МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

**Н. С. НАЗАРОВА, Д. В. ВІННИЧЕНКО,
С. С. КОЗИРЄВ**

МЕТОДИЧНІ ВКАЗІВКИ

до практичних робіт з курсу

"Інформаційно-керуючі системи та комплекси"

Частина 1

Рекомендовано Методичною радою НУК

Миколаїв 2009

УДК 681.3.06

Назарова Н.С., Вінниченко Д.В., Козирєв С.С. Методичні вказівки до практичних робіт з курсу "Інформаційно-керуючі системи та комплекси": У 2 ч. – Миколаїв: НУК, 2009. – Ч.1. – 28 с.

Кафедра імпульсних процесів і технологій

Практичні роботи, що входять до методичних вказівок, містять вузлові питання по розробці керуючих програм і створенню завантажувальних модулів, які використовуються в інформаційно-керуючих системах електророзрядних установок. Час виконання кожної роботи складає 4 аудиторні години. З метою полегшення підготовки і проведення практичних робіт надаються деякі теоретичні відомості. Великий обсяг інформації міститься у інтерактивному довіднику [2] і довіднику ng.exe [3], що додатково надаються студенту. Це не виключає лекційної підготовки і використання рекомендованої літератури. Методичні вказівки призначено для студентів кафедри "Імпульсні процеси і технології" з метою забезпечення ґрунтовного вивчення дисципліни "Інформаційно-керуючі системи та комплекси" навчального плану підготовки бакалаврів за напрямом 050701 "Електротехніка та електротехнології".

Рецензент канд. техн. наук Овчиннікова Л.Є.

Згідно з наказом ректора НУК № 08 від 09.01.2008 методичні вказівки друкуються в авторській редакції і відповідальність за їх редагування несе автор.

ВСТУП

Для створення інформаційно-керуючих систем та комплексів використовуються промислові комп'ютери, у тому числі комп'ютери на базі Intel Pentium. Для оптимального програмування таких комп'ютерів необхідно добре знати їх архітектуру. Такі знання набуваються при вивченні мови низького рівня – асемблеру. Керування апаратними засобами інтерфейсів введення/виводу методами безпосереднього їх програмування з використанням переривань DOS і функцій API Windows застосовується також у мовах високого рівня. Тому інформація, наведена у методичних вказівках, дозволить отримати необхідні навики створення ефективних програм керування технологічними процесами.

Успішне вивчення основ дисципліни "Інформаційно-керуючі системи та комплекси" дозволить освоїти архітектуру сучасних мікропроцесорів і набути вміння розробки програмного забезпечення систем автоматичного керування установками для електророзрядних технологій.

ОРГАНІЗАЦІЯ І ПОРЯДОК ВИКОНАННЯ ПРАКТИЧНИХ РОБІТ

Практичні роботи з курсу "Інформаційно-керуючі системи та комплекси" є частиною однойменного курсу і призначені для практичного освоєння досліджуваного матеріалу.

Практичні заняття відбуваються в комп'ютерній лабораторії. Перед виконанням робіт студенти проходять загальний інструктаж.

До виконання практичної роботи студент допускається після теоретичної підготовки і засвоєння завдання поточної практичної роботи.

На початку занять викладач перевіряє готовність студента до виконання майбутньої роботи.

Студент приступає до виконання роботи з ДОЗВОЛУ викладача.

По кожній роботі оформляється звіт, що повинен містити: найменування роботи, мету виконання роботи, короткі теоретичні відомості, опис виконаного завдання, робочий лист виконаного завдання, висновки.

При здачі звіту викладач опитує студента в обсязі матеріалу виконаної роботи, після чого робота вважається зарахованою.

1. АРХІТЕКТУРА ПРОМИСЛОВИХ КОМП'ЮТЕРІВ

Практична робота № 1. Створення об`єктного коду і завантажувального модуля на базі текстового файлу з програмою на асемблері для *I* 8086. Налагоджувач *td.exe*

Мета роботи: засвоїти методику створення об`єктного коду і завантажувального модуля, отримати навики користування налагоджувачем *td.exe* і інформаційними довідниками.

Основні теоретичні відомості

Для виконання усіх функцій по введенню коду програми, її трансляції, редагуванню і налагодженню необхідно використовувати окремі службові програми. Більша частина їх входить до складу спеціалізованих пакетів асемблера.

При роботі Вам будуть потрібні 4 файли з пакету *TASM*. Тому рекомендується створити робочий каталог. Створити в каталозі *\TASM* підкаталоги *\TASM\WORK* і *\TASM\PROGRAM*. Каталог *PROGRAM* будемо використовувати для збереження налагоджених кодів програм і їхніх завантажувальних модулів (файли з розширенням *.exe*). Каталог *WORK* будемо використовувати як робочий. У ньому будуть знаходитися необхідні для одержання завантажувального модуля файли з пакета транслятора *TASM* і файл вихідного модуля, з яким ми в даний момент працюємо. Після того як помилки у вихідному модулі усунуті, він разом зі своїм модулем, що виконується, пересилається в каталог

PROGRAM. З каталогу *WORK* видаляються всі непотрібні файли – і він готовий для роботи з наступним вихідним модулем на асемблері. Таким чином, у каталозі *WORK* завжди знаходиться робоча версія програми, а в каталозі *PROGRAM* – налагоджена версія.

1. Помістити в каталог *WORK* файли *tasm.exe, tlink.exe, td.exe* і *rtm.exe*. Якщо ви щось пропустите, програми *tasm.exe* і *t1ink.exe* видадуть вам повідомлення про це.

2. Помістити файл **.asm* у каталог *WORK*.

Після всіх цих дій можна починати роботу.

На рис. 1 наведена загальна схема процесу розробки програми на асемблері на прикладі програми *first.asm*. На схемі виділено чотири кроки цього процесу. На першому кроці, коли вводиться код програми, можна використовувати будь-який текстовий редактор. Основною вимогою до нього є те, щоб він не вставляв сторонніх символів (спеціальних символів форматування). Файл повинен мати розширення *.asm*.

