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

«МІКРОПРОЦЕСОРНА ТЕХНІКА»

КОМП'ЮТЕРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 171 «Електроніка»,
спеціалізацією «Електронні компоненти і системи»*

Київ
КПІ ім. Ігоря Сікорського
2017

«Мікропроцесорна техніка»: Комп’ютерний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 171 «Електроніка», спеціалізації «Електронні компоненти і системи» / КПІ ім. Ігоря Сікорського ; уклад.: Т. О. Терещенко, О.В. Хоменко – Електронні текстові данні (1 файл: 331 кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2017. – 46 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № від р.)
за поданням Вченої ради факультету електроніки (протокол № 12/2017 від .12.2017 р.)

Електронне мережне навчальне видання

«МІКРОПРОЦЕСОРНА ТЕХНІКА»
КОМП’ЮТЕРНИЙ ПРАКТИКУМ

Укладачі: *Терещенко Тетяна Олександрівна,* докт. техн. наук.
Хоменко Олександр Васильович

Відповідальний редактор: *Ямненко Ю. С.,* завідувач кафедри промислової електроніки, д-р техн. наук, проф.

Рецензенти: *Михайлов С.Р.,* доцент кафедри електронних приладів та пристроїв, канд. техн. наук, доц.

Навчальний посібник «Мікропроцесорна техніка: комп’ютерний практикум» сприяє набуттю практичних навичок створення програмного забезпечення мікропроцесорних систем та методів їх налагодження.

ЗМІСТ

ВСТУП

Метою виконання циклу лабораторних робіт є:

- закріплення і експериментальна перевірка теоретичних положень найважливіших розділів і тем навчального матеріалу;
- оволодіння інтегрованими програмними комплексами відпрацювання прикладного програмного забезпечення, що спрощують цей процес;

Під час виконання робіт студенти відпрацьовують основні етапи створення та налагодження програмного забезпечення мікропроцесоррних пристроїв.

Лабораторна робота 1

1. Основи програмування на мові Асемблер. Дослідження програмної моделі мікропроцесора i8086

1.1. Основи програмування на мові Асемблер

Мета роботи: Знайомство з програмним забезпеченням та основами програмування на мові Асемблер

Порядок виконання роботи

1. Знайти на системному диску C: компьютера у директорії 1810 файли, *tasm.com, afd.com.*
2. Загрузити *tasm.com.* Набрати та проаналізувати текст програми заданої викладачем, зрозуміти зміст кожної з асемблерних команд.
3. Сформувати об'єктний код програми *lab1.asm* та записати його до файлу *lab1.com* за допомогою програми TASM.
4. Вийти з *tasm.com*, при цьому не забудьте проконтролювати запис текстового файлу.
5. Загрузити *afd.com*, а в нього робочу відкомпельйовану программу.
6. Завантажити відладчика AFD, а нього програму *lab1.com* та крок за кроком виконати всі її команди. Після виконання кожної команди аналізувати стан кожного з регістрів процесору та комірками пам'яті.

Наприклад, текст програми може мати такий вигляд:

```
        org   100h

m1:     mov   AX,DS:[SI]
        cmp    AL,AH
        jz v1
        mov   CX,AX
v1:     mov   DS:[SI+1], ax
        end
```

Зміст звіту

1. Програма з коментарями
2. Висновки

Контрольні питання:

1. Що таке початкова та об'єктна програма?
2. Програмова модель мікропроцесора К1810ВМ86, призначення всіх його регістрів
3. Основні регістри процесору типа Intel 8086, їх призначення
4. Сегментація програми, сегменти коду, даних та стеку, їх розташування в пам'яті ЕОМ. Директиви мови Асемблеру
5. Організація стекової пам'яті та її призначення

Теоретичні відомості

Основи роботи з TASM

Програма транслятора з мови асемблера мікропроцесора К1810 TASM (*Turbo-Assembler)* призначена для написання та часткового відлагодження програм, які написані на мові асемблера. Для початку роботи з програмою потрібно запустити командній файл tasm.com. На екрані монітора повинно з'явитися головне меню системи TASM, яке має наступний вигляд:

Assem Source	Edit Source	Get Source	Write Source
Run Codefile	Hexdump File	Kill File	List File
Symbol List	Xrefer List	Directory	New Drive-Directory
Asm ptions	Value	Quit	

Виклик тієї чи іншої функції меню проводиться через натиснення клавіши з відповідною виділеною літерою (нище вони приведені у дужках).

Команда	*Призначення*
A**ssemble Source **(A)	Асемблювання введеної програми;
E**dit Source **(E)	Редагування тексту програми;
G**et Source **(G)	Занесення файла з початковим текстом програми до пам'яті ;
W**rite Source **(W)	Запис остаточного текста програми у файл;
R**un Codefile **(R)	Виконання команд програми після її трансляції;
H**exdump File **(H)	Проглядання команд програми у кодах шістнадцятирічної системи;
K**ill File **(K)	Знищення файлу з текстом програми ;
L**ist File **(L)	Проглядання текста початкової програми;
S**ymbol List **(S)	Проглядання таблиці символьних імен програми;
X**refer List **(X)	Службова інформація програми;
D**irectory **(D)	Перегляд змісту діску або каталогу;
N**ewDrive-Directory **(N)	Зміна діску або каталогу;
A**sm Options **(O)	Параметри асемблювання;
V**alue **(V)	Значення введеного числа. Можливі параметри – шістнадцятерічне, десяткове, вісімкове, двійкове, знак по ASCII;
Q**uit **(Q)	Закінчення роботи з програмою

Для того, щоб почати вводити текст асемблерної програми, потрібно увійти до редактора (команда **Edit** головного меню). Після цього написання програми проводиться звичайним чином з використанням стандартних клавіш керування курсором та редагування.

Для редагування вже існуючого та збереженого на диску файла потрібно перед входом до редактора прочитати його за допомогою функції головного меню

Get. Після закінчення роботи в редакто- рі треба повернутися до головного меню TASM за допомогою клавіши **“ESC”** та провести трансляцію програми. Для цього насамперед треба встановити опції асемблювання за допомогою функції головного меню **Asm Options**. Після виклику цієї функції з’являється меню другого рівня. Та чи інша функція асеблювання програми встановлюється за допомогою функціональних клавіш **F1, F2, ... , F10**. Напроти кожної функції вказується її стан, який можна змінити через натиснення відповідної клавіші:

ON - включено, OFF - виключено.

Клавіша	*Функція асемблювання*	*Що виконує*
F1	Screen	Виведення лістінгу програми на екран
F2	Printer	Виведення лістінгу програми на друк
F3	Symbols	Виведення таблиці символьних імен
F4	Xrefer	Виведення таблиці змісту
F5	Memory	трансляція до пам’яті машини
F6	ErrWait	зупинка трансляції при визначенні помилки
F7	ObjFile	трансляція до об’єктного файлу
F8	ComFile	Трансляція до командного файлу
F9	LSTFile	Трансляція до файлу лістінгу
F10	UnDef	Виведення невизначених імен

Після встановлення потрібних опцій асемблювання можна викликати функцію асемблювання головного меню TASM (**Assemble Source**) та отримати необхідний результат. Для збереження тексту асемблерної програми на диску треба використати функцію меню **Write Source.** При цьому завжди потрібно вказувати ім’я файлу для виведення інформації.

Основи роботи з програмою AFD

Програма **AFD** дозволяє відлагоджувати програми, які написані на мові Асемблеру мікропроцесорів К1810ВМ86/88 і має такі можливості:

1. Занесення до пам’яті виконуємих модулей та їх дезасемблювання.

2. Виконання всієї занесеної програми або її частини.

3. Виконання окремих інструкцій програми у покроковому режимі.

4. Аналізує стан всіх регістрів процесора та комірок пам’яті прямого доступу.

5. Вносить зміни до всіх регістрів процесора та комірок пам’яті доступу.

Головне меню відладчика **AFD** являє собою систему з шести вікон та з командної строки. За допомогою клавіш керування курсором ↑ та ↓ маркер строки переміщується по вікну програми-дізасемблеру, а інші функції виконуються за допомогою функціональніх клавіш F1...F10:

Призначення функціональних клавіш системи AFD

Клавіша	*Функція*	*Що виконує*

Клавіша	*Функція*	*Що виконує*
F1	Step	Виконання однієї інструкції програми, відміченої маркером у вікні дизасемблера
F2	Step Proc	Повне виконання одного рядка програми. Наприклад, якщо це визов подпрограми, то вона буде виконана одразу повністю як одна інструкція
F3	Retrieve	Повторення введенних команд командної строки. Можна повторити до шести команд
F4	Help	Виклик файлу допомоги
F5	Set BRK	Встановлення точок зупинення
F6	----------	Клавішу незадієно
F7	Up	Перенесення маркера догори
F8	Dn	Перенесення маркера донизу
F9	Le	Перенесення маркера ліворуч
F10	Ri	Перенесення маркера праворуч.

