Національний технічний університет України
«Київський політехнічний інститут»
Факультет біомедичної інженерії

В.В. Шликов, О.Г.Кисельова, А.О. Матвійчук

МІКРОПРОЦЕСОРИ

Методичні вказівки
до виконання лабораторних робіт

для студентів напрямів підготовки
6.051402 «Біомедична інженерія»
6.051003 «Приладобудування»

Затверджено Вченою радою
факультету біомедичної інженерії

Київ - 2014

Гриф надано Вченою радою ФБМІ

Протокол № 2 від 27.10.2014 р.

Рецензент:

А.О. Попов, к.техн., доц.
Національний технічний університет України
«Київський політехнічний інститут»

Відповідальний редактор:

В.І. Зубчук, к.техн.н., доцент
Національний технічний університет України
«Київський політехнічний інститут»

Шликов В.В., Кисельова О.Г., Матвійчук А.О.

«Мікропроцесори». Методичні вказівки до виконання лабораторних робіт для студентів напрямів підготовки 6.051402 «Біомедична інженерія», 6.051003 «Приладобудування» / Автори: В.В. Шликов, О.Г. Кисельова, А.О. Матвійчук – К.: НТУУ «КПІ», 2014. –123 с.

Зміст

ЛАБОРАТОРНА РОБОТА № 1

ОЗНАЙОМЛЕННЯ З РОБОТОЮ ЕМУЛЯТОРА EMU8086

Мета роботи: ознайомлення зі структурою навчальної мікро-ЕОМ (емулятора Emu8086), органами керування й режимами її роботи.

1 Короткі теоретичні відомості

Програмне середовище Emu8086 сполучає в собі потужний редактор вихідного коду, асемблер, дезасемблер, програмний емулятор (віртуальний ПК) з відладчиком і поетапне навчання.

1.1 Структура програмного середовища емулятора EMU8086

Початок роботи в програмному середовищі Emu8086. Необхідно запустити програму *Emu8086*, вибравши її значок у меню "Пуск", або безпосередньо запустити додаток Emu8086.exe (мал. 1.1).

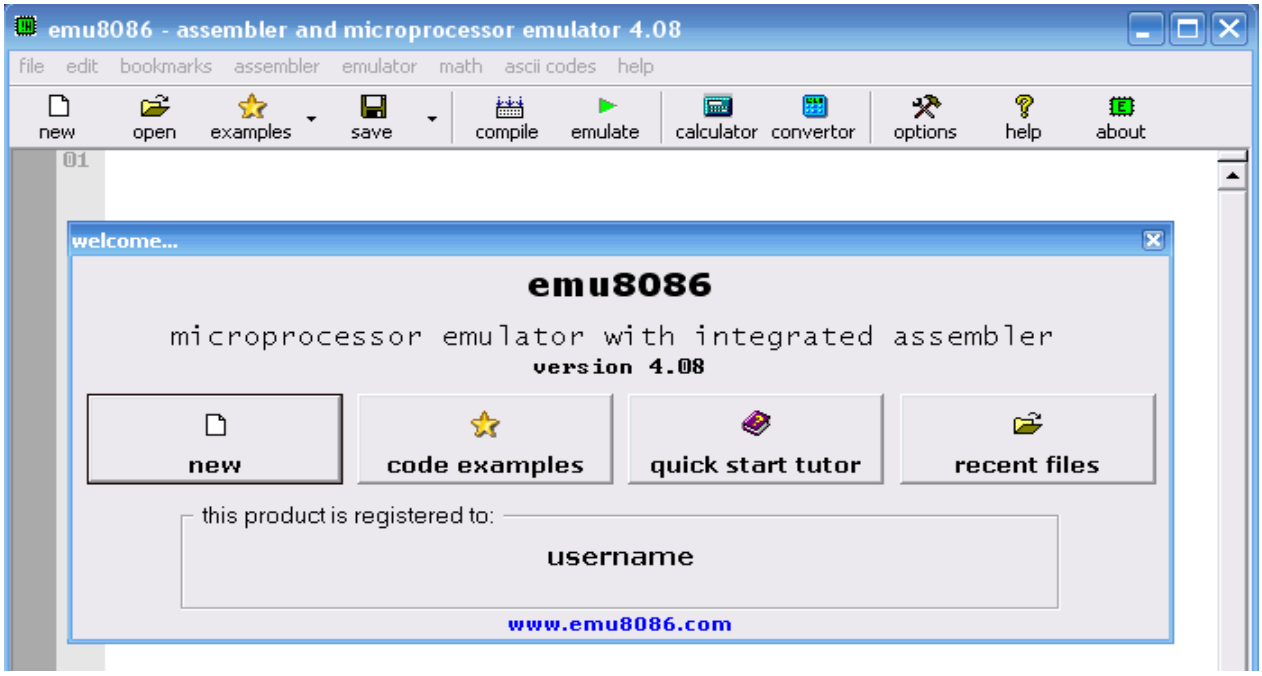


Рис. 1.1. Запуск програмного середовища Emu8086

Обрати "Samples (приклади)" з меню "File".

Натиснути кнопку [COMpile and Emulate] (або натиснути клавішу F5).

Натиснути кнопку [Single Step] (покроковий режим) (або натиснути клавішу F8), і спостерігайте за виконанням коду.

Відкрити один із прикладів програми (рис. 1.2).

Надрукувати код всередині текстової області та натиснути кнопку [COMpile]. Середовище запитає, де зберегти відкомпільований файл. Після завершення компіляції можливо натиснути кнопку [Emulate] для завантаження відкомпільованого файлу в емулятор.