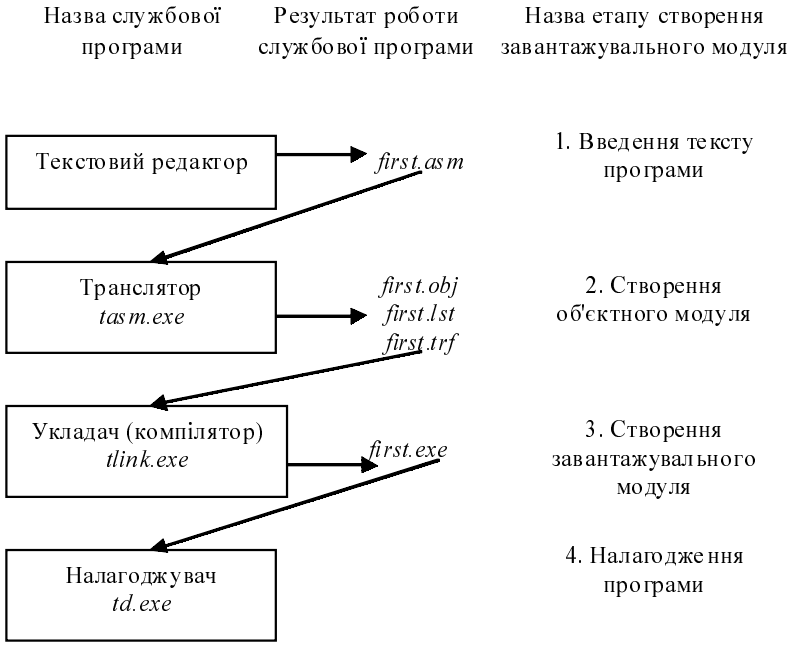

Рис. 1

Наступний крок – трансляція програми. На цьому кроці за допомогою транслятора *tasm.exe* з пакету *TASM* формується об'єктний модуль, що містить у собі вихідну програму в машинних кодах і деяку іншу інформацію, необхідну для налагодження і компонування створеного модуля з іншими модулями.

Формат командного рядка для запуску *tasm.exe*:

TASM [опції] ім'я_вихідного_файлу [,ім'я_об'єктного_файлу] [,ім'я_файлу_лістингу] [,ім'я_файлу_перехресних_посилань]

Інформацію про формат командного рядка і можливі значення параметрів можна одержати запустивши *tasm.exe* без задання будь-яких аргументів. Параметри укладені в квадратні дужки можуть бути відсутні. Таким чином, обов'язковим аргументом командного рядка є лише ім'я_вихідного_файлу. Цей файл повинен знаходитися на диску й обов'язково мати розширення *.asm*. За ім'ям вихідного файлу через кому можуть випливати необов'язкові аргументи, що позначають імена об'єктного файлу, файлу лістингу і файлу перехресних посилань. Якщо не задати їх, то відповідні файли попросту не будуть створені. Якщо ж їх необхідно створити, то якщо імена об'єктного файлу, файлу лістингу і файлу перехресних посилань повинні збігатися з ім'ям вихідного файлу (найбільш типовий випадок), то потрібно просто поставити коми замість імен цих файлів: tasm.exe *first*, , ,. У результаті будуть створені файли, як показано на рис.1, для кроку 2.

Перейдемо в каталог *WORK* і запустимо на трансляцію програму *first.asm* командним рядком виду:

tasm.exe /zi first, , ,

У результаті на екрані ви одержите послідовність рядків. Найперший з них буде інформувати вас про номер версії пакета *TASM*, що використовувався для трансляції даної програми. Далі йде рядок, що містить ім'я трансльованого файлу. Якщо ваша програма містить помилки, то транслятор видасть на екран рядка повідомлення, що починаються словами "*Error*" і "*Warning*". Програма уроку *first* (див. лістинг 1) синтаксично правильна, але з навчальною метою ви можете внести яку-небудь нісенітницю і подивитися на результат. Наявність рядка з "*Error*" говорить про те, що у вас у програмі є неприпустимі, з точки зору синтаксису, комбінації символів. Логіка роботи програми для транслятора не має ніякого значення. Наявність рядка "*Warning*" означає, що конструкція синтаксично правильна, але не відповідає де-

яким правилам мови, і це може послужити джерелом наступних помилок. Для усунення помилок потрібно визначити місце їхнього виникнення і проаналізувати ситуацію.

Для локалізації помилок краще використовувати інформацію зі спеціального файлу лістингу, що створюється транслятором. Цей файл має розширення *.lst*; його ім'я визначається відповідно до тих правил, що визначені вище. Лістинг – дуже важливий документ, і йому потрібно приділити належну увагу. Файл лістингу містить, зокрема, код асемблера вихідної програми. Але в лістингу приводиться розширена інформація про цей код. Для кожної команди асемблера вказуються її машинний (об'єктний) код і зсув (*offset*), або адреса у кодовому сегменті. Крім того, наприкінці лістингу *TASM* формує таблиці, що містять інформацію про мітки і сегменти, використані у програмі. Якщо є помилки або сумнівні ділянки коду, то *TASM* включає в кінець лістингу повідомлення про них.

Після того як усунуто помилки й одержано об'єктний модуль, можна приступати до наступного кроку – створення завантажувального модуля. Процес створення завантажувального модуля поділяється на два кроки – трансляцію і компонування (укладання або компіляція). Це зроблено для того, щоб можна було поєднувати разом кілька модулів (написаних на одному чи декількох мовах). Формат об'єктного файлу дозволяє, за певних умов, об'єднати кілька окремо трансльованих вихідних модулів в один модуль. При цьому у функції укладача (компілятора) входить дозвіл зовнішніх посилань (посилань на процедури і змінні) у цих модулях. Результатом роботи укладача (компілятора) є створення завантажувального файлу з розширенням *.exe*. Після цього операційна система може завантажити такий файл у пам'ять і виконати його.

Повний формат командного рядка для запуску укладача (компілятора) досить складний:

TLINK [опції] список_об'єктних_файлів
[,ім'я_завантажувального_модуля] [,ім'я_файлу_карти]
[,ім'я_файлу_бібліотеки] [,ім'я_файлу_визначень] [,ім'я ресурсного файлу]

Тут:

опції – необов'язкові параметри, що керують роботою укладача (компілятора). Список найчастіше використовуваних опцій приведе-

ний у довіднику [3]. Кожній опції повинен передувати один з наступних символів: "–" чи "/";

список_об'єктних_файлів – обов'язковий параметр, що містить список файлів, які компонуються, з розширенням .obj. Файли повинні бути розділені пропусками або знаком "+", наприклад: *tlink /v prog + mdf + fdr*. При необхідності імена файлів містять вказівку шляху до них;

ім'я сформованого завантажувального модуля – необов'язковий параметр, що позначає ім'я сформованого завантажувального модуля. Якщо воно не зазначено, то ім'я завантажувального модуля буде збігатися з першим зі списку імен об'єктних файлів;

ім'я_файлу_карти – необов'язковий параметр, наявність якого зобов'язує укладач (компілятор) створити спеціальний файл із картою завантаження. У ній перелічуються імена, адреси завантаження і розміри всіх сегментів, що входять у програму;

ім'я_файлу_бібліотеки – необов'язковий параметр, що являє собою шлях до файлу бібліотеки (.lib). Цей файл створюється й обслуговується спеціальною утилітою *tlib.exe* пакета *TASM*. Утиліта дозволяє об'єднати часто використовувані підпрограми у вигді об'єктних модулів в один файл. Надалі ви можете вказувати в командному рядку *tlink.exe* імена потрібних для компонування об'єктних модулів і файл бібліотеки, у якому варто шукати ці підпрограми. Якщо ви компонуєте *Windows* – додаток, то на місці параметра ім'я_файлу_бібліотеки вказується ім'я бібліотеки імпорту;

ім'я_файлу_визначень – необов'язковий параметр, що являє собою шлях до файлу визначень (*.def*). Цей файл використовується при компонуванні *Windows* – додатків (див. урок 18);

ім'я_ресурсного_файлу – необов'язковий параметр, що являє собою шлях до файлу з ресурсами *Windows* – додатка (*.res*). Цей файл використовується при компонуванні *Windows* – додатків.