Після запуску програми AFD.COM з`являється головне меню системи. Далі користуючись відповідними командами можна відлагодити необхідну програму. Для цього в командній строці вводяться такі команди:

Набір головних команд, які підримуються відладчиком AFD

Команда	*Призначення*
L [F name ,adr] {ім`я файла, адреса }	Ввести файл до пам'яті ЕОМ. Адреса розміщення може бути вказана явно, але це не обов'язково. По змовченню адреса розтошування програми буде CS:0100H. Число прочитаних байтів після завершення програми вказуєтьсся парою реєстрів BX:CX.
W [F name.,adr,repeator] {ім`я файла, адреса, довжина}	Записати дані до файлу. Початкова адреса повинна знаходитися у сегменті даних DS. Довжина файлу повинна бути у межах чотирьох шістнадцятирічних цифр. Всі параметри команди є обов'язковими.
D [adr] {адреса}	Показати коди команд, починаючи з вказаної адреси. Сегмент пам'яті визначається реєстром CS. З цієї ж адреси можна і виконувати програму.
R [reg = значення]	Занести число до реєстру процесора. Наприклад, R A=100H – команда заносить до реєстру A число 100H. Якщо набрати R FL = значення – можна завантажити весь реєстр стану як один шістнадцятирічний реєстр. Але є можливим і окремий доступ до бітів реєстру стану через імена OF, DF, IF, SF, ZF, AF, PF, CF.
M_n [adr] {адреса}	Показати перше або друге вікно стану пам'яті (значення параметру n=1 або n=2), починаючи з вказаної адреси. По змовченню сегмент є той самий, що у вікні. Для непрямої адресації можна вказати якийсь регістр (наприклад [SI]).
G {стартова адреса}, {адреса зупинки}	Виконати програму від стартової адреси до адреси зупинки. Якщо не вказана стартова адреса, то виконання програми почнеться з адреси, на який вона була припинена. Якщо не вказана адреса зупинки, то програма бу-

Команда	*Призначення*
	де виконуватись до першої точки припинення або до кінця. Крім того, можна припинити виконання програми через натиснення клавіш Ctrl+Esc.
QUIT {R}	Повернення до DOS. Опція R робить програму afd реезидентною. В цьому випадку її ініціалізація проводиться через натиснення клавіш Ctrl+Esc.
A [adr] {адреса}	Початок асемблювання. Якщо адреса не вказана, то асемблювання провадиться від існуючої інструкції. Клавішами контроля курсора можна переглядати програму. Виристовується сегмент CS.
P ***{адреса, строка}***	Заповнення пам'яті кількома символами. Виконується в сегменті CS.
F [adr, repeator, string] {адреса, довжина, константа}	Заповнення пам'яті константами. Виконується в сегменті DS. Вказується число повторів констант (довжина).
S [adr, string] {адреса, константа}	Пошук данних у пам'яті. Якщо адреса не вказана, початок від CS:0.Коли дані будуть знайдені, то вони з'являться у вікні M2 з параметромHS. S команда без параметрів повторює пошук знову.
C [adr1, adr2, repeator] {адреса 1, адреса 2,довжина}	Порівняти дві області пам'яті. Якщо області відрізняються, то M1 встановлюється як область першого параметра, а M2, як область другого параметра. Сегмент DS.
CO [adr1, adr2, repeator] {адреса.1, адреса 2, довжина}	Зкопіювати дані з області з початковою адресою 1 до області з початковою адресою 2 заданою довжиною (сегмент DS).
I [adr] {адреса}	Ввести і переглянути дані з порту. Адреса може бути 8-ми або 16-ти бітною, або встановлена в регістрі.
O {адреса, значення}	Вивести данні до порту. Якщо дані слово, операція буде виконана.
T{B}	Переглянути буфер трасування. Якщо 'B' не вказано, місткість буфера трасування буде показано.
BW [F name] {ім'я файла}	Занести точки зупинок до файлу.
PT {старт,довжина, {ім'я файла}}	Вивести на друк місткість буферу транслювання. Старт вказує на початок першої•інструкції, яка буде виведена. Кількість надрукованих інструкцій вказана як довжина чи кількість інструкцій, які виводяться. По замовченню всі.

Головна перевага відладчика **AFD** - його відносна простота та зручність багатовіконного інтерфейсу. Ці переваги особливо відчутні при відлагодженні невеликих **com**-модулей. Недолік цієї програми - неможливість проглядення робочого екрана програми, що виконується.

1.2. Дослідження програмної моделі мікропроцесора

Мета роботи: Вивчити програмну мо- дель мікропроцесора i8086. Навчитися інтерпретувати стан регістрів мікропроцесора з використанням емулятора. Засвоїти правила запису даних

Порядок роботи:

1. Використовуючи середовище емулятора МП i8086, створити і налагодити проект асемблерної програми відповідно до індивідуального завдання
2. Використовуючи середовище емулятора скласти лістинг програми
3. Записати коментарі та пояснити результати

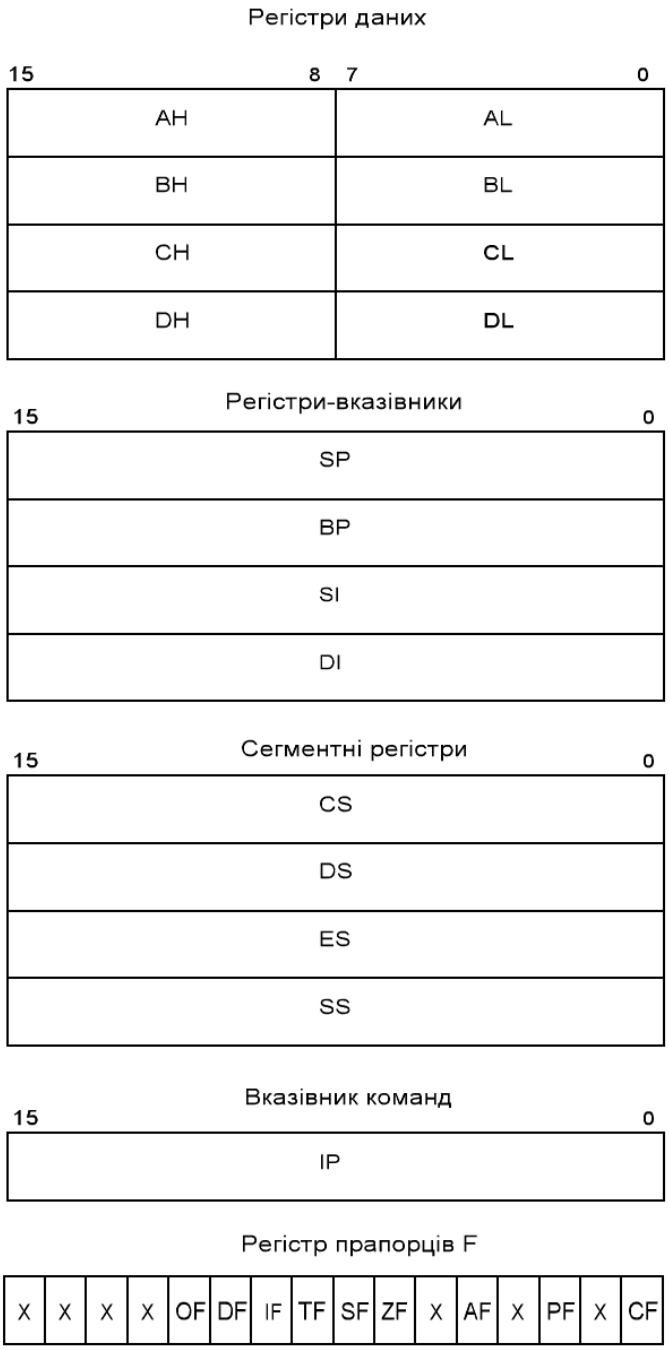
Зміст звіту

1. Програма з коментарями
2. Висновки

Теоретичні відомості

Програмною моделлю МП називається сукупність програмно доступних регістрів, тобто тих регістрів вміст яких можна зчитати або змінити за допомогою команд. Програмну модель МП i8086 складають регістри загального призначення (РЗП), сегментні регістри, вказівник команд і регістр прапорців. Програмна модель МП i8086 представлена на рис. 1.

РЗП поділяються на регістри даних і регістри-вказівники. До регістрів даних відносяться 16-розхрядні регістри: AX, BX, CX, DX. Кожен із цих регістрів складається з двох 8-розрядних регістрів, які можна незалежно адресувати іменами AH, BH, CH, DH (старші байти – Hihg) та AL, BL, CL, DL (молодші байти – Low). Регістри-вказівники SP (Stack pointer – вказівник стеку), BP (Base pointer – базовий регістр), SI (Source Index – індекс джерела), DI (Destination Index – індекс призначення) є 16-розрядними. Усі РЗП можна використати для зберігання даних, але в деяких командах допускається використання певного регістра за замовчуванням: AX – при множенні, діленні, введенні та виведенні слів; AL - при множенні, діленні, введенні та виведенні байтів, десятковій корекції, перетворенні байтів; AH – при множенні і діленні байтів; BX – при трансляції; CX – як лічильник циклів і вказівник довжини рядків у рядкових командах; CL – для зберігання зміщення з указанням змінної; DX – при множенні та діленні слів, при введенні та виведенні з непрямою адресацією; SP – при операціях із стеком; SI, DI – при рядкових операціях. На відміну від 8-розрядних мікропроцесорів регістр SP зберігає зміщення останньої зайнятої комірки стека.



4.

Рисунок 1 - Програмна модель МП i8086

Лабораторна робота №2
Програмування на мові асемблера. Команди передачі даних

Мета роботи: Навчитися складати програму на асемблері для пересилання даних

Порядок роботи:

1. Розглянути набір команд передачі даних та вивчити особливості їх застосування Див.Додаток А.
2. Для закріплення отриманих теоретичних знань створити програму, використовуючи команди передачі даних..
3. Підготуватися до захисту лабораторної роботи за наведеними запитаннями.

Зміст звіту

1. Програма з коментарями
2. Висновки

Контрольні питання:

1. Які команди мови асемблера дають можливість працювати з портами вводу/виводу?
2. Описати можливості використання портів вводу/виводу для програмування таймера.
3. Які припустимі операнди в команді *MOV*?
4. За допомогою якої команди можна здійснити обмін даними між операндами?
5. Як можна зберегти (відновити) вміст регістра прапорів?
6. Як здійснюється завантаження дальніх покажчиків?
7. Як працює команда перетворення даних *XLAT*?

Лабораторна робота №3

Програмування на мові асемблера. Команди обробки даних

Мета роботи: Навчитися складати програму на асемблері для обробки даних

Порядок роботи:

1. Проаналізувати індивідуальне завдання, видане викладачем та написати програму на мові Асемблеру. З'ясувати, як будуть змінюватися дані в усіх регістрах процесору при виконанні кожної з команд програми.
2. Сформувати об'єктний модуль за допомогою програми TASM.
3. Запустити програму на обробку. Переконайтеся у тому, що її коректно виконується (комп'ютер не "підвисає").
4. Завантажити програму до відладчика AFD або TASM та крок за кроком виконати усі її команди. Після виконання кожної з команд проаналізувати стан кожного з регістрів процесору.