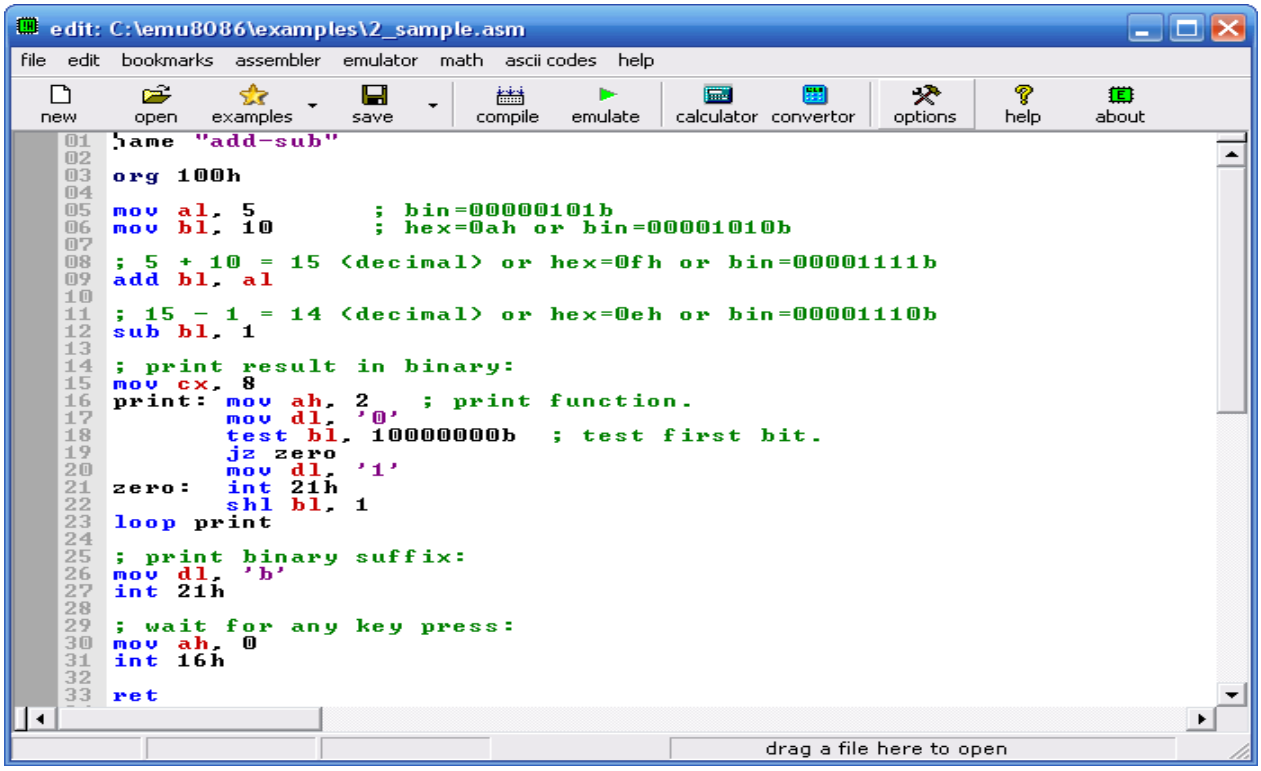


Рис. 1.2. Код асемблера усередині текстової області

Директиви, що визначають тип здійсненного файлу:

#MAKE_COM#

#MAKE_BIN#

#MAKE_BOOT#

#MAKE_EXE#

Існує можливість вставити ці директиви у вихідний код для визначення потрібного вам типу здійсненного файлу. У тому випадку, якщо компілятор не знайде ні однієї із цих директив, він запитає *тип файлу* перед його створенням.

Опис типів здійсненних файлів:

#MAKE_COM# - найстаріший і найпростіший формат здійсненного файлу. Такі файли завантажуються із префіксом 100h (256 байтів). Оберіть COM Template з меню New, якщо плануєте компілювати COM-файл. Директива компілятора ORG 100h повинна бути додана перед кодом. Виконання завжди починається з першого байту файла. Підтримується командним рядком DOS та Windows.

#MAKE_EXE# - більш "просунутий" формат здійсненного файлу. Не обмежений розмір і кількість сегментів. Сегмент стека повинен бути визначеним у програмі. Можна обрати EXE Template з меню New для створення простої exe-програми з певними сегментами Даних, Стека та Коду. Крапка входу (де починається виконання) визначається програмістом. Підтримується командним рядком DOS й Windows.

#MAKE_BIN# - простий виконуючий файл. Існує можливість визначити значення всіх регістрів, сегмент і зсув для області пам'яті, куди цей файл буде завантажений. Якщо завантажити файл "MY.BIN" в емулятор, то він буде доступний для файлу "MY.BINF". Завантажиться файл "MY.BIN" у місце, де розташований файл "MY.BINF".

Регістри встановлюються з файлу "MY.BINF" (відкрийте цей файл у редакторі для зміни або перегляду). У тому випадку, якщо емулятор не знайде файл "MY.BINF", будуть використовуватися поточні значення регістрів і файл "MY. BIN" завантажиться в поточний CS: IP.

1.2 Процес налагодження програми в середовищі емулятора EMU8086

Після успішної компіляції потрібно натиснути кнопку [Emulate], щоб завантажити файл, що компілюється в емуляторі (рис. 1.3).

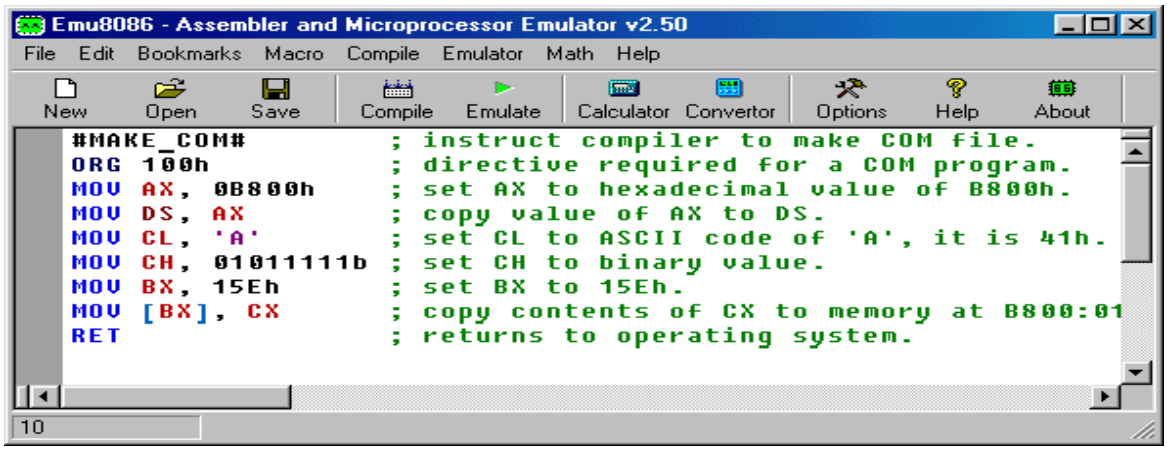


Рис. 1.3. Завантаження файлу, що компілюється в емуляторі

Для завантаження коду в емулятор необхідно натиснути кнопку [Emulate] . Виконання починається зі значення в CS: IP.

Навіть якщо немає вихідного тексту існує можливість завантажити executables. Для цього слід обрати "Show Emulator" в меню "Emulator" (рис. 1.4).