Розглянутий нами формат командного рядка використовується і для 32-розрядного варіанта укладача (компілятора) *tlink32.exe*.

Так само як і для синтаксису *tasm.exe*, зовсім не обов'язково запам'ятовувати докладно синтаксис команди *tlink.exe*. Для того щоб одержати список опцій програми *tlink.exe*, досить просто запустити її без вказівки параметрів.

Для виконання даного прикладу запустимо програму *tlink.exe* командним рядком виду:

tlink.exe /v first.obj

У результаті одержимо завантажувальний модуль з розширенням *.exe – first.exe*.

Обов'язковим етапом процесу розробки є налагодження.

На етапі налагодження, використовуючи опис алгоритму, виконується контроль правильності функціонування як окремих ділянок коду, так і всієї програми в цілому. Але навіть успішне закінчення налагодження ще не є гарантією того, що програма буде працювати правильно з усіма можливими вихідними даними. Тому потрібно обов'язково провести тестування програми, тобто перевірити її роботу на "граничних" і свідомо некоректних вихідних даних.

Для локалізації логічних помилок у програмах використовують спеціальний тип програмного забезпечення – програмні налагоджувачі.

Налагоджувач *Turbo Debugger* (*td.exe*), розроблений фірмою *Borland International*, являє собою віконне середовище налагодження програм на рівні вихідного тексту на мовах *Pascal, C*, асемблер. Він дозволяє вирішити дві головні задачі: визначити місце логічної помилки; визначити причину логічної помилки.

Перелічимо деякі можливості *td.exe*:

виконання трасування програми в прямому напрямку, тобто послідовне виконання програми, при якому за один крок виконується одна машинна інструкція;

виконання трасування програми у зворотному напрямку, тобто виконання програми по одній команді, але в зворотному напрямку;

перегляд і зміну стану апаратних ресурсів мікропроцесора під час покомандного виконання програми.

Це дозволяє визначити місце і джерело помилки в програмі.

Примітки:

1. Застосування опції */zi* дозволяє транслятору зберегти зв'язок символічних імен у програмі і їхніх зсувах у сегменті коду, що дозволить налагоджувачу вести налагодження, використовуючи оригінальні імена.

2. Створення об'єктного модуля повинне бути здійснене з опцією */v*:

tlink /v ім'я_об'єктного_модуля.

Опція */v* вказує на необхідність збереження інформації для налагодження у файлі, що виконується.

3. Запуск налагоджувача зручніше робити з командного рядка з вказівкою модуля програми, що виконується та підлягає налагодженню: *td* ім'я_ завантажувальний_модуль.

Лістинг 1

```
data      segment     para public 'data'  ;сегмент даних
message   db          'Привет!',13,10,'$'
data      ends
code      segment     para public 'code'  ; початок сегмента коду
main      proc        ; початок процедури main
mov ax, data
mov ds, ax            ; встановити регістр DS таким чином, щоб
                      ; він вказував на сегмент даних
mov ah, 9             ; функція DOS виведення рядка
mov dx,OFFSET Message; посилання на повідомлення "Привет!"
int 21h               ; вивести "Привет!" на екран
mov ah, 4ch           ; функція DOS закінчення програми
int 21h               ; виклик DOS
main      endp        ; кінець процедури main
code      ends        ; кінець сегмента коду
end       main        ; кінець програми с точкою входу main
```

Для виконання операцій введення/виводу в асемблері використовуються функції переривань *BIOS* і *DOS*. Інформацію про них можна одержати з довідника *ng.exe* [3], що активізується комбінацією клавіш [*Shift*]+[*F*1]. (Меню – Системний сервіс – *Int* 21*h* (Функції *DOS*)).

Для введення символів із клавіатури використовуються функції 1*h*, 10*h* переривання *DOS* (*Int* 21h), для виведення символів на екран монітора – 2*h*, 9*h* (*Int* 21*h*), для завершення програми – 4*ch* (*Int* 21*h*).

Порядок виконання роботи

1. Створити робочий каталог.

2. Скопіювати в робочий каталог програму *first.exe*.

3. За допомогою транслятора *tasm.exe* створити об'єктний файл.

4. За допомогою укладача (компілятора) *tlink.exe* створити завантажувальний модуль.

5. Відкрити завантажувальний модуль у налагоджувачу *td.exe*. Виконувати програму покроково і спостерігати за станом регістрів мікропроцесора.

6. Виконати завдання приведені в таблиці 1 за варіантом.

7. Оформити звіт, у якому навести послідовність виконаних службових команд для створення завантажувального модуля, лістинг програми за пунктом 6 і відповіді запитання.