Зміст звіту

1. Роздрукований листинг програми на мові Асемблеру
2. Таблиця результатів з трьох колонок. У першу занести початковий код асемблерної програми файлу, в другу - її машинний код, в третю - стан усіх регістрів процесору після її виконання
3. Зафіксувати числове значення усіх комірок пам'яті, у яких розташовано програму, з'ясувати, чому до них були завантажені саме такі числа
4. Зробити висновки за роботою

Контрольні питання:

1. Які ви знаєте арифметичні команди мови Асемблер? Як вони записуються? Які мають операнди? Який результат їх виконання?
2. Поясніть динаміку зміни чисел у регістрах процесора після виконання кожної з команд вашої програми.
3. З якими числами можлива робота ПК на базі процесора *I*8086? З якими числами можлива робота ПК на базі сучасних процесорів?
4. Які арифметичні команди використовуються для додавання операндів?
5. Які арифметичні команди використовуються для віднімання операндів?
6. Які можливості має асемблер для знакових і беззнакових множень та ділень?
7. Яким чином можна робити операції із числами, розрядність яких перевищує розрядність процесора?
8. Число якої розрядності отримаємо в результаті множення двох 32-бітних чисел?
9. Яким чином при арифметичних операціях ураховується переносу старший розряд чи позичання із старшого розряду?

Лабораторна робота № 4
Програмування на мові асемблера. Команди логічних операцій та команди зміщень

Мета роботи: навчитися створювати програми на мові асемблер за допомогою логічних операцій та команди зміщень

Порядок виконання роботи

1. Розглянути логічні команди, команди зміщень, команди переходів.
1. На основі вивченого матеріалу створити кодуючий та декодуючий пристрій, стійкий до спотворення сигналу
2. Проаналізувати індивідуальне завдання та побудувати блок-схему для запропонованого алгоритму.
3. Написати програму на мові Асемблеру, яка реалізує алгоритм згідно з завданням

Зміст звіту

1. Блок-схема алгоритму
2. Роздрукована програма на мові Асемблеру
3. Таблиця результатів
4. Висновки

Контрольні питання:

1. Які логічні команди для роботи з бітами ви знаєте?
2. Навести алгоритм правого та лівого циклічного зміщень.
3. Навести алгоритм арифметичного та логічного зміщень.
4. Чим відрізняється циклічне зміщення від звичайного?
5. Які саме прапори містить регістр прапорів?
6. Які команди можуть змінювати значення прапорів?
7. Які команди умовного переходу ви знаєте?
8. Навести алгоритм кодування за кодом Хемінга.
9. Навести алгоритм декодування за кодом Хемінга.

Лабораторна робота №5
Програмування на мові асемблера. Команди циклів та роботи з масивами. Створення COM-файлів

Мета роботи: Навчитися створювати програми на мові Асемблер за допомогою команд вітвлення та циклу

Порядок виконання роботи

1. Проаналізувати індивідуальне завдання та побудувати блок-схему для запропонованого алгоритму.
2. Написати програму на мові Асемблеру, яка реалізує алгоритм згідно з завданням
3. Сформувати об'єктний код за допомогою програми TASM.
4. Запустити програму на обробку. Переконайтеся у тому, що вона коректно виконується (комп'ютер не "підвисає").
5. Завантажити програму до відладчика AFD або TASM та крок за кроком виконати усі її команди. Після виконання чергової команди аналізуйте стан кожного з регістрів процесору.

Зміст звіту

1. Блок-схема алгоритму
2. Роздрукована програма на мові Асемблеру
3. Таблиця результатів з трьох колонок. До першої занести початковий код асемблерної програми файлу, до другої - її машинний код, до третьої - стан усіх регістрів процесору після її виконання. Для команд циклу фіксувати значення регістрів для кожної з ітерацій. Обов'язково фіксувати значення регістра стану! та числові значення усіх комірок пам'яті, у яких розташовано програму з поясненнями, чому до них завантажено саме такі числа
4. Висновки

Контрольні питання:

1. Яких стандартних помилок припускаються малодосвічені програмісти при реализації циклів та циклічних процесів?
2. Типи адресації, що підтримуються у мові Асемблеру, в яких асемблерних командах вони використовуються?
3. Команда безумовного переходу у мові Асемблеру, її зв'язок з типами адресації.
4. Команда циклу в мові Асемблеру, особливості її реалізації.
5. Команди умовного переходу. На які головні групи вони поділяються та з чим це пов'язано?

Лабораторна робота №6

Дослідження стандартних підпрограм BIOS. Програмування на мові асемблера з включенням стандартних підпрограм для формування зображення на дисплеї ПЕОМ. Створення EXE-файлів.

Мета: Навчитися організовувати exe файл та виводити символи на екран монітора за допомогою переривань.

Порядок роботи

1. Ознайомитись з теоретичними відомостями по перериванням BIOS 10H. та MS-DOS 21H.
2. Написати структуру модуля типу exe
3. Написати програму виводу зображення заданого викладачем на монітор персонального комп'ютеру з використанням переривань

Зміст звіту

1. Програма з коментарями
2. Висновки

Контрольні питання:

1. Порівняйте написання COM І EXE модулів.
2. В чому полягають собливості написання модуля типу exe?
3. Як встановити курсор на задану позицію монітора?
4. Як вивести символ на монітор?
5. Як «погасити» монітор?

Теоретичні відомості.

Особливості написання модуля типу exe:

* Обов'язкове визначення всіх трьох сегментів - коду, стеку та даних.
* Обов'язкова ініціалізація сегментних регістрів через директиву **ASSUME**.
* На початку exe-модуля, що завантажений до пам'яті, стоїть PSP, який їм використовується. Програма автоматичного завантаження MS-DOS використовує регістр DS для встановлення початкової адреси PSP. Тому у програмі користувача треба насамперед зберігти цю адресу, завантаживши її до стеку. Після закінчення програми це значення буде використане командою **RET** для повернення до MS-DOS.
* Для операційної системи потрібно, щоб наступне значення стеку було нульовим. Для цього треба за допомогою команди **SUB** почистити регістр AX та завантажити до стеку нульове значення.
* Програма завантаження встановлює вірні адреси сегментів коду та стека. Але ж регістр сегменту даних DS використовується нею для інших цілей і має після її виконання невірне значення. Тому треба перезавантажити регістр DS за допомогою двох команд **MOV**.

* exe-програма повинна мати модульну структуру. Тому всі частини програми, включаючи головну, реалізуються як процедури.

Таким чином, структура коректно написаної exe-програми така:

```
STACKSG SEGMENT PARA STACK 'Stack'
        DW    32 DUP(?); Визначення стеку
STACKSG ENDS
DATASG SEGMENT PARA 'Data'
.....................................................
DATASG ENDS
CODESG SEGMENT PARA 'Code'
BEGIN  PROC    FAR
     ASSUME CS:CODESG,DS:DATASG,SS:STACKSG,ES:DATASG
                PUSH DS;
                SUB  AX,AX;
                PUSH AX;
                MOV  AX,DATASG ;
                MOV DS,AX;
                .....................................
                RET
        BEGIN     ENDP
        CODESG ENDS
  END BEGIN
```

Реалізація екранних операцій

Розглянемо головні екранні операції мови Асемблер. Їх можна реалізувати трьома засобами:

- **через безпосереднє звертання до відеопам'яті.**
- **через використанням переривання BIOS 10H.**
- **через використанням переривання MS-DOS 21H.**

Встановлення курсора на задану позицію. Як вам відомо, екран монітора у текстовому режимі містить 25 рядків по 80 символів у кожному. Координати встановлення курсора містять номер рядка та столбця, але нумерація починається з нуля. Тобто, можливі горизонтальні координати - 0...24 (0...18H), а вертикальні - 0...79 (0...4FH). Встановити курсор на задану позицію можна через переривання BIOS 10H. Функція переривання визначаються так:

* До регістру AX треба завантажити число 02 (нумер функції переривання), що відповідає цій операції.
* Завантажити до регістру DH номер рядку, а у регістру - DL номер столбця.
* Завантажити до регістру BX номер сторінки екрану (для одноекранних програм обов'язково завантажити число 0).

Наприклад, для встановолення ку- рсора на п’яту позицію дванадцятого рядку треба написати такий фрагмент програми:

```
MOV AH, 02
MOV BH, 00
MOV DH, 05
MOV DL, 12
INT  10H
```

Очищення екрану. При початку роботи програми часто-густо треба виконати функцію очишення екрану. Переривання BIOS 10H дозволяє очищувати будь-яку область прямокутну область екрану, тобто область для очищення може починатися з будь-якої позиції та будь-якою позицію закінчуватись. Початкове значення рядка завантажується до регістру CH, початкове значення столбця - до регістру CL, кінцеве значення рядка - до регістру DH, кінцеве значення столбця - до регістру DL. Для визначення функції переривання треба занести AH число 06H. Треба чітко слідкувати за тим, щоб координати кінцевої позиції перевищували координати початкової. Для придання екрану нормального атрибуту (виведення білих символів на чорному фоні) треба занести в регістр BH число 07H. Наступний фрагмент програми виконує очищення всього екрану:

```
MOV  AX, 0600H
MOV BH, 07
MOV CX, 00
MOV DX, 184FH
INT  10H
```

Встановлення розміру курсору. Як вам відомо, вертикальний розмір курсора при роботі у текстовому режимі MS-DOS може змінюватись від великого значення (майже на всю довжину рядка, перекриваючи літери) до малого (тонка рісочка). Переривання BIOS 10H дозволяє ефективно змінювати розмір курсора. Але нажаль, операції яка б могла встановити однотипну форму курсора для будь-яких типів монітора не існує. Більш того, якщо програміст вирішив “погасити” курсор при виконанні своєї програми, на моніторі з іншими параметрами (наприклад на чорно-білому) він при виконанні цієї ж програми може прийняти максимальний розмір. Проте, для більшості сучасних моніторів максимальна висота курсора становить 13 ліній.