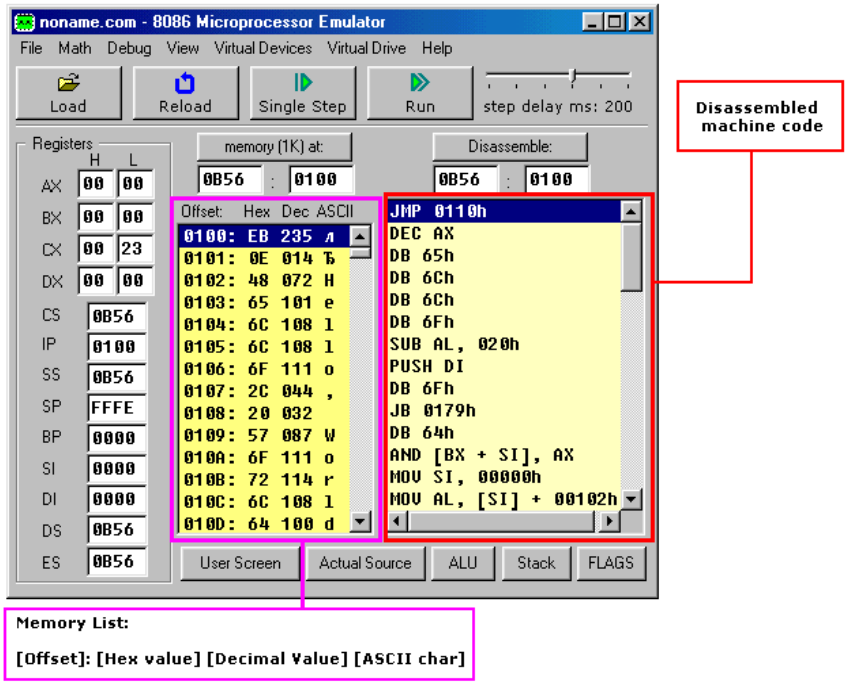


Рис. 1.4. Дослідження областей пам'яті

У пам'яті перераховують перший рядок - зсув, другий рядок - значення hexadecimal, третій рядок - десяткове значення, і останній рядок - значення символу ASCII.

Кнопка [Single Step] виконує команди, одна за іншою зупиняє роботу кожної команди.

Кнопка [Run] виконує команди одну за іншою із затримкою, установленою затримкою кроку між командами.

Двічі натиснувши на текстових полях регістра, відкриється вікно "Extended Viewer" зі значенням регістра, перетвореного до всіх можливих форм. Є можливість змінювати значення регістра безпосередньо в цьому вікні.

Двічі натиснувши на елементі списку пам'яті, відкриється "Extended Viewer" зі значенням WORD, завантаженим зі списку пам'яті в обраному місці розташування. Менш істотний байт - у молодшій адресі: LOW BYTE, завантажений від обраної позиції та HIGH BYTE - від наступної адреси пам'яті. Існує можливість змінювати значення слова пам'яті безпосередньо у вікні "Extended Viewer".

Також можливо змінювати значення регістрів під час виконання. Кнопка [Flags] дозволяє переглядати й змінювати прапорці під час виконання.

Виконання завжди починається з першого байту файла. Цей тип файлу унікальний для емулятора *Emu8086*.

1.3 Обробка помилок у середовищі емулятора EMU8086

Якщо програма працює не зовсім коректно, то потрібно виконати етап ВІДЛАГОДЖЕННЯ програми за допомогою спеціальної програми-відладчика (рис. 1.5).

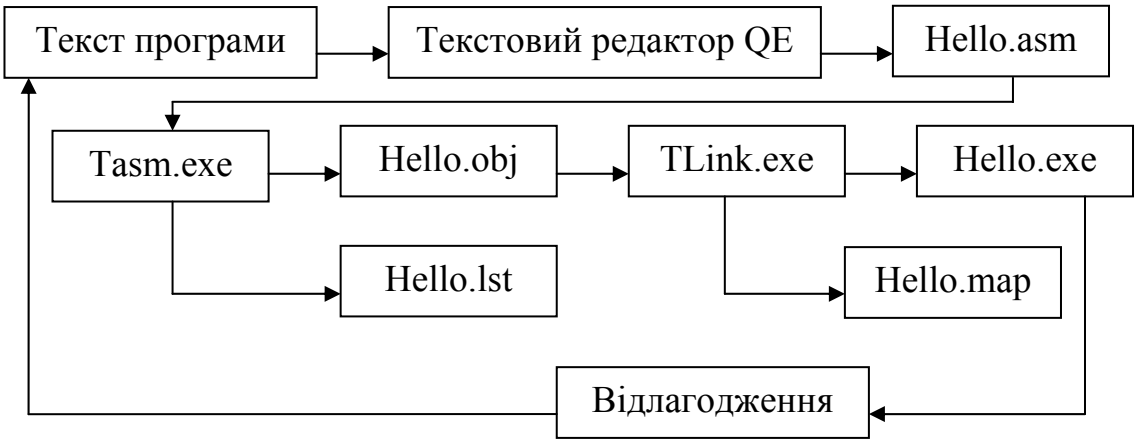


Рис. 1.5 Процес створення асемблерної програми

При знаходженні помилки доводиться проводити корекцію програми, повертаючись до кроку 1. Таким чином, процес створення асемблерної програми можна зобразити у вигляді наступної схеми. Кінцевою метою є працездатний виконуючий файл HELLO. EXE.

Компілятор виводить звіт про помилки в окремому вікні (рис. 1.6).