Таблиця 1

Номер варіанту	I : вивести на екран	II : вивести на екран рядок з символів
1	Своє ПІБ та дату народження	☺☻♣○♫☼►◄▒▓─┼╞╪╛▌
2	Свою повну домашню адресу та телефон	♥♦♣↨↑↓→← ┴┬ ╚╔ ┌▄▟ ■
3	Повну назву місця свого народження	☺♣○→←↔▲▼ ▒▓╢ ╟╖¤■
4	Розшифровку абревіатури ІІПТ НАНУ	☻♥○☼►↑↓→←∟↔▼ ┌▉▌
5	Розшифровку абревіатури EEPROM	☺ ☻♥♦♣♫☼►↓↔▲▼ ▒▓
6	Розшифровку абревіатури ІАЕ НУК	♫☼►◄↕↔▲▒╝╝╨╤╥ ╙ ■
7	Розшифровку абревіатури РВУ МВС	☻♥♣☼◄↕§→←▼ ▒╟╖╝╝╛
8	Свою повну домашню адресу та телефон	☺♣○↨↑▼ ▒┴┼═╠╥╪╛ ┌
9	Своє ПІБ та дату народження	♦♣♫►§↑↓→↔▼ ▒▓┐ ╪┘ ■
10	Розшифровку абревіатури НУК	◄↕↨↑↓→▲ ▒▓▓┐ └┴┼ ▛√
11	Розшифровку абревіатури ТОЕ, ТАУ	☺ ☻♥♦♣♫☼←∟ ╝╝ ╞╟╧╛
12	Повну назву місця свого навчання	▒▓♦♣♫►→←▼ ╢╨ ╒╫▉
13	Розшифровку абревіатури ІІПТ НАНУ	☻♦♣♫☼↓→←∟ ▼ ▓▕ ╨ ┌▄
14	Розшифровку абревіатури ІАЕ НУК	☺ ☻♥♫☼►▬↨↑↔▼ ┌▉√№
15	Своє ПІБ та дату народження	♥♦♣○♫☼↓→←↔▲▼ ▒┼╦
16	Розшифровку абревіатури РВУ МВС	☺ ☻♣♫↕¶§→←▼ ▒╦╟╫╫
17	Повну назву місця свого навчання	♣♫☼↨↑↓→←▼▓ └┴├ ╔╔╫
18	Свою повну домашню адресу та телефон	☺♥♦♣♫☼►↕↔▼┼╞╦√№
19	Розшифровку абревіатури EEPROM	☻♥♦◄↕↨↔▼▒╟╖╝╟ └■

Контрольні запитання

1. Призначення транслятора *tasm.exe.*
2. Який ключ транслятора дозволяє включити в завантажувальний модуль інформацію для налагодження?
3. Призначення укладача (компілятора) *tlink.exe.*
4. Який ключ укладача (компілятора) *tlink.exe* дозволяє включити в завантажувальний модуль інформацію для налагодження?

5. Які основні розділи меню налагоджувача *td.exe*, їх призначення?

6. Як відкрити вікно *CPU* при роботі з *td.exe*?

7. Які функції переривання *DOS* використовуються для введення символів з клавіатури?

8. Які функції переривання *DOS* використовуються для виведення інформації на монітор?

Практична робота № 2. Кодування процесора в реальному режимі. Команди пересилання даних. Арифметичні операції: додавання, множення, ділення. Структурне програмування

Мета роботи: засвоїти методику кодування процесора в реальному режимі. Навчитися застосовувати команди пересилання даних, арифметичні операції: додавання, множення, ділення. Звертання до процедур.

Основні теоретичні відомості

Програма на асемблері при програмуванні в реальному режимі відображає організацію фізичної оперативної пам'яті і тому складається зі спеціалізованих блоків, які називаються сегментами. Для опису цих блоків використовуються директиви сегментації, інформацію про які наведено у [4, 5] і інтерактивному довіднику [2] з асемблеру (Структура програми на асемблері – Стандартні директиви сегментації).

Система команд мікропроцесора містить більш 300 машинних команд. З появою кожної нової моделі мікропроцесора їхня кількість, як правило, зростає, відбиваючи тим самим архітектурні нововведення, що відрізняють дану модель від попередніх. Набір машинних команд можна структурувати по групах: команди пересилання даних, арифметичні команди, логічні команди, команди передачі керування, ланцюгові команди.

Вичерпну інформацію про команди можна одержати з інтерактивного довідника [2] (Опис команд – Опис команд мікропроцесора, упорядкованих по функціональній ознаці) або з [5].

Мова асемблеру має програмні засоби підтримки концепції структурного програмування. Для оформлення процедур як окремих об'єктів існують спеціальні процедури PROC/ENDP і машинна команда ret. Ці засоби докладно розглянуто у розділах 10 і 14 підручника [5] і у довіднику [3]. При роботі з процедурами велике значення приділяється організації керуючих і інформаційних зв'язків між структурними одиницями програми (модулями), що спільно розв'язують одну або декілька великих задач. Розробка всього сучасного програмного забезпечення

виконується відповідно до концепції структурного і модульного програмування. Одним з самих поширених способів передачі даних у підпрограмі є використання стеку, для роботи з яким використовуються команди *push* і *pop* [2] та регістри *sp, bp* [2] (Програмна модель мікропроцесора – шістнадцять регістрів користувача – Регістри загального призначення).

Порядок виконання роботи

1. Розробити програму обчислення заданої функції за варіантом (табл. 2) з використанням стандартних процедур сегментації для *I*8086. Вхідні дані ввести з клавіатури. Результат вивести на екран монітору. В програмі повинно здійснюватися звертання до *n* підпрограм (за варіантом), вкладених одна в одну, тобто звертання до чергової підпрограми здійснюється з попередньої програмної одиниці. Підпрограми умовні, отже крім команд *CALL* і *RET* достатньо в підпрограмах при необхідності використати команду *NOP* (нічого не робити). Передбачити "ініціалізацію стеку".

2. Виконати пункти 1...5 з першої практичної роботи.

3. Оформити звіт, у якому навести лістинг розробленої програми.

Таблиця 2

Номер варіанту	*n*	Функція	Номер варіанту	*n*	Функція
1	15	*(c+d)*(a+b)*	16	15	*(c–d)/(a–b)*
2	14	*c/d+(a+b)*	17	14	*c/(a–b)*
3	13	*c/d–(a+b)*	18	13	*c*d+(a–b)*
4	12	*(c–d)*(a+b)*	19	12	*c*d–(a–b)*
5	11	*(c+d)/(a+b)*	20	11	*c*a*b*
6	10	*(c–d)/(a+b)*	21	10	*(c+d)*a*b*
7	9	*c/(a+b)*	22	9	*c/d+a*b*
8	8	*c*d+(a+b)*	23	8	*c/d–a*b*
9	7	*c*d–(a+b)*	24	7	*(c–d)*a*b*
10	6	*c*(a–b)*	25	6	*(c+d)/a*b*
11	5	*(c+d)*(a–b)*	26	5	*(c–d)/a*b*
12	4	*c/d+(a–b)*	27	4	*c/a*b*
13	3	*c/d–(a–b)*	28	3	*c*d+a*b*
14	2	*(c–d)*(a–b)*	29	2	*c*d–a*b*
15	1	*(c+d)/(a–b)*	30	1	*c*(a+b)*

Контрольні запитання

1. Призначення директив сегментації.

2. Чим відрізняються стандартні і спрощені директиви сегментації?

3. Які регістри використовуються для виконання команд додавання?

4. Які регістри використовуються для виконання команд множення?

5. Які регістри використовуються для виконання команд ділення?

6. Пояснити роль регістрів *SP* – "вказівник стеку" при роботі з процедурами.