Для встановлення розміру курсора може бути ефективно використана **функція 02H** переривання BIOS 10H. У регістр CL заноситься нижня, а у регістр CH - верхня лінія курсору. Фрагмент програми для придання курсору мінімального розміру (одна лінія) можна записати так:

```
MOV     AH,01
MOV     CH,00
MOV     CL,01
INT     10H
```

Читання поточного положення курсору. Ця функція виконується через переривання BIOS 10H, функція 03H. Після повернення до головної програми в регістрах процесора містяться такі числа:

DH - нумер рядка;

DL - нумер столбця;

CH - верхня лінія курсора;

CL - нижня лінія курсора.

Зрозуміло, що всі дані, які знаходились у цих регістрах до виконання команди INT будуть втрачені.

Виведення сиволів на екран. Всі процедури BIOS та MS-DOS для виведення на екран розміщюють символ, що виводиться на поточній позиції курсора та зміщюють його на одну позицію праворуч. При досягнені 80 позиції курсор переміщується на першу позицію наступного рядка (якщо програмістом не були дани "особливі вказівки" зміщувати в цьому випадку символ на 1 позицію проворуч). Більшість переривань потребують виведення символів потребують задавання його **атрибуту (атрибут - це чисельний опис колярів_символу та його фону)**, а деякі виводять символ з поточним атрибутом.

Але не є складним і безпосереднє виведення даних на екран монітору через звернення до відеопам'яті. Для цього є достатнім розташування ASCII-коду символу та його атрибуту у відповідних комірках. У всіх кольорових моніторах пам'ять відеоадаптера розташована за адресою B8000H. Перша комірка пам'яті містить ASCII-код символу, що виводмться, друга його атрибут. Таким чином, для виведення числа на першу позицію першого рядку треба у комірку пам'ті з адресою B8000H занести його ASCII-код, а у комірку з адресою B8001H його атрибут. ASCII-код символу, що розташований на другій позиції першого рядку, буде міститися за адресою B8002H і так далі.

Атрибут символу формується таким чином: Перші три біти містять коляр тексту, що закодований у форматі RGB (від англійського скорочення складових частин білого коляру: R - червоний, G - зелений, B - синій). Четвертий біт - біт яскравості тексту (0-мала, 1-велика). П'ятий, шостий та сьомий біт кодують коляр текстового фону у форматі RGB, а встановлення в одиницю восьмого біту призводить до блимання символу.

Коди кольорів у форматі RGB такі:

Чорний - 000;
Синій - 001;
Зелений - 010
Блакитний - 011;
Червоний - 100;
Бузковий - 101;
Бурий - 110;
Білий - 111

Більш ефективно виведення символів на екран можна організувати за допомогою переривань BIOS та MS-DOS. **Функція 0AH** переривання BIOS 10H виводить символ на поточну позицію курсора. Номер функції, як завжди, завантажує-

тиься до регістру AH. До регістру AL завантажується ASCII-код символа, а до регістру CX - число повторень при виведенні, але ж воно буде здійснюватися на ту ж саму позицію. Курсор не переміщується.

Функція 09 переривання BIOS 10H ідентична функції 0AH, але забезпечує виведення символів з атрибутом. Атрибут завантажується до регістру BL.

Функція 13H переривання дозволяє виводити на екран рядок символів з встановленням атрибуту та зміщенням курсора. Адреса рядку завантажується до регістру BL, а довжина – до регістру CX.

Як можна побачити, використання переривання BIOS 10H теж не завжди є ефективним, особливо при виведенні рядків символів. Проте через використання цього апаратного переривання зручно здійснювати елементарні екранні операції.

Переривання MS-DOS 21H є більш зручним при практичному рішенні задач виведення інформації на екран. Головні його функції такі:

Функція 06 – Забезпечує введення символу з клавіатури та його виведення на екран. Якщо в регістр DL завантажене число FFH, то буде переривання буде здіснювати введення символу з клавіатури. ASCII-код символу буде завантажений до регістру AL. А якщо в регістр DL завантажене число, яке меньше за FFH, то воно буде ідентификовано як розширений ASCII-код символа, що треба вивести на екран.

Функція 09 – Вивод рядка символів. Дуже зручна функція, яка часто-густо використовується. Виведення на екран проводиться послідовно з сусідніх комірок пам’яті до тих пір, доки не зустрінеться символ долару ($). Наприклад:

```
N1    DB    ‘Групи ФІ-51, ФІ-52’
……..
      MOV AH, O9
      LEA   DX, M1
      INT   21H
```

Додаток А. Система команд МП *i*8086

Система команд МП *i*8086 (табл.А.1) містить 91 мнемокод. Усі команди МП можна поділити на п’ять груп:

1) команди передачі інформації (команди пересилки, роботи зі стеком, введення-виведення);
2) команди обробки інформації (арифметичні, логічні, команди зсуву);
3) рядкові команди;
4) команди передачі керування, включаючи команди переривань;
5) команди керування станом МП.

У табл. А.1 вжито такі позначення:

src – операнд-джерело;

dest – операнд-призначення;

reg – 8- /16-розрядний РЗП;

*reg*8 – 8-розрядний РЗП;

*reg*16 – 16-розрядний РЗП;

sr – сегментний регістр;

mem – 8-/16-розрядна комірка пам’яті;

*mem*8 – 8-розрядна комірка пам’яті;

*mem*16 – 16-розрядна комірка пам’яті;

r/m – 8-/16-розрядний регістр або комірка пам’яті;

r/m/i – 8-/16-розрядний регістр, комірка пам’яті або безпосередній операнд;

immed – безпосередній операнд;

disp – 8-/16-розрядне зміщення при заданні адреси;

*disp*8 – 8-розрядне зміщення;

*disp*16 – 16-розрядне зміщення;

target – мітка, до якої здійснюється перехід;

seg target – перша логічна адреса (сегментний адрес) мітки target;

offset target – друга логічна адреса (зміщення у сегменті) мітки *target*;

A – акумулятор *AL* або *AX*;

m[*disp*] – комірка пам’яті з ефективною адресою *EA* = *disp*.

Таблиця А.1– Система команд мікропроцесора *i*8086

Мнемокод команди	Опис команди	Алгоритм команди	Кількість байт	Кількість тактів
1	2	3	4	5
КОМАНДИ ПЕРЕДАЧІ ІНФОРМАЦІЇ				
Команди пересилання				
MOV dest, src	Пересилання даних з регістра, комірки пам'яті або безпосереднього операнда у регістр або пам'ять	*reg* ← *reg*	2	2
		sr ← *reg*	2	2
		reg ← *sr*	2	2
		mem ← *reg*	2-4[1)]	9+*EA*
		reg ← *mem*	2-4[1)]	8+*EA*
		mem ← *sr*	2-4[1)]	9+*EA*
		sr ← *mem*	2-4[1)]	8+*EA*
		a ← *mem*	3	11
		mem ← *a*	3	11
		*mem*8 ← *immed*	3-5[2)]	10+*EA*
		*mem*16 ← *immed*	4-6[3)]	10+*EA*
		*reg*8 ← *immed*	2	4
		*reg*16 ← *immed*	3	4

Продовження таблиці А.1

1	2	3	4	5
XCHG r/m, reg	Обмін даними між регістрами або регістром і пам'яттю	*reg* ←→ *reg* *mem* ←→ *reg* *A* ←→ *reg*	2 2-4[1)] 1	4 17+*EA* 3
XLAT	Перекодування вмісту *AL* в значення байта пам'яті з адресою *ES*: [*BX* + (*AL*)]	*AL* ← *ES*: [*BX* + (*AL*)]	1	11
*LEA reg*16, *mem*	Завантаження ефективної адреси комірки пам'яті *mem* у регістр	*reg* ← *EA*	2-4[1)]	2+*EA*
*LDS reg*16, *mem*	Завантаження в регістр reg16 слова із комірки пам'яті за адресою [*mem*], у *DS* – наступного слова з комірки за адресою [*mem* + 2]	*reg* ← [*mem*] *DS* ← [*mem*+2]	2-4[1)]	16+*EA*
*LES reg*16, *mem*	Завантаження в регістр reg16 слова із комірки пам'яті за адресою [*mem*], в *ES* – наступного слова з комірки за адресою [*mem* + 2]	*reg* ← [*mem*] *ES* ← [*mem* + 2]	2-4[1)]	16+*EA*
LAHF	Завантаження молодшого байта регістра прапорців *FL* в *AH*	*AH* ← *FL*	1	4
SAHF	Збереження *AH* у молодшому байті регістра прапорців *FL*	*FL* ← *AH*	1	4
Команди роботи зі стеком				
PUSH r/m/sr	Пересилання слова з регістра або з пам'яті у стек	*SP* ← *SP* – 2 [*SS*:*SP*] ← *r/m*	2-4[1)]	11/(16 + + *EA*)
		SP ← *SP* – 2 [*SS*:*SP*] ← *sr*	1	10

Продовження таблиці А.1

1	2	3	4	5
PUSHF	Пересилання у стек вмісту регістра прапорців	*SP* ← *SP* – 2 [*SS*:*SP*] ← *F*	1	10
POP r/m/sr	Пересилання слова даних зі стека у регістр або пам'ять	*r/m* ← [*SS*:*SP*] *SP* ← *SP* + 2	2-4[1)]	8/(16 ++ *EA*)
		sr ← [*SS*:*SP*] *SP* ← *SP*+2	1	8
POPF	Пересилання даних зі стека у регістр прапорців	*F* ← [*SS*:*SP*] *SP* ← *SP* + 2	1	8
Команди введення/виведення				
IN AL, *P*8 *IN AL*, *DX*	Введення в акумулятор *AL* байта з 8-розрядного порту з адресою *P*8 або з адресою, що зберігається в *DX*	*AL* ← Порт (*P*8) *AL* ← Порт (*DX*)	2 1	10 8
IN AX, *P*8 *IN AX*, *DX*	Введення в акумулятор *AX* слова з 16-розрядного порту з адресою *P*8 або з адресою, що зберігається в *DX*	*AX* ← Порт (*P*8) *AX* ← Порт (*DX*)	2 1	10 8
*OUT P*8, *AL* *OUT DX*, *AL*	Виведення байта з акумулятора *AL* у 8-розрядний порт з адресою *P*8 або з адресою, що зберігається в *DX*	Порт (*P*8) ← *AL* Порт (*DX*) ← *AL*	2 1	10 8
*OUT P*8, *AX* *OUT DX*, *AX*	Виведення слова з акумулятора *AX* у 16-розрядний порт з адресою *P*8 або з адресою, що зберігається в *DX*	Порт (*P*8) ← *AX* Порт (*DX*) ← *AX*	2 1	10 8