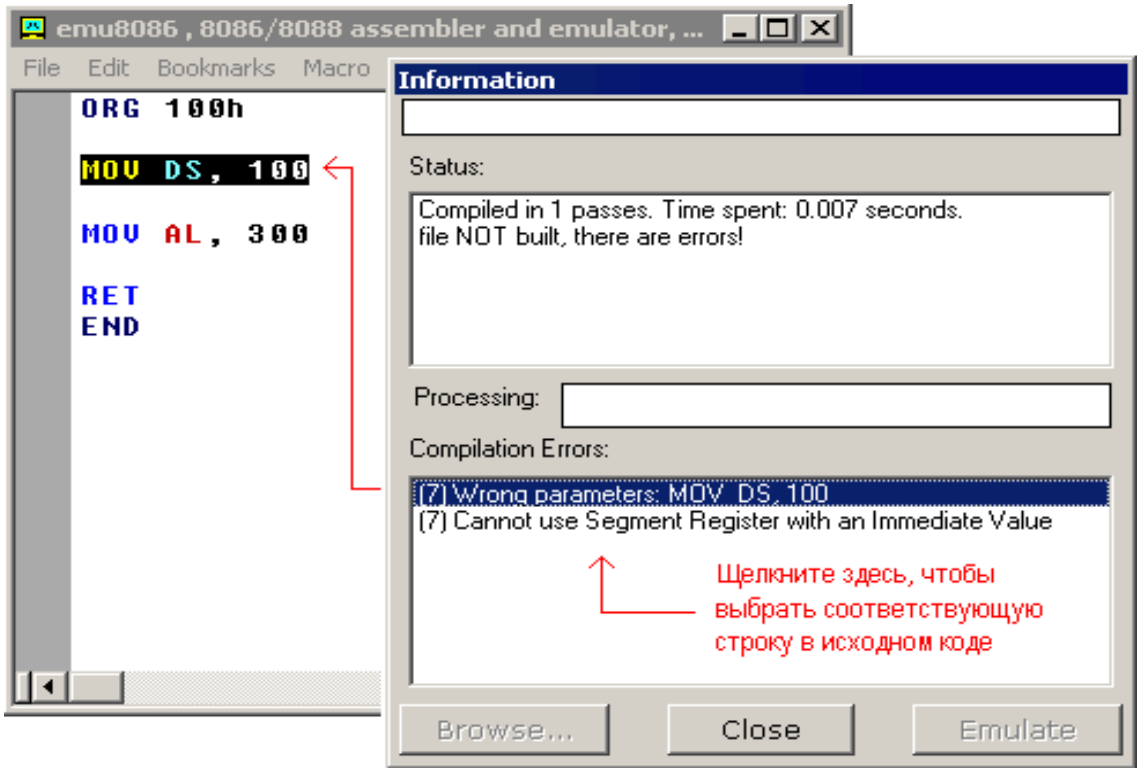


Рис. 1.6. Звіт про помилки

MOV DS, 100 - це неприпустима команда, тому що в сегментний регістр не можна встановлювати безпосередньо значення, повинні використовуватися регістри загального призначення.

MOV AX, 100 MOV DS, AX MOV AL, 300 - це неприпустима команда, регістр AL має тільки 8 біт, його максимальне значення 255 (або 11111111b), а мінімальне - 128.

Компілятор робить декілька проходів перед генерацією правильного машинного коду. Якщо він знаходить помилку та не виконує необхідну кількість проходів, вона може видати неправильне повідомлення про помилку.

Наприклад:

```
#make_COM#
ORG 100h
MOV AX, 0
MOV CX, 5
m1: INC AX
LOOP m1; це неправильна помилка!
MOV AL, 0FFFFh; помилка тут.
RET
```

Список помилок, що генеруються:

(7) Condition Jump out of range (Умова переходу за межі діапазону) !: LOOP m1 (9) Wrong parameters (Помилкові параметри): MOV AL, 0FFFFh (9) Operands do not match (Операнди не співпадають): Second operand is over 8 bits (Другий операнд більше 8 біт). Перше повідомлення (7) - неправильне. Компілятор не закінчив обчислення зсувів для міток, тому вважається, що зсув мітки m1 - це 0000. Дана адреса лежить за межами діапазону, тому починається зі зсуву 100h.

Внесіть виправлення в цей рядок: MOV AL, 0FFFFh (AL не може містити значення 0FFFFh). Ці дії усунуть обидві помилки! Наприклад:

```
#make_COM#
ORG 100h
MOV AX, 0
MOV CX, 5
m1: INC AX
LOOP m1; той же самий код без помилки!
MOV AL, 0FFh; всі!
RET
```

При збереженні зкомпільованого файлу, компілятор також зберігає два інших файли, які використаються емулятором для відображення фактичного вихідного коду при його виконанні та виборі відповідного рядка.

asm-файл містить оригінальний вихідний код, що був використаний для створення виконуючого файлу.

debug-файл містить інформацію, що дозволяє емулятору обирати рядок оригінального вихідного коду під час виконання машинного коду.

symbol - таблиця символів. Вона містить інформацію, що дозволяє відображати вікно "Variables" (Змінні). Цей текстовий файл можна подивитися в текстовому редакторі.

binf-файл містить інформацію, що використовується емулятором для завантаження BIN-файлу в зазначене місце розташування і установки значень регістрів попередньому виконанню (створюється тільки в тому випадку, якщо виконуючий файл - це BIN-файл).

1.4 Правила оформлення асемблерних програм

При наборі програм мовою асемблер необхідно дотримуватись наступних правил:

- директиви набирати великими буквами, інструкції - малими;

- не виходити за межі екрану, тобто не робити текст ширше 80 знаків - його не зручно буде редагувати й друкувати;

- для відступів користуватись табуляцією (кнопка [TAB]);

- блоки коментарів задавати з однаковим відступом. Оптимальним вважається такий рядок:

<TAB><TAB>mov<TAB>ax,<пробіл>bx< (1-3) TAB>; <пробіл> текст коментарів.

Кількість табуляцій перед коментарем визначається довжиною аргументів команди та може бути від 1 до 3.

2 Завдання для виконання

2.1. Запустити емулятор EMU8086.

2.2. Користуючись правилами оформлення асемблерних програм, виправити слова "Please Register" на будь-які інші (не забути вставити їх в апострофи).

2.3. Запустити додаток, натиснувши кнопку “Emulate” або клавішу F5.

2.4. Запустити отриманий код на виконання, використовуючи кнопку “RUN” або натиснути функціональну клавішу F9.

2.5. Відкомпілювати програму. Повернутись в головне вікно форми, попередньо закривши всі відкриті вікна, далі натиснути кнопку “COMpile”.

2.6. Отриманий COM-файл запустити у вбудованому командному рядку WINDOWS 98 на виконання або запустити сеанс dos в total COMmander.

2.7. Провести експеримент з іншими прикладами, що відкриваються при натисканні на клавішу “Samples” у головному вікні емулятора.

2.8. Ознайомитися з вбудованою в емулятор EMU8086 довідкою. В ній міститься вся необхідна інформація для роботи із програмою, базові знання щодо написання програм мовою Assembler та ін.

3 Контрольні питання

3.1. Які основні відмінності асемблерних програм?

3.2. Яка структура асемблерної програми?

3.3. У чому відмінність інструкції від директиви?

3.4. Які правила оформлення програм мовою асемблера?

3.5. Які етапи отримання виконуючого файлу?