7. Перелічити і пояснити призначення сегментних регістрів при виконанні програм реального режиму.

8. Перелічити і пояснити правила використання регістрів загального призначення.

Практична робота № 3. Кодування процесора в реальному режимі для обробки масивів за допомогою циклів

Мета роботи: засвоїти методи кодування процесора в реальному режимі для обробки масивів за допомогою операторів безумовного і умовного переходів і операторів циклу.

Основні теоретичні відомості

Для опису простих типів даних у програмі на асемблері використовуються спеціальні директиви резервування й ініціалізації даних, що, по суті, є вказівками транслятору на виділення для змінної з заданим ім'ям визначеного обсягу оперативної пам'яті. Якщо проводити аналогію з мовами високого рівня, то директиви резервування й ініціалізації даних є визначеннями змінних. Відрізняються вони тим, що для одної змінної (під одним ім'ям) може бути ініціалізовано декілька значень одного типу, розміщених у пам'яті послідовно, тобто для них резервується потрібний обсяг пам'яті. Наприклад, об'явити просту змінну *a* типу *byte* і масив *mas* із 10 елементів типу *byte* можна за допомогою директиви *db* (визначити *byte*):

a db ?

mas db 10 *dup* (?)

За допомогою цих директив можна визначати змінні простих типів, покажчики і змінні структурованих типів, наприклад, масиви або

структури. Докладна інформація про директиви резервування й ініціалізації даних наведена у інтерактивному довіднику з асемблеру [2] (Інтерактивний довідник з асемблеру – Типи даних).

Масиви у асемблері задаються як змінні простого типу. Доступ до елементів масиву здійснюється за допомогою індексного оператора [операнд], у якості операнду може використовуватись один з індексних регістрів (*si, di*). Запис *z*[*si*] значить, що треба прочитати значення, яке знаходиться у оперативній пам'яті за адресою, що дорівнює сумі зміщення змінної *z* і *si*·(тип_даних_змінної_*z*). Докладна інформація про роботу з масивами наведена у довіднику [2] (Типи даних – Данні складного типу).

Одною з основних конструкцій алгоритмічної мови будь-якого рівня є цикл. У асемблері передбачено оператори циклу, аналогічні за алгоритмом роботи циклам *FOR* мов високого рівня. Початкове значення лічильника циклу треба помістити у регістр *cx*, перед початком циклу поставити мітку (наприклад, *m*1:), наприкінці тіла циклу розмістити оператор циклу (наприклад, *loop m*1). При кожному проходженні оператора *loop*, лічильник циклу у регістрі *cx* буде зменшуватись на 1, при цьому, якщо *cx*≠0, то керування буде передаватися на мітку *m*1, у іншому випадку відбудеться вихід з циклу і керування буде передано на оператор, що стоїть після оператору циклу. Докладна інформація про оператори циклу наведена у інтерактивному довіднику [2] (Опис команд – *LOOP* у таблиці).

Цикли також можуть бути організовані за допомогою операторів безумовного – *jmp* мітка1 (перехід на мітка1) – і умовного переходу. Для здійснення умовних переходів спочатку потрібно порівняти два операнди за допомогою команди

cmp операнд1, операнд2 (порівняти оператор1 і оператор2).

За результатами порівняння всановлюються прапори – окремі біти у регістрі прапорів [2] (Програмна модель мікропроцесора – 16 регістрів користувача – Регістри стану і керування). У відповідності до комбінацій прапорів може бути виконано перехід за однією з команд умовного переходу [2] (Опис команд – Опис команд, упорядкованих за функціональною ознакою – Умовні), які починаються з букви *j* (*jump*). Інші букви відповідають першим буквам англійських слів. Наприклад, у слові дорівнює (*equal*) – перша буква е, тому команда

je мітка2

означає перехід на мітка2, якщо операнд1 дорівнює операнд2.

Аналогічно *jg* мітка3 (перехід на мітка3, якщо операнд1 більше операнд2), *jl* мітка4 (перехід на мітка4, якщо операнд1 менше операнд2). Докладна інформація про оператори безумовного і умовного переходів наведена у довіднику [2] (Опис команд – Опис команд, упорядкованих за функціональною ознакою – Умовні).

Порядок виконання роботи

1. Розробити програму обчислення суми чисел (за варіантом табл. 3). Результат вивести на екран монітору.

2. Побудувати циклічний алгоритм і скласти програму в командах МП *I*8086 (асемблер) створення в регістрі *xx* (акумуляторі) лічильника за модулем *n* з кроком *m* (в кожному циклі вміст регістра *xx* збільшується на *m* до максимального значення *n*). В програмі необхідно передбачити початкову очистку акумулятора і запис кожного нового стану лічильника у вічко ОЗП, адреса якого на одиницю більша попереднього. Початкову адресу взяти довільно (використовувати команди умовного переходу *je, …*).

3. Виконати пункти 1...5 з першої практичної роботи.

4. Оформити звіт, у якому навести додатково блок-схеми алгоритмів.

Таблиця 3

Номер варіанту	Сума чисел	*xx*	*n*	*m*
1	тільки непарних от 1 до 15	*ax*	256	1
2	за винятком 2,7,10 от 1 до 15	*bx*	256	2
3	за винятком 3,8,11 от 1 до 15	*dx*	256	3
4	за винятком 1,9,12 от 1 до 15	*cx*	256	4
5	за винятком 2,7,13 от 1 до 15	*ax*	128	4
6	за винятком 4,7,14 от 1 до 15	*bx*	128	3
7	за винятком 5,9,15 от 1 до 15	*dx*	128	2
8	за винятком 6,7,11 от 1 до 15	*cx*	128	1
9	за винятком 2,4,12 от 1 до 15	*ax*	64	2
10	тільки парних от 0 до 16	*bx*	64	3
11	тільки непарних от 0 до 16	*dx*	64	4
12	за винятком 2,7,10 от 0 до 16	*cx*	64	1
13	за винятком 3,8,11 от 0 до 16	*ax*	100	3

Продовж. табл. 3.