Продовження таблиці А.1

1	2	3	4	5
КОМАНДИ ОБРОБКИ ІНФОРМАЦІЇ				
Арифметичні команди				
ADD r/m, r/m/i	Додавання двох операндів	*r/reg* ← *r/reg* + *reg*	2	3
		reg ← *reg* + *r/m*	2-4[1)]	16 + *EA*
		*reg*8 ← *reg*8 + *immed*	2-4[1)]	9 + *EA*
		*reg*16 ← *reg*16 + *immed*	3	4
		*mem*8 ← *mem*8 + *immed*	4	4
		*mem*16 ← *mem*16 + *immed*	3-5[2)]	17 + *EA*
		AL ← *AL* + *immed*	2	4
		AX ← *AX* + *immed*	3	4
ADC r/m, r/m/i	Додавання двох операндів і прапорця перенесення *CF* від попередньої операції. Прапорець перенесення додається до молодшого біта результату	*reg* ← *reg* + *reg* + *CF*	2	3
		mem ← *mem* + *reg* + *CF*	2-4[1)]	16 + *EA*
		reg ← *reg* + *mem* + *CF*	2-4[1)]	9 + *EA*
		*reg*8 ← *reg*8 + *immed* + *CF*	3	4
		*reg*16 ← *reg*16 + *immed* + *CF*	4	4
		*mem*8 ← *mem*8 + *immed* + *CF*	3-5[2)]	17 + *EA*
		mem16 ← *mem16* + *imm* + *CF*	4-6[3)]	17 + *EA*
		AL ← *AL* + *immed* + *CF*	2	4
		AX ← *AX* + *immed* + *CF*	3	4
INC r/m	Інкремент (додавання з одиницею). Команда не діє на прапорець *CF*	*reg*8 ← *reg*8 + 1	2	3
		*reg*16 ← *reg*16 + 1	1	2
		mem ← *mem*+1	2-4[1)]	15 + *EA*
AAA	Корекція після додавання розпакованих двійково-десяткових чисел		1	4
DAA	Корекція після додавання упакованих двійково-десяткових чисел		1	4
SUB r/m, r/m/i	Віднімання двох операндів	*reg* ← *reg* – *reg*	2	3
		mem ← *me* – *reg*	2-4[1)]	16 + *EA*
		reg ← *reg* – *mem*	2-4[1)]	9 + *EA*

Продовження таблиці А.1

1	2	3	4	5
		reg8 ← *reg – immed*	3	4
		reg16 ← *reg1 – immed*	4	4
		mem8 ← *mem – immed*	3-5[2)]	17 + *EA*
		mem16 ← *mem1 – immed*	4-6[3)]	17 + *EA*
		AL ← *A – immed*	2	4
		AX ← *A – immed*	3	4
SBB r/m, r/m/i	Віднімання байта та прапорця позики *CF* від попередньої операції. Прапорець позики віднімається від молодшого біта результату	*reg* ← *reg – reg – CF*	2	3
		mem ← *mem – reg – CF*	2-4[1)]	16 + *EA*
		reg ← *reg – mem – CF*	2-4[1)]	9 + *EA*
		reg8 ← *reg8 – immed – CF*	3	4
		*reg*16 ← *reg16 – immed – CF*	4	4
		mem8 ← *mem8 – immed – CF*	3-5[2)]	17 + *EA*
		mem16← *mem16– immed –CF*	4-6[3)]	17 + *EA*
		AL ← *AL – immed – CF*	2	4
		AX ← *AX – immed – CF*	3	4
DEC r/m	Декремент (віднімання одиниці). Команда не діє на прапорець *CF*)	*reg*8 ← *reg*8 – 1	2	3
		*reg*16 ← *reg*16 – 1	1	2
		mem ← *mem* – 1	2-4[1)]	15 + *EA*
NEG r/m	Зміна знака операнда	*reg* ← – *reg*	3	3
		mem ← – *mem*	2-4[1)]	16 + *EA*
CMP r/m, r/m/i	Порівняння двох операндів – встановлення вмісту регістра прапорів *F* за результатом віднімання (без збереження результату віднімання)	*F* ⇐ *reg – reg*	2	3
		F ⇐ *mem – reg*	2-4[1)]	9 + *EA*
		F ⇐ *reg – mem*	2-4[1)]	9 + *EA*
		F ⇐ *reg*8 – *immed*	3	4
		F ⇐ *reg*16 – *immed*	4	4
		F ⇐ *mem*8 – *immed*	3-5[2)]	1 + *EA*
		F ⇐ *mem*16 – *imm*	4-6[3)]	1 + *EA*
		F ⇐ *AL – immed*	2	4
		F ⇐ *AX – immed*	3	4
AAS	Корекція після віднімання розпакованих двійково-десяткових чисел		1	4

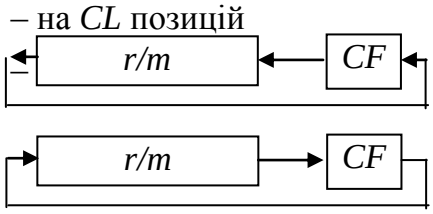
Продовження таблиці А.1

1	2	3	4	5
DAS	Корекція після віднімання упакованих двійково-десяткових чисел		1	4
MUL r/m	Множення *AL* (*AX*) на беззнакове значення *r/m*	$AX \leftarrow \|AL\| \times \|reg8\|$	2	70 – 77
		$(DX, AX) \leftarrow \|AX\| \times \|reg16\|$	2	118 – 133
		$AX \leftarrow \|AL\| \times \|mem8\|$	2-4[1)]	76 – 83 + *EA*
		$(DX, AX) \leftarrow \|AX\| \times \|mem16\|$	2-4[1)]	124 – 139 + *EA*
IMUL r/m	Множення *AL* (*AX*) на знакове значення *r/m*	$AX \leftarrow \|AL\| \times \|reg8\|$	2	80 – 98
		$(DX, AX) \leftarrow \|AX\| \times \|reg16\|$	2	128 – 154
		$AX \leftarrow \|AL\| \times \|mem8\|$	2-4[1)]	96 – 104 + + *EA*
		$(DX, AX) \leftarrow \|AX\| \times \|mem16\|$	2-4[1)]	134 – 160 + *EA*
AAM	Корекція після множення розпакованих двійково-десяткових чисел		2	83
DIV r/m	Ділення акумулятора на беззнакове число (ділення на нуль викликає переривання *INT* 0)	$\|AX\| : \|reg8\| \rightarrow AL$ (остача *AH*)	2	80 – 90
		$\|DX, AX\| : reg16 \rightarrow AX$ (остача *DX*)	2	144 – 162
		$\|AX\| : \|mem8\| \rightarrow AL$ (остача *AH*)	2-4[1)]	86 – 96 + + *EA*
		$\|DX, AX\| : \|mem16\| \rightarrow AX$ (остача *DX*)	2-4[1)]	150 – 168 + EA
IDIV r/m	Ділення акумулятора на ціле число (8- або 16-розрядне). Ділення на нуль викликає переривання *INT* 0	$AX : reg8 \rightarrow AL$ (остача *AH*)	2	101 – 112
		$(DX, AX) : reg16 \rightarrow AX$ (остача *DX*)	2	165 – 184
		$AX : mem8 \rightarrow AL$ (остача *AH*)	2-4[1)]	144 – 168 + + *EA*
		$DX, AX : mem16 \rightarrow AX$ (остача *DX*)	2-4[1)]	166 – 190 + + *EA*

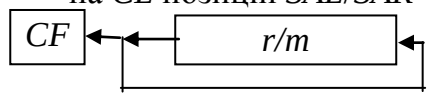
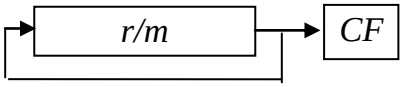
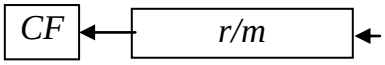
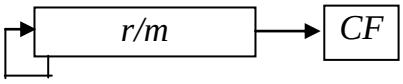
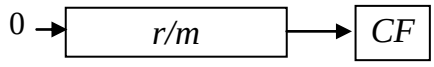
Продовження таблиці А.1

1	2	3	4	5
AAD	Корекція перед діленням розпакованих двійково-десяткових чисел		2	60
CBW	Перетворення байта *AL* на слово *AX* (повторення вмісту знакового розряду (*AL*.7) регістра *AL* в усіх розрядах регістра *AH*)	*AH* ← (*AL*7)	1	2
CWD	Перетворення слова *AX* на подвійне слово *DX*, *AX* (повторення вмісту знакового розряду (*AX*.15) регістра *AX* в усіх розрядах регістра *DX*)	*DX* ←*AX*15)	1	5
Логічні команди				
NOT r/m	Інверсія (інверсія всіх бітів операнда)	*reg* ← $\overline{\text{reg}}$ *mem* ← $\overline{\text{mem}}$	2 2-4[1)]	3 16 + *EA*
AND r/m, r/m/i	Логічне І двох операндів	*reg* ← *reg* ∧ *reg*	2	3
		mem ← *mem* ∧ *reg*	2-4[1)]	16 + *EA*
		reg ← *reg* ∧ *mem*	2-4[1)]	9 + *EA*
		*reg*8 ← *reg*8 ∧ *immed*	3	4
		*reg*16 ← *reg*16 ∧ *immed*	4	4
		*mem*8 ← *mem*8 ∧ *immed*	3-5[2)]	17 + *EA*
		*mem*16←*mem*16 ∧ *immed*	4-6[3)]	17 + *EA*
		AL ← AL ∧ *immed*	2	4
		AX ← AX ∧ *immed*	3	4
OR r/m, r/m/i	Логічне АБО двох операндів	*reg* ← *reg* ∨ *reg* *mem* ← *mem* ∨ *reg* *reg* ← *reg* ∨ *mem* *reg*8 ← *reg*8 ∨ *immed* *reg*16 ← *reg*16 ∨ *immed*	2 2-4[1)] 2-4[1)] 3 4	3 16 + *EA* 9 + *EA* 4 4