3.6. Для чого потрібний етап відлагодження програми?

3.7. Описати основні моменти створення виконуючого файлу та емуляції роботи програми.

3.8. Які кроки технічного створення асемблерної програми в програмах TASM й MASM?

3.9. Основні можливості емулятора EMU8086.

3.10. Охарактеризувати методи боротьби із зависанням в DOS.

ЛАБОРАТОРНА РОБОТА №2

РОЗРОБКА ПЕРШОЇ ПРОГРАМИ МОВОЮ АСЕМБЛЕР

Мета роботи: ознайомлення зі структурою асемблерної програми, створення першої програми мовою асемблер.

1. Короткі теоретичні відомості

1.1. Структура асемблерної програми

Для того, щоб програма виконалася на будь-якій ОС, вона повинна бути скомпільована в виконавчий файл. Основні два формати виконуючих файлів в DOS – це COM і EXE.

Файли типу COM містять тільки скомпільований код без будь-якої додаткової інформації про програму. Весь код, дані й стек розміщуються в одному сегменті й не можуть перевищувати 64 Кб.

1	.model tiny
2	.code
3	org 100h

4	begin:
5	mov ah, 9
6	mov dx,offset message
7	int 21h
8	ret
9	message db "Привіт", 0dh, 0ah, '$'
10	end begin

Розглянемо вихідний текст програми для того, щоб зрозуміти, як вона працює.

Перший рядок визначає модель пам'яті TINY, у якій об'єднані сегменти коду, даних і стека. Ця модель призначена для створення файлів типу COM.

В DOS для формування адреси використовується сегмент і зсув. Для формування адреси рядка "ПРИВІТ" використовується пара регістрів DS (сегмент) і DX (зсув). При завантаженні *.COM-програми, всі сегментні регістри приймають значення, що дорівнює тому сегменту, в який завантажилася програма (у т.ч. й DS). Тому немає необхідності завантажувати в DS сегмент рядка (він уже завантажений).

Директива CODE: починає сегмент коду, який, у нашому випадку, також повинен містити й дані.

ORG 100h встановлює значення програмного лічильника (IP) в 100h, тому що при завантаженні COM-файлу, DOS займає перші 256 байт (100h) блоком даних PSP і розташовує код програми тільки після цього блоку. Всі програми, які компілюються у файли типу COM, повинні починатися із цієї директиви.

Мітка BEGIN: розташовується перед першою командою в програмі й в подальшому, буде використовуватися в директиві END (Begin - англ. початок; end - кінець) для того, щоб вказати, з якої команди починається програма.

Замість слова BEGIN можна було б використати іншу мітку, наприклад, START. У такому випадку, програма завершається міткою END START.

1.2. Регістри процесора.

Регістр процесора - це спеціально відведена пам'ять для зберігання будь-якого числа.

Наприклад:

Якщо потрібно скласти два числа, то можна записати:

A=5

B=8

C=A+B.

A, B й C - це свого роду регістри (якщо говорити про комп'ютер), у яких можуть зберігатися деякі дані. A=5 можна прочитати як: число 5 присвоюємо A.

Для присвоєння регістру будь-якого значення, в Асемблері існує оператор mov (від англ. move - завантажити). Команда MOV AH,9 поміщає число 9 у регістр AH - номер функції DOS "вивід рядка".

Команда MOV DX, OFFSET MESSAGE поміщає в регістр DX зсув мітки MESSAGE відносно початку сегмента даних, що у нашому випадку збігається із сегментом коду.

OFFSET (англійською - це зсув). Коли, при асемблюванні, Асемблер дійде до цього рядка, він замінить OFFSET MESSAGE на АДРЕСУ (зсув) цього рядка в пам'яті. Якщо записати OFFSET MESSAGE (хоча, правильніше буде MOV DX, WORD OFFSET MESSAGE), то в DX завантажиться не адреса (зсув), а перші два символи рядка (у цьому випадку "Пр"). Так як DX – шістнадцятирозрядний регістр, у нього можна завантажити тільки два байти (один символ завжди один байт).

Команда INT 21H викликає системну функцію DOS (int від англ. interrupt - переривання). Можна замінити рядок INT 21H на INT 33, програма буде працювати коректно. Однак в Асемблері прийнято вказувати номер переривання в шістнадцятирозрядній системі.

Переривання MS-DOS - це свого роду підпрограма (частина MS-DOS), що перебуває постійно в пам'яті й може викликатися в будь-який час із будь-якої програми.

Дана команда - основний засіб взаємодії програм з операційною системою. У прикладі викликається функція DOS номер 9 - вивести рядок на екран. Дана функція виводить рядок від початку, адреса якого задається в регістрах DS: DX, до першого символу $. При запуску COM-файлу регістр DS автоматично завантажується сегментною адресою програми, а регістр DX був підготовлений попередньою командою.

Програма додавання двох чисел:

- Початок програми

A=5 (у змінну A заносимо значення 5)

B=8 (у змінну B заносимо значення 8)

- Виклик підпрограми Додавання

C дорівнює 13

A=10 (у змінну A заносимо значення 10)

B=25

- Виклик підпрограми Додавання

C дорівнює 35

- Кінець Програми

…

- Підпрограма Додавання

C=A+B

- Повернення з підпрограми - повертаємося в те місце, звідки викликали

- Кінець підпрограми

У даному прикладі було двічі викликано підпрограму Додавання, що склала два числа, передані їй у змінних A й B. Результат міститься в змінній C. Коли викликається підпрограма, комп'ютер запам'ятовує з якого місця вона була викликана, а потім, коли підпрограма закінчила роботу, комп'ютер

повертається в те місце, звідки вона викликалася. Таким чином, можна викликати підпрограми невизначену кількість разів з будь-якого місця.

Команда RET використовується для повернення із процедури. DOS викликає COM-програми так, що команда RET коректно завершує програму.

DOS при виклику COM-файлу поміщає в стек сегмента адресу програми й нуль, так що RET передає керування на нульову адресу поточного сегмента, тобто на перший байт PSP. Там перебуває код команди INT 20H, що і використовується для повернення керування в DOS. Можна відразу закінчити програму командою INT 20h, хоча це довше на 1 байт.

Наступний рядок прикладу визначає рядок даних, що містить текст “ПРИВІТ", керуючий символ ASCII повернення каретки з кодом ODh, управляючий символ ASCII переведення рядка з кодом 0Ah і символ $, який завершає рядок (якщо цей символ не написати, то 21h переривання продовжить вивід до тих пір, поки не зустрінеться у пам'яті символ $, на екрані з’явиться “сміття”). Перше слово (англ.message - повідомлення) - назва повідомлення. Воно може бути будь-яким.

