея виводиться рядок символів '*'.

```
NAME    MOUSE5
        .MODEL  TINY
CODE    SEGMENT
        ASSUME  CS:CODE
        ORG     100H
BEGIN:  JMP     MAIN
PRESS   DW      ?
X       DW      ?
Y       DW      ?
MAIN    PROC
        MOV     AX,0
        INT     33H
        MOV     AX,1
        INT     33H
        MOV     PRESS,0
M1:     MOV     AX,3
        INT     33H
```

```
            MOV     PRESS,BX
            MOV     X,CX
            MOV     Y,DX
            CMP     X,48
            JL      M1
            CMP     X,200
            JG      M1
            CMP     Y,8
            JL      M1
            CMP     Y,200
            JG      M1
            CMP     PRESS,1
            JNE     M1
            CALL    OUTPUT
            RET
MAIN        ENDP
OUTPUT      PROC
            MOV     AH,2
            MOV     DH,10
            MOV     DL,5
            MOV     BH,0
            INT     10H
            MOV     AH,9
            MOV     AL,'*'
            MOV     CX,40
            MOV     BL,0E1H
            INT     10H
            RET
OUTPUT      ENDP
CODE        ENDS
            END     BEGIN
```

Порядок виконання роботи

1. Вивчити теоретичну частину даної лабораторної роботи.
2. Скласти програму для керування пристроєм "миша" в текстовому режимі роботи відеоадаптера з використанням функцій переривання int 33h.
3. Скласти програму для керування пристроєм "миша" в графічному режимі роботи відеоадаптера з використанням функцій переривання int 33h.

ЛІТЕРАТУРА

1. Григорьев В. Л. Программирование однокристальных микропроцессоров. – М.: Энергоатомиздат, 1987.

2. Использование Turbo Assembler при разработке программ. – Киев, "Диалектика", 1994. – 288 с.

3. Абель П. Язык Ассемблера для IBM PC и программирования/ Пер. с англ. – М.: Высш. школа, 1992 – 447 с.

4. Финогенов К.Г. Основы языка ассемблера. – М.: Радио и связь, 1999.-288 с.

5. Юров В. Assembler: Учебник. – СПб и др.: Питер, 2000. – 622 с.

НАВЧАЛЬНЕ ВИДАННЯ

Василь Петрович Семеренко,
Валентина Аполінаріївна Каплун

Системне програмування мовою Асемблера

Частина 2

Лабораторний практикум

Оригінал-макет підготовлено авторами

Редактор В.О. Дружиніна
Коректор З.В. Поліщук

Навчально-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК №746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 2.03.05р. Гарнітура Times New Roman
Формат 29,7х42 1/4 Папір офсетний
Друк різографічний Ум. друк. арк. 5.08
Тираж 75 прим.
Зам. № 2005- С3С

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК №746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