Номер варіанту	Сума чисел	*xx*	*n*	*m*
13	за винятком 3,8,11 от 0 до 16	*ax*	100	3
14	за винятком 1,9,12 от 0 до 16	*bx*	100	4
15	за винятком 2,7,13 от 0 до 16	*dx*	100	1
16	за винятком 4,7,14 от 0 до 16	*ax*	256	1
17	за винятком 5,9,15 от 0 до 16	*bx*	256	2
18	за винятком 6,7,11 от 0 до 16	*dx*	256	3
19	за винятком 2,4,12 от 0 до 16	*cx*	256	4
20	тільки парних от 2 до 17	*ax*	128	4
21	тільки непарних от 2 до 17	*bx*	128	3
22	за винятком 2,7,10 от 2 до 17	*dx*	128	2
23	за винятком 3,8,11 от 2 до 17	*cx*	128	1
24	за винятком 1,9,12 от 2 до 17	*ax*	64	2
25	за винятком 2,7,13 от 2 до 17	*bx*	64	3
26	за винятком 4,7,14 от 2 до 17	*dx*	64	4
27	за винятком 5,9,15 от 2 до 17	*cx*	64	1
28	за винятком 6,7,11 от 2 до 17	*ax*	100	3
29	за винятком 2,4,12 от 2 до 17	*bx*	100	4
30	тільки парних от 1 до 15	*dx*	100	1

Контрольні запитання

1. Як задаються змінні у програмі на асемблері?
2. Як задати змінну типу масив?
3. Як організувати доступ до елементів масиву?
4. Які оператори циклу використовуються у асемблері? Принцип дії.
5. Які оператори переходу використовуються у асемблері? Принцип дії.

2. СТАНДАРТНІ ІНТЕРФЕЙСИ ОБМІНУ ДАНИМИ ПРОМИСЛОВИХ КОМП'ЮТЕРІВ

Практична робота № 4. Пересилання даних в порти введення/виводу за інтерфейсом *Centronics* в реальному режимі

Мета роботи: засвоїти методи програмування пересилання даних в порти введення/виводу за інтерфейсом *Centronics*.

Робота з паралельним портом

Докладно інформація про роботу з паралельним портом викладена у [1]. *BIOS* може працювати з трьома рівнозначними принтерними портами. У процесі тестування й ініціалізації системи *BIOS* знаходить працездатні принтерні порти і записує їх базові адреси в таблицю. Таблиця адрес розташовується в області даних *BIOS* за адресою 0000:0408*h*. Можливі наступні значення базових адрес:

378*h* – принтерний порт *LPT*1;

278*h* – принтерний порт *LPT*2;

3*BCh* – принтерний порт на платі адаптера монохромного дисплея.

Принтерні порти можуть продукувати запити на переривання:

*LPT*1 – *IRQ*7, *INT* 0*Fh*;

*LPT*2 – *IRQ*5, *INT* 0*Dh*.

Кожен принтерний порт (принтерний адаптер) обслуговує кілька портів введення/виведення. Розглянемо їх призначення.

Порт 378*h*

Цей порт призначений для запису виведеного на принтер байта даних. Можливо також читання тільки що записаного байта.

Порт 37A*h*

Порт керування принтером, доступний для читання і запису має значення біт, які наведено у табл. 4.

Таблиця 4

Номер біту								Значення біту
7	6	5	4	3	2	1	0	*L* – низький рівень, *H* – високий рівень
							H	строб даних, приймає значення 1 при виведенні байта
						H		автоматичний перевід рядка після символу "повернення каретки" *CR*
					H			очистка принтера, активний рівень – 0
				H				вибір принтера для роботи
			H					дозвіл переривання від принтера
L	*L*	*L*						не використовуються (встановлені в 0)

Якщо переривання від принтера дозволені, вони продукуються, коли сигнал готовності принтера ACK (контакт 10) приймає рівень логічного 0.

Порт 379*h*

Порт стану принтера, доступний тільки для читання, має значення біт, які наведено у табл. 5.

Таблиця 5

Номер біту								Значення біту
7	6	5	4	3	2	1	0	*L* – низький рівень, *H* – високий рівень
					L		*L*	Не використовуються (встановлені в 0)
						H		Таймаут, занадто велика затримка при виконанні операції друку, можливо, що принтер несправний
				L				Сигнал помилки, активний рівень – 0
			H					1 – принтер обраний для роботи, 0 – принтер у стані *offline*
		H						папір закінчився
	L							готовність принтера, активний рівень – 0
H								1 – принтер готовий, 0 – принтер зайнятий, знаходиться в стані *offline* чи відбулася помилка

Для роботи з портами введення/виводу використовуються команди *in, out* процесора [5] розділ 7 або [1, 2, 4].

Для запуску друку символу потрібно на короткий час установити біт 0 регістра керування, а потім скинути його.

Переривання відбувається по закінченні виводу символу на друк для першого принтера на сьомому рівні контролера переривань (*IRQ*7, вектор переривання 0*Fh*), для другого принтера - на п'ятому рівні (*IRQ*5, вектор 0*Dh*). Слід зазначити також, що *IRQ*5 використовується *XT*-контролером твердих дисків для генерації своїх переривань. Зазвичай цей біт не використовується (скидається), а перевірка готовності принтера виконується на підставі опитування регістра стану.

Типова послідовність дій для виводу на друк одного символу наступна:

вивести переданий байт у регістр даних;

у циклі перевіряти стан принтера до установки біта 7 регістра стану (тут можливе використання таймаута);

перевірити біти 3-5 регістра стану на наявність помилки;

установити і відразу ж скинути нульовий біт регістра керування, для цього підходить наступна послідовність команд:

```
mov dx, 37Ah          ; адреса регістра керування
mov al, 00001101b     ; установити біти 0, 2 і 3
out dx,al             ; вивести команду
xor al, 1             ; скинути біт 0
out dx, al            ; повторно вивести команду.
```

Звичайно рідко приходиться працювати з принтером на рівні портів введення/виводу, тому що досить використання функцій *BIOS* або *MS DOS*, що призначені для цього. Приведена вище інформація може знадобитися для розробки власного драйвера чи принтера для підключення до принтерного порту якого-небудь іншого пристрою введення/виводу, наприклад, аналого-цифрового перетворювача.

BIOS використовує для роботи з принтером функції 0, 1, 2 переривання *INT* 17*h* [3].

Функція 00*h* призначена для друку одного символу:

На вході:

AH = 00*h*;

AL = *ASCII*-код символу для друку;

DX = номер принтера: 0, 1 чи 2.

На виході: *AH* = слово стану принтера (див. нижче).

Ця функція виводить на принтер один символ, заданий у регістрі *AL*. У регістр *DX* необхідно записати номер принтера, що використовується, для *LPT*1 це 0, для *LPT*2 – 1 і т.д.

Після виконання переривання регістр *AH* буде містити слово стану, що має формат, наведений у табл.2.

Викликавши функцію 0 переривання *INT* 17*h*, програма повинна перевірити окремі біти слова стану і переконатися в тому, що висновок байта відбувся без помилок. Найчастіше оператор забуває перевести принтер у стан *ONLINE*, або вставити папір, або взагалі включити принтер. У цьому випадку доцільно нагадати оператору про необхідність виконання цих дій і потім повторити друк символу.