Керуючі символи (ODh й 0Ah) переводять курсор на першу позицію наступного рядка.

Директива END завершує програму, одночасно вказуючи, з якої мітки повинно починатися її виконання.

Створимо ще один рядок з назвою message1. Потім, починаючи з рядка (9) вставимо наступні команди й скомпілюємо програму заново.

9	mov dx,offset message1
10	int 21h
11	int 20h
12	message db "Привіт", 0dh, 0ah, '$'
13	message1 db "Група", 0dh, 0ah, '$'
14	end begin

2. Завдання для виконання

2.1. Запустити емулятор EMU8086.

2.2. Користуючись правилами оформлення асемблерної програми, набрати код, наведений у прикладі 1, запустити код на виконання.

2.3. Відкомпілювати приклад №2.

2.4. Повернутися в головне вікно форми, попередньо закривши всі відкриті вікна, далі натиснути кнопку “COMpile".

2.5. Отриманий COM-файл запустити в сеансі DOS.

2.6. Створити на мові Pascal програму виводу на екран слова “Привіт" і порівняти розміри одержаних файлів (Pascal й Assembler).

3. Контрольні запитання

3.1. Характеристика структури файлу типу *.COM.

3.2. Яка структура асемблерної програми?

3.3. З якою метою в код програми на асемблері для DOS вводиться рядок ORG 100h?

3.4. Яке призначення команди MOV?

3.5. Переривання 21h й 20h. Їх призначення.

3.6.Яка мета і доцільність використання команди RET замість переривання 20h?

3.6. Символ '$' методика застосування.

3.7. “BEGIN: - END BEGIN”. Правила застосування.

ЛАБОРАТОРНА РОБОТА №3

*СТРУКТУРА ВИКОНУЮЧИХ ФАЙЛІВ ТИПУ *. EXE. ПРОСТІ АРИФМЕТИЧНІ ДІЇ МОВОЮ АСЕМБЛЕРА*

Мета роботи: Вивчення принципів складання найпростіших *.exe-програм. Вивчення прийомів роботи з найпростішими операторами арифметичних дій.

1. Короткі теоретичні відомості.

Файли типу EXE містять заголовок, у якому описується розмір файлу, необхідний об’єм пам'яті, список команд у програмі, що використовують абсолютні адреси, які залежать від розташування програми в пам'яті і т.д. Exe-файл може мати будь-який розмір. Формат EXE також використовується для виконуючих файлів у різних версіях DOS-розширювачів й Windows, але зі значними змінами.

Операційна система DOS не використовує розширення для визначення типу файлу. Перші два байти заголовка exe-файлу - символи "MZ" або "ZM". Якщо файл починається із цих символів і довший деякого граничного значення, різного для різних версій DOS, то він завантажується як EXE, якщо ні – то як COM.

Exe-програми дещо складніші у виконанні, але для них відсутні обмеження розміру в 64 кілобайта, так що всі більші за розміром програми використовують саме цей формат. Звичайно, асемблер дозволяє вмістити в 64 кілобайтах досить складні й більші алгоритми, а всі дані зберігати в окремих файлах.

Простий приклад EXE-файла:

. model small	; сегмент стека розміром в 256 байт
. stack 100h	; сегмент стека розміром в 256 байт
. code	; сегмент коду, що містить і дані.

Begin:	; мітка початку коду програми
mov ax,@data;	;сегментна адреса рядка message, поміщається в DS
mov ds,ax	
mov dx,offset string	поміщає в регістр DX зсув мітки String відносно початку сегменту даних
mov ah,9	; поміщає номер функції DOS "вивід рядка (9)" у регістр AH.
int 21h	; функція DOS "вивід рядка"
mov ax,4C00h	; завершення програми типу - exe
int 21h	;функція DOS "завершити програму"
. data	; початок сегмента даних
string db "Privet", 0Dh,0Ah,'$'	; рядок, що містить дані, які виводять на екран.
end begin	; мітка закінчення коду програми

У прикладі визначаються три сегменти - сегмент стека директивою STACK розміром в 256 байт, сегмент коду, що починається з директиви CODE, і сегмент даних, що починається з DATA. При запуску exe-програми регістр DS уже не містить адреси сегмента з рядком string (він вказує на сегмент, що містить блок даних PSP), а для виклику використовуваної функції DOS цей регістр повинен мати сегментну адресу рядка. Команда MOV AX,@DATA завантажує в AX сегментну адресу групи сегментів даних @DATA, а MOV DS,AX копіює його в DS. Програми типу EXE повинні завершуватися системним викликом DOS 4Ch: у регістр AH поміщається значення 4Ch, у регістр AL - код повернення (у даному прикладі код повернення 0, регістри AH й AL завантажуються однією командою MOV AX,4C00h), після чого викликається переривання 21h.

2. Прості арифметичні оператори.

2.1. Додавання.

Команда ADD здійснює додавання першого й другого операндів. Вихідне значення першого операнду втрачається, заміщаючись результатом додавання. Другий операнд не змінюється. У якості першого операнда команди ADD

можна вказувати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обидва операнди одночасно як комірки пам'яті. Операнди можуть бути байтами або словами й представляти числа зі знаком або без знака. Команду ADD можна використовувати для додавання як звичайних цілих чисел, так і двійково-десяткових (з використанням регістра AX для зберігання результату). Команда впливає на прапорці OF, SF, ZF, AF, PF й CF.

Команда	Призначення	Процесор
ADD приймач, джерело	Додавання	8086

Приклади:

mov al,10 - --> завантажуємо в регістр AL число 10

add al,15 - --> al = 25; al - приймач, 15 - джерело

mov ax,25000 - --> завантажуємо в регістр AX число 25000

add ax,10000 - --> ax = 35000; ax - приймач, 10000 - джерело

mov cx, 200 - --> завантажуємо в регістр CX число 200

mov bx,760 - --> a в регістр BX - 760

add cx,bx - --> cx = 960, bx = 760 (bx не змінюється); cx - приймач, bx – джерело

2.2. Віднімання.

Команда SUB віднімає другий операнд (джерело) з першого (приймача) і поміщає результат на місце першого операнда. Вихідне значення першого операнда втрачається. Таким чином, якщо команду віднімання записати в загальному вигляді:

SUB операнд1, операнд2

то її дію можна умовно зобразити наступним чином:

операнд1 - операнд2 - > операнд1

У якості першого операнда можна вказати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обоє операндів одночасно як комірки пам'яті. Операнди можуть бути байтами або словами й представляти числа зі знаком або без знака. Команда впливає на прапори OF, SF, ZF, AF, PF й CF.