Продовження таблиці А.1

1	2	3	4	5
		*mem*8 ← *mem*8 ∨ *immed*	3-5[2)]	17 + *EA*
		*mem*16 ← *mem*16 ∨ *immed*	4-6[3)]	17 + *EA*
		AL ← *AL* ∨ *immed*	2	4
		AX ← *AX* ∨ *immed*	3	4
XOR r/m, r/m/i	Виключальне АБО	*reg* ← *reg* ⊕ *reg*	2	3
		mem ← *mem* ⊕ *reg*	2-4[1)]	16 + *EA*
		reg ← *reg* ⊕ *mem*	2-4[1)]	9 + *EA*
		*reg*8 ← *reg*8 ⊕ *immed*	3	4
		*reg*16 ← *reg*16 ⊕ *immed*	4	4
		*mem*8 ← *mem8* ⊕ *immed*	3-5[2)]	17 + *EA*
		*mem*16 ← *mem16* ⊕ *immed*	4-6[3)]	17 + *EA*
		AL ← *AL* ⊕ *immed*	2	4
		AX ← *AX* ⊕ *immed*	3	4
TEST r/m, r/m/i	Перевірка (логічне І без запису результату і встановлення прапорців відповідно до результату)	*F* ⇐ *reg* ∧ *reg*	2	3
		F ⇐ *mem* ∧ *reg*	2-4[1)]	9 + *EA*
		F ⇐ *reg* ∧ *mem*	2-4[1)]	9 + *EA*
		F ⇐ *reg*8 ∧ *immed*	3	4
		F ⇐ *reg*16 ∧ *immed*	4	4
		F ⇐ *mem*8 ∧ *immed*	3-5[2)]	10 + *EA*
		F ⇐ *mem*16 ∧ *immed*	4-6[3)]	10 + *EA*
		F ⇐ *AL* ∧ *immed*	2	4
		F ⇐ *AX* ∧ *immed*	3	4
Команди зсуву				
RCL/RCR r, 1 *RCL/RCR m*, 1 *RCL/RCR r, CL* *RCL/RCR m, CL*	Циклічний зсув ліворуч/ праворуч через біт *CF* – на одну позицію – на *CL* позицій  		2 2-4[1)] 2 2-4[1)]	2 15 + *EA* 8 + *4CL* 20 + + *EA* + + *4CL*

Продовження таблиці А.1

1	2	3	4	5
ROL/ROR r, 1 *ROL/ROR m*, 1 *ROL/ROR r, CL* *ROL/ROR m, CL*	Циклічний зсув ліворуч/ праворуч – на одну позицію – на *CL* позицій *SAL/SAR*  ліворуч  праворуч		2 2-4[1)] 2 2-4[1)]	2 15 + *EA* 8 + 4*CL* 20 + + *EA* + + 4*CL*
SAL/SAR r, 1 *SAL/SAR m*, 1 *SAL/SAR r, CL* *SAL/SAR m, CL*	Зсув арифметичний ліворуч/ праворуч – на одну позицію – на *CL* позицій  ліворуч  праворуч При зсуву праворуч значення біта у старшому розряді залишається незмінним		2 2-4[1)] 2 2-4[1)]	2 15 + EA 8 + 4CL 20 + + EA + + 4CL
SHR r, 1 S*HR m*, 1 *SHR r, CL* *SHR m, CL*	Зсув логічний праворуч [4)] – на одну позицію – на *CL* позицій 		2 2-4[1)] 2 2-4[1)]	2 15 + *EA* 8 + *4CL* 20 + + *EA* + + *4CL*
Рядкові команди				
REP	Префікс повторення рядкових операцій до онулення *CX* (*CX* декрементується після виконання кожної рядкової операції)		1	

Продовження таблиці А.1

1	2	3	4	5
REPE (*REPZ*)	Префікс умовного повторення – повторення при *ZF* = 1, або до обнуління *CF*		1	
REPNE (*REPNZ*)	Префікс умовного повторення – повторення при *ZF* = 0, або до обнуління *CF*		1	
MOVSB	Копіювання байта з комірки пам'яті з адресою *DS*: [*SI*] в комірку *ES*:[*DI*]	ES: [*DI*]← DS: [SI] *SI*← *SI* ± 1[7]; *DI* ← *DI* ± 1[7]	1	18[5] 9 + 17 *CX*[6]
MOVSW	Копіювання слова з *DS*: [*SI*] в *ES*: [*DI*]	*ES*: [*DI*] ← *DS*: [*SI*] *SI* ← *SI*±2[7]; *DI* ← *DI*±2[7]	1	18[5] 9 + 17 *CX*[6]
LODSB	Копіювання байта з *DS*: [*SI*] в *AL*	*AL* ← *DS*: [*SI*] *S I*← *SI*±1[7]	1	11[5] 9 + 10 *CX*[6]
LODSW	Копіювання слова з *DS*: [*SI*] в *AX*	*AX* ← *DS*: [*SI*] *SI* ← *SI*±2[7]	1	11[5] 9 +1 0 *CX*[6]
STOSB	Запис байта з *AL* в *ES*: [*DI*]	*ES*: [*DI*] ← *AL* *DI* ← *DI* ± 1[7]	1	11[5] 9 + *10 CX*[6]
STOSW	Запис слова з *AX* в *ES*: [*DI*]	*ES*: [*DI*] ← *AX* *DI* ← *DI* ± 2[7]	1	11[5] 9 + *10 CX*[6]
CMPSB	Порівняння байтів *DS*: [*SI*] і *ES*: [*DI*] із записом результату порівняння у регістру прапорців	*F* ⇐ *ES*: [*DI*] – *DS*: [*SI*] *SI* ← ← *SI* ± 1[7]; *DI* ← *DI* ± 1[7]	1	22[5] 9 + 22 *CX*[6]
CMPSW	Порівняння слів *DS*: [*SI*] і *ES*: [*DI*] із записом результату порівняння у регістр прапорців	*F* ⇐ *ES*: [*DI*] – *DS*: [*SI*] *SI* ← ← *SI* ± 2[7]; *DI* ← *D I*± 2[7]	1	22[5] 9 + 22 *CX*[6]

Продовження таблиці А.1

1	2	3	4	5
SCASB	Порівняння байта *DS*: [*SI*] і *AL* із записом результату порівняння у регістр прапорців	*F* ⇐ [*DS*: *SI*] –*AL*; *SI* ← *SI* ± 1[7)]	1	15[5)] 9 + 15 *CX*[6)]
SCASW	Порівняння слова [*DS*: *SI*] і *AX* із записом результату порівняння у регістр прапорців	*F* ⇐ [*DS*: *SI*] – *AX*; *SI* ← *SI* ± 2[7)]	1	15[5)] 9 + 15 *CX*[6)]
КОМАНДИ ПЕРЕДАЧІ КЕРУВАННЯ				
*JMP target*16 *JMP NEAR target*8 *JMP reg* *JMP mem*	Внутрішньосегментний безумовний перехід до цільової адреси *target* (перехід у межах сегмента довжиною 64 Кбайт)	*IP* ← *IP* + *disp*16 *IP* ← *IP* + *disp*8 *IP* ← *IP* + *reg* *IP* ← *IP* + *mem*	3 2 2 2-4[1)]	15 15 2 18 + *EA*
JMP FAR target *JMP FAR mem*	Міжсегментний безумовний перехід до цільової адреси *target* (перехід у межах усієї ємності пам'яті 1 Мбайт)	*IP* ← *offset target*, *CS* ← *seg target* *IP* ← [*mem*] *IP* ← [*mem*+2]	5 2-4[1)]	15 24 + *EA*
JCX target	Перехід, якщо *CX* = 0		2	18/6 [8)]
LOOP target	Цикл: *CX* ← *CX*-1 і перехід, якщо *CX* ≠ 0		2	16/4 [8)]
LOOPE (*LOOPZ*) *target*	*CX* ← *CX*-1 і перехід, якщо *CX* ≠ 0 і *ZF* = 1		2	18/6 [8)]
LOOPNE (*LOOPNZ*) *target*	*CX* ← *CX* – 1 і перехід, якщо *CX* ≠ 0 і *ZF* = 0		2	19/5 [8)]
JA (*JNBE*) *target*	Перехід, якщо перший беззнаковий операнд більше за другий (*CF* = *ZF* = 0)		2	16/4 [8)]
JAE (*JNB*) *target*	Перехід, якщо перше беззнакове число не менше за друге (*CF* = 0)		2	16/4 [8)]
JB (*JC*) *target*	Перехід, якщо перше беззнакове число менше за друге (*CF* = 1)		2	16/4 [8)]