Якщо принтер несправний, програма повинна надати оператору

можливість скасувати друк тексту. Нижче наведено приклад програми, що виконує друк тексту і аналізує помилки, що можуть виникнути в процесі друку.

Зверніть увагу на біт 1 байта стану – таймаут. Якщо принтер знаходиться в стані *OFFLINE*, функція 0 переривання *INT* 17*h* очікує деякий час готовності принтера, після чого, якщо принтер так і не перейшов у стан готовності, встановлює біт 1 у байті стану. Область даних *BIOS* за адресою 0000*h*:0478*h* містить чотири байти, що використовуються як лічильники часу при очікуванні готовності принтера.

Переривання *INT* 17*h* має ще дві функції, що виконують ініціалізацію принтера і повертають поточний стан принтера.

Функція 01*h* ініціалізує принтер:

На вході: *AH* = 01*h*;

DX = номер принтера: 0, 1 чи 2.

На виході: *AH* = слово стану принтера.

Ця функція виконує апаратну очистку принтера. Якщо ви завантажили в принтер який-небудь шрифт (наприклад, кирилицю), після очистки принтера завантаження шрифту прийдеться виконувати заново. Тому не слід виконувати очистку принтера, якщо це дійсно не потрібно. Завичай принтер необхідно очищати або перед налагодженням його на заданий режим роботи, що виконується один раз, або при зміні цього режиму.

Слово стану принтера може бути отримане за допомогою функції 02*h*:

На вході: *AH* = 02*h*;

DX = номер принтера: 0, 1 чи 2.

На виході: *AH* = слово стану принтера.

Цю функцію зручно використовувати перед початком друкування для визначення готовності принтера до роботи.

Порядок виконання роботи

1. До розрядів *B*0...B7 *LPT* – порту крізь схему інтерфейсу під'єднані розташовані в лінію світлодіоди, з'єднані за схемою з загальним катодом, таким чином, що при видачі через *i*-ий розряд "1" – засвічується *i*-ий світлодіод. Для передачі логічного рівня напруги на кожний світлодіод у схемі інтерфейсу необхідно установити і скинути нульовий біт регістра керування (як для керування друком символу). Скласти програму на асемблері, що по черзі засвічує першу комбіна-

цію світлодіодів, потім другу і третю і програма знову повторюється (табл. 6). За допомогою такої програми досягається режим "вогні, що біжать".

2. Час "світіння" світлодіодів повинен визначатись тимчасовою затримкою, що реалізується за допомогою *n* вкладених лічильників (см. табл. 4) і команд організації циклів та використання стеку (*loop, pop, push*). Показати, як можна в широких межах змінити час затримки.

Таблиця 6

Номер варіанту	Комбінація			
	перша	друга	третя	n
1.	0, 3, 6	1, 4, 7	2, 5	3
2.	1, 2, 3	4, 5, 7	6, 0	4
3.	0, 7 ,6	5, 4, 3	2, 1	5
4.	1, 3, 5	7, 2, 4,	6, 0	6
5.	0, 6, 4	2, 7, 5	3, 1	7
6.	1, 4, 7	2, 5, 0,	3, 6	8
7.	6, 3, 0	5, 2, 7	4, 1	7
8.	1, 5, 1	2, 6, 2	3, 7, 3	6
9.	7, 3, 7	6, 2, 6	5, 1, 5	5
10.	4, 0, 4	5, 1, 5	6, 2, 6	4
11.	2, 6, 2	1, 5, 1	0, 4, 0	3
12.	5, 2, 7	4, 1, 6	3, 0	4
13.	1, 6, 3	0, 5, 2	7, 4	5
14.	4, 7, 2	5, 0, 3	6, 1	6
15.	0, 6, 4	2, 7, 5	3, 1	7

Контрольні запитання

1. Як за допомогою вкладених пустих циклів керувати тривалістю часової затримки у широких межах?

2. Для чого при використанні вкладених циклів типу *loop* використовується стек?

3. За якою моделлю організовано стек?

4. Які регістри в арифметико-логічному пристрої мікропроцесора використовуються для організації роботи зі стеком і їх призначення.

5. Які адреси мають регістри *LPT* – порту у просторі введення/виводу у *Intel*- сумісному комп'ютері?

6. Що містить поняття інтерфейс *Centronics*?

7. Привести номера і призначення контактів *DB*-25S *LPT* - порта.

8. Описати фізичний і електричний інтерфейс *LPT* - порта.

9. Які переривання і функції *BIOS* або *MS DOS* призначені для роботи з *LPT* - портом.

Практична робота № 5. Програмування *LPT* в захищеному режимі

Мета роботи: вивчення методів роботи з *LPT* в *MS Windows 98/NT* і створення програми автоматизованого робочого місця (додатка *MS Windows 98/NT*).

Основні теоретичні відомості

Для роботи з портами під керуванням *MS Windows 98/NT* необхідно враховувати те, що операційна система перехоплює виклики переривань *BIOS* або *DOS* при роботі у режимі віртуального *I*8086 і використовує власні підпрограми обробки переривань [1, 4, 5]. Тому необхідно використовувати *API*–функції *Windows* або розробити спеціальні програми – драйвери для доступу до портів або використовувати готові драйвери і бібліотеки, наприклад, *DlPortIO.SYS* та *DlPortIO.DLL*.

Для розробки програми автоматизованого робочого місця для роботи з портами необхідно керуватися основними принципами розробки додатка *MS Windows 98/NT*: забезпечення безвідмовної роботи під керуванням операційної системи, базування на принципах об'єктно-модульного програмування, використання стандартних програмних засобів обробки інформації (таких як *API*–функції *Windows* і динамічні бібліотеки *.DLL*), забезпечення інтуїтивно зрозумілого інтерфейсу користувача. Приклад форми програми для керування портами наведено на рис. 2.

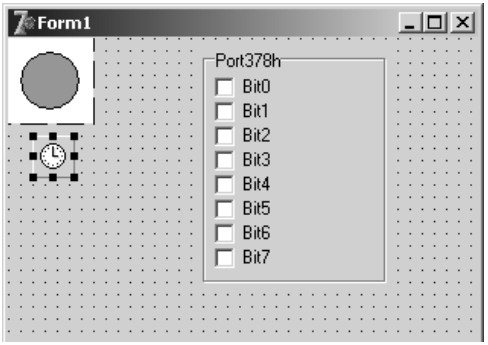

Рис. 2

Найпростіший спосіб обміну інформацією між програмами – це використання бібліотек *.DLL*. Насправді, біб-

ліотека *.DLL* – це не програма, а сховище програмного коду (наприклад функцій) і ресурсів (наприклад форм). Вона підключається до програми динамічно, після того як програма запущена. Використовувати бібліотеки *.DLL* дуже зручно. Нехай, наприклад, задана програма прогнозування деякого значення на підставі даних, що вводяться користувачем. Створимо інтерфейс користувача, а сам алгоритм розрахунку прогнозу (досить складний) виділимо в окрему функцію і помістимо в бібліотеку *.DLL*, що підключається до програми. Тепер, якщо розроблювач поліпшить алгоритм прогнозування, то користувачам досить поміняти тільки бібліотеку *.DLL*, а вносити зміни в код, відповідальний за інтерфейс, взагалі не прийдеться.