Команда	Призначення	Процесор
SUB приймач, джерело	Віднімання	8086

Приклади:

mov al,10

sub al,7 - --> al = 3; al - приймач, 7 - джерело

mov ax,25000

sub ax,10000 - --> ax = 15000; ax - приймач, 10000 - джерело

mov cx,100

mov bx,15

sub cx,bx - --> cx = 85, bx = 15 (bx не змінюється); cx - приймач, bx - джерело

2.3. Інкремент (збільшення на 1).

Команда INC додає 1 до операнду, в якості якого можна вказувати регістр (крім сегментного) або комірку пам'яті розміром у байт або у слово.

Не допускається використовувати в якості операнда безпосереднє значення. Операнд інтерпретується як число без знака. Команда впливає на прапорці OF, SF, ZF, AF й PF. Команда не впливає на прапорець CF; якщо потрібно вплинути на цей прапорець, то необхідно використати команду Add Op, l.

Команда INC збільшує на одиницю регістр або значення операнда в пам'яті, еквівалентна команді ADD, але виконується набагато швидше.

Команда	Призначення	Процесор
INC приймач	Збільшення на одиницю	8086

Приклади:

mov al,15

inc al - --> AL = 16 (еквівалентна add al,1)

mov dh,39h

inc dh - -->DH = 3Ah (еквівалентна add dh,1)

mov cl,4Fh

inc cl - --> CL = 50h (еквівалентна add cl,1)

2.4. Декремент (зменшення на 1).

Команда DEC віднімає 1 з операнда, у якості якого можна вказувати регістр (крім сегментного) або комірку пам'яті розміром як у байт, так й у слово. Не допускається використовувати в якості операнда безпосереднє значення. Операнд інтерпретується як число без знака. Команда впливає на прапорці OF, SF, ZF, AF й PF.

Вона еквівалентна команді SUB, але виконується набагато швидше.

Команда	Призначення	Процесор
DEC приймач	Зменшення на одиницю	8086

Приклади:

mov al,15

dec al - --> тепер AL = 14 (еквівалентна sub al,1)

mov dh,39h

dec dh - --> DH = 38h (еквівалентна sub dh,1)

mov cl,4Fh

dec cl - --> CL = 4Dh (еквівалентна sub cl,1)

3. Завдання для виконання.

3.1. Запустити емулятор EMU8086.

3.2. Одержати завдання у викладача (один з п'яти варіантів табл. №1) і, користуючись правилами оформлення асемблерних програм, написати програми розрахунку значення A (два-три варіанта).

2.3. Програму скомпілювати у файл типу *. exe.

3. Контрольні запитання

3.1. Структура файлів типу *. exe.

3.2. Структурні відмінності файлів *.exe від *.COM в операційному середовищі DOS.

3.3. Команда add. Основне призначення.

3.4. Команда sub. Основне призначення.

3.5. Команда inc. Основне призначення.

3.6. Команда dec. Основне призначення.

Таблиця №1

№ вар.	Розрахункова формула	B	C	D
1	A=B+C-D	1	35	23
2	A=B+C+D	65	1	1
3	A=C-D+B	1	33	1
4	A=D+A-B	18	1	88
5	A= B-C+D	45	10	1

За узгодженням з викладачем можна змінити як розрахункову формулу, так і значення коефіцієнтів (B, C, D).

ЛАБОРАТОРНА РОБОТА № 4

СПОСОБИ АДРЕСАЦІЇ МОВОЮ АСЕМБЛЕР

Мета роботи: вивчити основні способи адресації.

1. Короткі теоретичні відомості про способи адресації

Способом або режимом адресації називають процедуру знаходження операнда для виконуючої команди. Якщо команда використовує два операнда, то для кожного з них повинен бути заданий спосіб адресації, причому режими адресації першого й другого операнда можуть як збігатися, так і розрізнятися. Операнди команди можуть перебувати в різних місцях: безпосередньо в складі коду команди, у якому-небудь регістрі, у комірці пам'яті; в останньому випадку існує кілька можливостей вказівки його адреси. Способи адресації є елементом архітектури процесора, відбиваючи закладені в ньому можливості пошуку операндів. З іншої сторони, різні способи адресації певним чином позначаються в мові асемблера і є розділом мови.

У програмах, написаних мовою асемблера термін "операнд" застосовується для позначення тих фізичних об'єктів, з якими має справу процесор при виконанні машинної команди й, говорячи про операнди команд мови, насправді розуміють операнди машинних команд. Стосовно команд асемблера використовується термін "параметри".

В архітектурі сучасних 32-розрядних процесорів Intel передбачені досить витончені способи адресації.

Розрізняють наступні режими адресації:

- ✓ регістровий;
- ✓ безпосередній;
- ✓ прямий;
- ✓ регістровий непрямий (базовий або індексний);

- ✓ регістровий непрямий зі зсувом (базовий або індексний);
- ✓ базово-індексний;
- ✓ базовий індексний зі зсувом.

1.1. Регістровий режим

Значення операнда-джерела попередньо запам'ятовується в одному з вбудованих регістрів мікропроцесора.

Сам регістр стає ефективною адресою. Операнд (байт або слово) знаходиться в регістрі. Цей спосіб застосовується для всіх програмно-адресованих регістрів процесора:

inc CX	; Збільшення на 1 вмісту CX
push DS	; Сегментна адреса зберігається в стеці
xchg BX,BP	; Регістри BX й BP обмінюються вмістом
mov ES,AX	; Вміст AX пересилається в ES

1.2. Безпосередній режим

Безпосередня адресація. Операнд (байт або слово) вказується в команді й після трансляції надходить у код команди; він може мати будь-який зміст (число, адреса, код ASCII), а також бути представлений у вигляді символічного позначення.

mov AH, 40h	; Число 40h завантажується в AH
mov AL,'*'	; Код ASCII символу "*' завантажується в AL
int 21h	; Команда переривання з аргументом 21h
limit equ 528	; Число 528 отримує позначення limit
mov CX,limit	; Число, яке позначене як limit, завантажується в CX

Команда mov, яка використовується в останньому рядку, має два операнда: перший операнд визначається за допомогою регістрової адресації, другий - за допомогою безпосередньої.