Продовження таблиці А.1

1	2	3	4	5
JE (*JZ*) *target*	Перехід, якщо числа дорівнюють (*ZF* = 1)		2	16/4 [8)]
JG (*JNLE*) *target*	Перехід, якщо перше більше за друге знакове число (*SF* = (*ZF*&*OF*))		2	16/4 [8)]
JGE (*JNL*) *target*	Перехід, якщо перше знакове число більше або дорівнює другому (*SF* = *OF*)		2	16/4 [8)]
JL (*JNGE*) *target*	Перехід, якщо перше знакове число менше за друге (*SF* ≠ *OF*)		2	16/4 [8)]
JLE (JNG) target	Перехід, якщо перше знакове число менше або дорівнює другому (*SF* ≠ *OF* або *ZF* = 0)		2	16/4 [8)]
JNC (JAE/JNB)target	Перехід, якщо немає переносу (*CF* = 0)		2	16/4 [8)]
JNE (JNZ) target	Перехід, якщо числа не рівні (*ZF* = 0)		2	16/4 [8)]
JNO target	Перехід, якщо немає переповнення (*OF* = 0)		2	16/4 [8)]
JNP (JPO) target	Перехід, якщо паритет непарний (*PF* = 0)		2	16/4 [8)]
JNS target	Перехід, якщо позитивний результат (*SF* = 0)		2	16/4 [8)]
JO target	Перехід, якщо є переповнення (*OF* = 1)		2	16/4 [8)]
JP (JPE) target	Перехід, якщо паритет парний (*PF* = 1)		2	16/4 [8)]
JS target	Перехід, якщо негативний результат (*SF* = 1)		2	16/4 [8)]
CALL NEAR target	Внутрішньосегментний виклик процедури (виклик у межах сегмента довжиною 64 Кбайт)	*SP* ← *SP* – 2 [*SS*:*SP*] ← *IP*; *IP* ← *target*	3	19
CALL NEAR reg		*IP* ← *reg*	2	16
CALL NEAR mem		*IP* ← *mem*	2 – 4 [1)]	21 + *EA*

Продовження таблиці А.1

1	2	3	4	5
CALL FAR target *CALL FAR mem*	Міжсегментний виклик процедури (виклик у межах усієї ємності пам'яті 1 Мбайт)	*SP* ← *SP* – 2 [*SS*:*SP*] ← *CS* *SP* ← *SP* – 2 [SS:SP] ← IP; *CS*, *IP* ← *target* *IP* ← [*mem*]; *CS* ← [*mem* + 2]	5 2-4[1)]	28 37 + *EA*
CALL FAR target *RET (n)* *RET NEAR (n)*	Повернення з внутрішньосегментної процедури. Необов'язковий параметр *n* задає корекцію значення вказівника стеку	*IP* ← [*SS*:*SP*]; *SP* ← *SP* + 2 *IP* ← *[SS:SP];* *SP* ←*SP* + *n*	1 3	8 12
RET FAR *RET FAR (n)*	Повернення з міжсегментної процедури Необов'язковий параметр *n* задає корекцію значення вказівника стека	*IP* ← [*SS*:*SP*]; *SP* ←*SP* + 2 *CS* ← [*SS*:*SP*]; *SP* ← *SP* + 2;	1 3	18 17
Команди переривань				
INT n	Виконання програмного переривання	*SP* ← *SP* – 2 [*SS*:*SP*] ← F *SP*←*SP* – 2 [*SS*:*SP*] ← *CS* *SP* ← *SP* – 2 [*SS*:*SP*] ← *IP*	2	51
INT 3	Виконання програмного переривання 3		1	52
INTO	Виконання програмного переривання 4, якщо прапорець *OF* = 1		1	53/4 [9)]
IRET	Повернення з переривання	*IP* ← [*SS*:*SP*]; *SP* ←*SP*+2 *CS* ← [*SS*:*SP*]; *SP* ← *SP*+2; *F* ← [*SS*:*SP*]; *SP* ← *SP*+2	1	24

Продовження таблиці А.1

1	2	3	4	5
КОМАНДИ КЕРУВАННЯ СТАНОМ МП				
CLC	Скидання прапорця перенесення	$CF \leftarrow 0$	1	2
CMC	Інверсія прапорця перенесення	$CF \leftarrow \overline{CF}$	1	2
STC	Установлення прапорця перенесення	$CF \leftarrow 1$	1	2
CLD	Скидання прапорця напряму	$DF \leftarrow 0$	1	2
STD	Установлення прапорця напряму	$DF \leftarrow 1$	1	2
CLI	Заборона маскованих апаратних переривань	$IF \leftarrow 0$	1	2
STI	Дозвіл маскованих апаратних переривань	$IF \leftarrow 1$	1	2
HLT	Зупин процесора		1	2
WAIT	Очікування сигналу на лінії *TEST*		1	3
ESC msk/mem	Передача коду команди *msk* або коду і операнда *mem* арифметичному співпроцесору		2 2-4[1)]	2 8 + *EA*
LOCK	Префікс блокування шини на час виконання наступної інструкції у максимальному режимі		1	2
NOP	Немає операцій		1	2

[1)] Команда займає два байти, якщо при адресації комірки пам’яті не використовується зміщення, тобто *disp*=0, наприклад, позначення комірки пам’яті *mem* відповідає позначенню [*SI*]; команда займає три байти, якщо використовується 8-розрядне зміщення *disp*8, наприклад, [*SI* + 25*H*]; команда займає чотири байти, якщо зміщення16-розрядне – *disp16*, наприклад, [*SI* + 1000*H*].

[2)] Команда займає три байти при *disp*=0, чотири байти при *disp8* і п’ять байт при *disp*16.

[3)] Команда займає чотири байти при *disp*=0, п’ять байт при *disp8* і шість байт при *disp*16.

[4)] Логічний зсув логічний ліворуч збігається з арифметичним зсувом ліворуч. Допускається замість позначення *SAL* використовувати позначення *SHL*.

[5] Наведено час виконання рядкової команди без префікса повторення.

[6] Наведено час виконання рядкової команди із префіксом повторення. У регістрі *CX* записано кількість повторень.

[7] При встановленому прапорці напряму (*DF* = 1) – операція додавання, у противному разі – віднімання.

[8] *m*/*n* – при виконанні переходу команда виконується за m тактів, у противному разі – за *n* тактів.

[9] При встановленому прапорці (*OF* = 1) команда виконується за 53 тактів, у протилежному випадку – за чотири такти.

Значення кількості тактів *EA*, необхідне для обчислення ефективної адреси, залежить від способу адресації операнда (табл. А.2).

Таблиця А.2 – Обчислення ефективної адреси *EA*

Адресація	Спосіб обчислення	Кількість тактів
Пряма	[*disp*]	6
Непряма	[*BX*], [*BP*], [*SI*], [*DI*]	5
Базова або індексна	[*BX* + *disp*], [*BP* + *disp*], [*SI* + *disp*], [*DI* + *disp*]	9
Базово-індексна без зміщення	[*BP* + *DI*], [*BX* + *SI*], [*BP* + *SI*], [*BX* + *DI*]	7 8
Базово-індексна зі зміщенням	[*BP* + *DI* + *disp*], [*BX* + *SI* + *disp*], [*BP* + *SI* + *disp*], [*BX* + *DI* + *disp*]	11 11 12 12

Вплив команд на значення прапорців ілюструє табл.А.3, в якій позначено: “**+**” – команда впливає на прапорець; “**–**“ – не впливає; “**1**” – встановлює прапорець в одиницю; “**0**” – скидає прапорець в нуль; “**?**” – стан невизначений (залежить від конкретних значень операндів).

Таблиця А.3 – Встановлення прапорців

Операція	Команди	Прапорці

		OF	*CF*	*AF*	*SF*	*ZF*	*PF*	*DF*	*IF*	*TF*
Додавання, віднімання	*ADD, ADC, SUB, SBB*	+	+	+	+	+	+	-	-	-
	CMP, NEG, CMPS, SCAS	+	+	+	+	+	+	-	-	-
	INC, DEC	+	-	+	+	+	+	-	-	-
Множення	*MUL, IMUL*	+	+	?	?	?	?	-	-	-
Ділення	*DIV, IDIV*	?	?	?	?	?	?	-	-	-
Десяткова корекція	*DAA, DAS,*	?	+	+	+	+	+	-	-	-
	AAA, AAS	?	+	+	?	?	?	-	-	-
	AAM, AAD	?	?	?	+	+	+	-	-	-
Логічні команди	*AND, OR, XOR, TEST*	0	0	?	+	+	+	-	-	-
Зсув	*RCL, RCR,*	+	+	?	-	-	-	-	-	-
	ROL, ROR dest - dest, CL	?	+	?	-	-	-	-	-	-
	SHL, SHR dest	+	+	?	+	+	+	-	-	-
	-dest, CL	?	+	?	+	+	+	-	-	-
	SAR	0	+	?	+	+	+	-	-	-
Відновлення прапорців	*POPF, IRET*	+	+	+	+	+	+	+	+	+
	SAHF	-	+	+	+	+	+	-	-	-
Переривання	*INT, INTO*	-	-	-	-	-	-	-	0	0
Керування прапорцями	*STC*	-	1	-	-	-	-	-	-	-
	CLC	-	0	-	-	-	-	-	-	-
	CMC	-	$\overline{CF}$	-	-	-	-	-	-	-
	STD	-	-	-	-	-	-	1	-	-
	CLD	-	-	-	-	-	-	0	-	-
	STI	-	-	-	-	-	-	-	1	-
	CLI	-	-	-	-	-	-	-	0	-

Додаток Б Функції переривання ОС MS-DOS та BIOS

ПЕРЕРИВАННЯ BIOS

INT 05H. Друк екрану
INT 10H. Управління дисплеєм
INT 11H. Запит списку приєднаного обладнання
Виявляє різноманітне обладнання, приєднане до системи, результат міститься в регістрі AX; після запуску, система виконує цю операцію і записує регістр AX в пам'ять за адресою 410H. Біти регистру AX мають такі значення:
15,14 Кількість принтерів;
13 Послідовний принтер;
12 Ігровий адаптер;
11..9 Кількість послідовних портів стандарту RS-232;
7,6 Кількість дисководів (якщо біт 0 дорівнює 1) - 00=1, 01=2, 10=3, 11=4;
5,4 Початковий відеорежим - 00 - не використовується, 01 - 40x25, кольоровий, 10 - 80x25, кольоровий, 11 - 80x25, монохромний;
1 Наявність сопроцесора (якщо 1);
0 Наявність дисководів (якщо 1).
INT 12H. Запит розміру фізичної пам'яті
INT 13H. Дискові операції вводу-виводу
INT 14H. Управління комунікаційним адаптером
INT 15H. Касетні операції вводу-виводу і спеціальні функції для комп'ютерів AT
INT 16H. Введення з клавіатури
INT 17H. Виведення на прінтер
INT 18H. Виклик системи БЕЙСІК, вбудованої до ROM
INT 19H. Перезапуск системи
INT 1AH. Запит і встановлення теперішнього часу та дати
Регістр AX встановлює код операції: 00 - запит (CX - старший байт, DX - молодший байт, якщо після останнього запиту пройшло більше 24 годин, то в регістр AL заноситься ненульове значення); 01 - запис (призначення регістрів CX та DX співпадає з їх значенням під час запиту часу).
INT 1FH. Адреси таблиці графічних символів