Наприклад, саме в *DLL* зберігаються всі процедури і функції *API Windows*, які дуже часто застосовуються безпосередньо або через аналогічні функції *C++Builder* або *Delphi*, що інкапсулюють ті чи інші функції *API Windows*. У *DLL* зберігаються також усі стандартні діалоги *Windows*, що використовуються ледве не в будь-якому додатку.

Для використання функцій роботи з паралельним портом у своєму додатку необхідно описати їх у модулі як показано в прикладі:

implementation

procedure DlPortWritePortUchar(Port:longword; Value:byte); stdcall; External 'dlportio.dll' name 'DlPortWritePortUchar';

Function DlPortReadPortUchar(Port:longword):byte; stdcall; External 'dlportio.dll' name 'DlPortReadPortUchar';

Наведений код повідомляє компілятор, що функції *DlPortWritePortUchar(Port:longword;Value:byte)* і *DlPortReadPortUchar(Port:longword):byte* є зовнішніми і знаходяться у файлі *'dlportio.dll'* у тому самому каталозі, що й розроблений Вами додаток.

Далі звертання до цих підпрограм здійснюється як виклик звичайних процедур, або функцій з тексту модуля коду.

У бібліотеці *DlPortIO.DLL* знаходяться наступні функції для роботи з портами:

UCHAR DLPORT_API DlPortReadPortUchar(IN ULONG Port); //зчитує з порта за адресою *Port* один байт.

USHORT DLPORT_API DlPortReadPortUshort(IN ULONG Port); //зчитує з порта за адресою *Port* два байти.

ULONG DLPORT_API DlPortReadPortUlong(IN ULONG Port); //зчітує з порта за адресою *Port* чотири байти.

VOID DLPORT_API DlPortReadPortBufferUchar(IN ULONG Port, IN PUCHAR Buffer, IN ULONG Count); //зчитує з порта за адресою *Port* по одному байту, записує їх у масив символів *Buffer*, кількість символів записує у змінну *Count*.

VOID DLPORT_API DlPortReadPortBufferUshort(IN ULONG Port, IN PUSHORT Buffer, IN ULONG Count); // теж саме але для даних розміром 2 байти.

VOID DLPORT_API DlPortReadPortBufferUlong(IN ULONG Port, IN PULONG Buffer, IN ULONG Count); // теж саме але для даних розміром 4 байти.

Далі описано функції для запису у порт даних, відповідних до наведених вище.

VOID DLPORT_API DlPortWritePortUchar(IN ULONG Port, IN UCHAR Value);

VOID DLPORT_API DlPortWritePortUshort(IN ULONG Port, IN USHORT Value);

VOID DLPORT_API DlPortWritePortUlong(IN ULONG Port, IN ULONG Value);

VOID DLPORT_API DlPortWritePortBufferUchar(IN ULONG Port, IN PUCHAR Buffer, IN ULONG Count);

VOID DLPORT_API DlPortWritePortBufferUshort(IN ULONG Port, IN PUSHORT Buffer, IN ULONG Count);

VOID DLPORT_API DlPortWritePortBufferUlong(IN ULONG Port, IN PULONG Buffer, IN ULONG Count).

Порядок виконання роботи

1. Розробити на алгоритмічній мові високого рівня (наприклад *Delphi*) додаток *MS Windows 98/NT*, який за допомогою елементів *CheckBox*, розташованих на формі, здійснює пересилання байту даних у порт 378*h LPT*-порта.

2. До розрядів *B*0...*B*7 *LPT*–порту схеми інтерфейсу під'єднані світлодіоди семисегментного індикатору так, що при видачі через *i*-ий розряд "1" – засвічується *i*-ий світлодіод (рис. 3). Скласти програму послідовного виводу на індикатор стилізованих букв або цифр заданого слова (табл. 7). Час "світіння" кожної букви постійний і повинен визначатись тимчасовою затримкою, що реалізується підпрограмою.

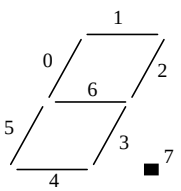

Рис. 3

Таблиця 7

Номер варіанту	Слово для передачі у порт	Номер варіанту	Слово для передачі у порт
1.	НАПРУГА	2.	ПРОГ
3.	ОПІР	4.	РІГ
5.	ПАПІР	6.	ГОРА
7.	СЛОВО	8.	НАПОР
9.	УПОР	10.	РОБО
11.	ЧАС	12. ...30	номер домашнього телефону

Контрольні запитання

1. Назвіть основні відмінності реального і захищеного режимів мікропроцесора.

2. Які засоби потрібно використовувати для доступу до регістрів *LPT* - порту в захищеному режимі і чому?

3. Що таке бібліотека *DLL* і як викликати бібліотечні функції з основної програми?

ПЕРЕЛІК ДЖЕРЕЛ

1. Гук М. Аппаратные средства IBM PC. Энциклопедия. – СПб.: Питер, 2001. – 816 с.

2. Електронний довідник з асемблеру. (Юров В.)

3. Електронний довідник ng.exe.

4. Конспект лекцій по дисципліні.

5. Юров В. Assembler. – СПб.: Питер, 2001. – 624 с.

ЗМІСТ

Навчальне видання

НАЗАРОВА Наталя Станіславівна
ВІННИЧЕНКО Дмитро Валерійович
КОЗИРЄВ Сергій Сергійович

МЕТОДИЧНІ ВКАЗІВКИ
до практичних робіт з курсу
"Інформаційно-керуючі системи та комплекси"

Частина 1

(*українською мовою*)

Комп'ютерна верстка *В.Г. Мазанко*
Коректор *М.О. Паненко*

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
ДК № 2506 від 25.05.2006 р.

Підписано до друку 07.04.09. Папір офсетний. Формат 60×84/16.
Друк офсетний. Гарнітура "Таймс". Ум. друк. арк. 1,7. Обл.-вид. арк. 1,8.
Тираж 100 прим. Вид. 14. Зам. № 155. Ціна договірна

Видавець і виготівник Національний університет кораблебудування,
54002, м. Миколаїв, вул. Скороходова, 5