ПЕРЕРИВАННЯ DOS

INT 21H. Запит функцій DOS. Основна операція DOS, що викликає функцію DOS у відповідності з кодом в регістрі AH
INT 22H. Адрес підпрограми обробки завершення задачі (див. INT 24H)
INT 23H. Адрес підпрограми реакції на Ctrl/Break (див. INT 24H)
INT 24H. Адрес підпрограми реакції на фатальну помилку. У цьому елементі та двох попередніх містяться адреси, які ініціалізує ситема у префіксі програмного сегменту та їх можна змінити у власних цілях. Подробиці дивись у технічному описі DOS.
INT 25H. Абсолютне зчитування з диску
INT 26H. Абсолютне записування на диск

INT 27H. Завершення програми, що лишає її резидентною. Дозволяє лишити COM-програму у пам’яті. Подробиці дивись у технічному описі DOS.

ФУНКЦІЇ ПЕРИВАННЯ DOS INT 21H

Нижче наведені базові функції для переривання DOS INT 21H. Код функції установлюється у регистрі AH:

00 Завершення програми (аналогічно до INT 20H)
01 Введення символу з клавиатури з відображенням на екрані
02 Виведення символу на екран
03 Введення символу з асинхроного комунікаційного каналу
04 Виведення символу на асинхроний комунікаційний канал
05 Виведення символу на пристрій друку
06 Пряме введення з клавиатури та виведення на екран
07 Введення з клавиатури без відображення та без перевірки Ctrl/Break
08 Введення з клавиатури без відображення з перевіркою Ctrl/Break
09 Виведення рядку символів на екран
0A Введення з клавиатури з буферізацією
0B Перевірка наявності введення з клавиатури
0C Очищення буферу введення з клавиатури та запит на введення
0D Зброс диску
0E Встановлення поточного дисководу
0F Відкриття файлу через PCB
10 Закриття файлу через PCB
11 Початковий пошук файлу за шаблоном
12 Пошук наступного файлу за шаблоном
13 Вилучення файлу з диску
14 Послідовне читання файлу
15 Послідовний запис файлу
16 Створення файлу
17 Переіменування файлу
18 Внутрішня операція DOS
19 Визначення поточного дисководу
1A Встановлення області передання даних (DTA)
1B Отримання таблиці FAT для поточного дисководу
1C Отримання таблиці FAT для будь-якого дисковода.
21 Читання з диску з прямим доступом
22 Запис на диск з прямим доступом
23 Визначення розміру файлу
24 Встановлення номеру запису для прямого доступу
25 Встановлення вектору переривань
26 Створення програмного сегменту
27 Читання блоку записів з прямим доступом
28 Запис блоку з прямим доступом
29 Перетворення назви файлу до внутрішніх параметрів

2A Отримання дати (CX-рік, DH- місяць, DL-день).
2B Встановлення дати
2C Отримання часу (CH-година, CL-хвилина, DH-секунда, DL-1/100секунди)
20 Встановлення часу
2E Встановлення/відміна верифікації запису на диск
Наступні розширені функції можливі у DOS починаючи з версії 2.0:
2F Отримання адреси DTA у регістровій парі ES:BX
30 Отримання номеру версії DOS у регістрі AX
31 Завершенння програми, після якого вона лишається резидентною у пам’яті
33 Перевірка Ctrl/Break
35 Отримання вектору переривання (адреси підпрограми)
36 Отримання розміру вільного простору на диску
38 Отримання державно-залежних форматів
39 Створення підкаталогу (команда MKDIR)
3A Вилучення підкаталогу (команда RMDIR)
3B Встановлення поточного каталогу (команда CHDIR)
3C Створення файлу без використання FCB
30 Відкриття файлу без використання FCB
3E Закриття файлу без використання FCB
3F Читання з файлу або введення з пристрою
40 Запис до файлу або введення до пристрою
41 Вилучення файлу з каталогу
42 Встановлення позиції для послідовного доступу
43 Зміна атрибутів файлу
44 Зміна атрибутів файлу
45 Зміна атрибутів файлу
46 "Зклеювання" дубльованих файлових номерів
47 Отримання поточного каталогу
48 . Виділення пам’яті з вільного простору
49 . Вивільнення виділеної пам’яті
4A Зміна довжини блоку виділеної пам’яті
4B Завантаження/виконання програми(подпроцесу)
4C Завершення підпроцесу з поверненням до керування
4D Отримання коду завершення підпроцесу
4E Початковий пошук файлу за шаблоном
4F Пошук наступного файлу за шаблоном
54 Отримання стану верифікації
56 Перейменування файлу
57 Отримання/встановлення дати та часу зміни файлу
Наступні розширені функції передбачені у DOS починаючи з версії 3.0:
59 Отримання розширеного кода помилки
5A Створення тимчасового файлу
5B Створення нового файлу
5C Блокування/розблокування доступу до файлу
62 Отримання адреси префиксу програмного сегменту (PSP)

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Жуйков В.Я., Терещенко Т.О., Петергеря Ю.С. Електронний підручник «Мікропроцесори і мікроконтролери» -- 2009 Гриф надано Міністерством освіти і науки України (лист № 1.4_18-Г-114 від 10.01.2009 р. - режим доступу до ресурсу:. http://www.kaf-pe.ntu-kpi.kiev.ua
2. Мікропроцесорна техніка. Друге видання. Доповнене./ Ю.І. Якименко, Т.О. Терещенко, Є.І. Сокол, В.Я. Жуйков, Ю.С. Петергеря. За ред. Т.О. Терещенко. – Київ, 2004. – 440 с
3. Жуйков В.Я, Терещенко Т.О., Ямненко Ю.С. Заграничний А.В. Електронний підручник "Мікропроцесорна техніка". - Рекомендовано до друку Вченою Радою НТУУ «КПІ» протокол №6 від 16.05.2016 р. режим доступу до ресурсу: http://kaf-pe.kpi.ua/?page_id=675, http://ela.kpi.ua/handle/123456789/18969
4. Мікропроцесорна техніка : підручник / В. Я. Жуйков, Т. О. Терещенко, Ю. С. Ямненко – 3-тє вид., перероб. і допов. – Київ: НТУУ «КПІ» Вид-во «Політехніка», 2015. – 440
5. Мікропроцесорна техніка: Навчальний посібник / В.Я. Жуйков, О.І. Захожай, Ю.Е Паеранд, Т. О. Терещенко Алчевськ: ДонДГУ, 2013 – 497 с.
6. Гук М. Процессоры Intel от 8086 до Pentium II. Санкт-Петербург,Питер Паблишинг, 1997
7. Мікропроцесорна техніка : навч. посіб. / В. В. Ткачов, Г. Грулер, Н. Нойбергер, С. М. Проценко, М. В. Козарь; ДВНЗ "Нац. гірн. ун-т". - Д. : НГУ, 2012. - 188 с. - Бібліогр.: с. 188 - укр.
8. Возняк О. Основи мікропроцесорної техніки Львівська філіяДніпропетровського національного університету залізничного транспорту імені академіка В.Лазаряна – 2017 - режим доступу до ресурсу: http://vozom.ho.ua/MP/
9. Терещенко Т. О., Тодоренко В.А., Батрак Л.М., Ямненко Ю. С. Мікропроцесорні пристрої. Навчальний посібник для студентів спеціальності «Електроніка». - К.: НТУУ «КПІ ім. Ігоря Сікорського», 2017. - 244с.
10. Рябенький В.М., Жуйков В.Я., Ямненко Ю.С., Заграничний А.В. Електронний підручник "Схемотехніка: Пристрої цифрової електроніки" у двох томах. - Рекомендовано до друку Вченою Радою НТУУ «КПІ», протокол №6 від 16.05.2016 р. режим доступу до ресурсу: http://kaf-pe.kpi.ua/?page_id=675, http://ela.kpi.ua/handle/123456789/18970
11. Схемотехніка: Пристрої цифрової електроніки: підручник. У 2 т. / В. М. Рябенький, В. Я. Жуйков, Ю. С. Ямненко, О. В. Борисов. – Київ : НТУУ «КПІ» Вид-во «Політехніка», 2015. – Т.1. – 400 с. – 500 пр.
12. Схемотехніка: Пристрої цифрової електроніки: підручник. У 2 т. / В. М. Рябенький, В. Я. Жуйков, Ю. С. Ямненко, О. В. Борисов. – Київ : НТУУ «КПІ» Вид-во «Політехніка», 2015. – Т.2. – 360 с. – 500 пр.
13. Схемотехника электронных систем. Том 3. Микропроцессоры и микроконтроллеры / Бойко В.І., Гуржій А.М., Жуйков В.Я., Зорі А.А., Співак В.М., Терещенко Т.О, Петергеря Ю.С. - СПб.: БХВ Петербург, 2004. – 464 с.

14. Схемотехніка електронних систем. Том 3. Мікропроцесори та мікроконтролери / Бойко В.І., Гуржій А.М., Жуйков В.Я., Зорі А.А., Петергеря Ю.С., Співак В.М., Терещенко Т.О, Якименко Ю.І. - К.: Вища школа, 2004. – 460 с.