Важливим застосуванням безпосередньої адресації є пересилання відносних адрес (зсувів), для цього використовується offset (зсув):

; Сегмент даних

string db “Privet"; Рядок символів

; Сегмент команд

mov DX,offset string; Адреса рядка заноситься в DX

1.3. Прямий режим.

Адресується пам'ять; адреса комірки пам'яті (слова або байта) вказується в команді (звичайно в символічній формі) і надходить у код команди:

; Сегмент даних

meml dw 0; Слово пам'яті містить 0

mem2 db 230; Байт пам'яті містить 230

; Сегмент команд

inc meml; Вміст слова mem1 збільшується на 1

mov DX, mem1; Вміст слова з ім'ям mem1 завантажується в DX

mov AL, mem2; Вміст байта з ім'ям mem2 завантажується в AL

Порівнюючи цей приклад з попереднім, зрозуміло, що вказівка в команді імені комірки пам'яті означає, що операндом є вміст цієї комірки; вказівка імені комірки з описувачем offset - що операндом є адреса комірки.

Пряма адресація пам'яті, на перший погляд, здається, простою і наочною. Якщо потрібно звернутися, наприклад, до комірки mem1, потрібно просто вказати її ім'я в програмі. Насправді, справа є складнішою. Адреса будь-якої комірки складається з двох компонентів: сегментної адреси й зсуву. Позначення mem1 й mem2 у попередньому прикладі, є зсувами. Сегментні ж адреси зберігаються в сегментних регістрах. Однак сегментних регістрів

чотири: DS, ES, CS й SS. Яким чином процесор дізнається, з якого регістра взяти сегментну адресу, і як повідомити йому про це в програмі?

Процесор розрізняє групу кодів, що носять назву префіксів. Існує кілька груп префіксів: повторення, розміру адреси, розміру операнда, заміни сегмента. Команди процесора, що звертаються до пам'яті, можуть в якості першого байту коду містити префікс заміни сегмента, за допомогою якого процесор визначає, з якого сегментного регістра взяти сегментну адресу. Для сегментного регістра ES код префікса становить 26h, для SS - 361i, для CS - 2Eh. Якщо префікс відсутній, сегментна адреса береться з регістра DS (хоча для нього теж передбачений свій префікс).

У наведеному прикладі, за замовчуванням, всі дані адресуються через сегментний регістр DS, так що замість inc meml можна було написати inc DS: mem. У випадку заміни сегментного регістра, його обов'язково потрібно вказувати явно:

inc ES: mem1

inc CS: mem2

Звертання до комірки пам'яті по відомій абсолютній адресі здійснюється таким чином:

mov AL,DS: [17h]	Завантаження в AL вмісту комірки зі зсувом 17h у сегменті, який визначається вмістом DS

1.4. Регістровий непрямий режим (базовий й індексний).

Адресується пам'ять (байт або слово). Відносна адреса комірки пам'яті знаходиться в регістрі, позначення якого заключається в прямі дужки. У МП 86 непряма адресація припустима тільки через регістри BX, BP, SI й DI. При використанні регістрів BX або BP адресацію називають базовою, при використанні регістрів SI або DI - індексною.

Якщо непряма адресація здійснюється через один з регістрів BX, SI або DI, то мається на увазі сегмент, який адресується через DS, тому при адресації через цей регістр позначення DS: можна опустити:

mov es: [bx],'1' → mov [bx],'1'

Цей фрагмент ефективніший попереднього щодо витрат пам'яті. Через відсутність у коді останньої команди префікса заміни сегмента, він займає на 1 байт менше місця.

Регістри BX, SI й DI у даних прикладах рівнозначні, і з однаковим успіхом можна скористатися кожним з них.

Інша річ полягає з регістром BP. Даний регістр спеціально призначений для роботи зі стеком, і при адресації через цей регістр у режимах непрямої адресації, мається на увазі сегмент стека; інакше кажучи, як сегментний регістр за замовчуванням використовується регістр SS.

Непряма адресація до стеку застосовується в тих випадках, коли необхідно звернутися до даних, які знаходяться в стеці, без вилучення їх звідти (наприклад, якщо ці дані необхідно зчитувати неодноразово).

Позначення даного способу адресації:

[BX]	(мається на увазі DS: [BX])
[BP]	(мається на увазі SS: [BP])
[SI]	(мається на увазі DS: [SI])
[DI]	(мається на увазі DS: [DI])

Використання базової адресації, на перший погляд, знижує ефективність програми, тому що вимагає додаткової операції - завантаження в базовий регістр необхідної адреси. Однак команда з базовою адресацією займає менше місця в пам'яті (тому що в неї не входить адреса комірки) і виконується швидше команди з прямою адресацією (через те, що команда коротша, процесору потрібно менше часу на її зчитування з пам'яті). Тому базова адресація ефективна в тих випадках, коли по заданій адресі доводиться звертатися

багаторазово, особливо, у циклі. З іншого боку, можливості цього режиму адресації невеликі, і на практиці частіше використовують більш складні способи.

Приклади:

mov SI, offset string	; В SI завантажується відносна адреса комірки string
mov AX, [SI]	; Вміст комірки string завантажується в AX
inc [SI]	; Збільшується вміст комірки string
mov BX, [SI]	; Новий вміст комірки string завантажується в BX
mov DI, SI	; Відносна адреса комірки string копіюється в DI

1.5. Регістровий непрямий режим зі зсувом (базовий й індексний).

Адресується пам'ять (байт або слово). Відносна адреса операнда визначається, як сума вмісту регістра BX, BP, SI або DI і зазначеної в команді константи, іноді вона називається зсувом. Зсув може бути числом або адресою. Так само, як й у випадку базової адресації, при використанні регістрів BX, SI й DI мається на увазі сегмент, що адресується через DS, а при використанні BP мається на увазі сегмент стека й, відповідно, регістр SS.

зсув = {SP, BP, DI, SI, BX} + зсув з команди

Іноді можна зустрітися з альтернативними позначеннями того ж способу адресації, який допускає асемблер. Замість, наприклад, 4 [BX] можна з таким же успіхом написати [BX+4], 4+ [BX] або [BX] +4. Така неоднозначність мови нічого крім плутанини не приносить, однак її треба мати на увазі, тому що із цими позначеннями можна зустрітися, наприклад, розглядаючи текст дезасембльованої програми.

Розглянемо тепер приклад використання базової адресації зі зсувом при звертанні до стеку:

зсув = {SP, BP, DI, SI, BX} + зсув з команди

Тут квадратні дужки [] - це теж оператор. Він обчислює адресу як суму того, що перебуває всередині дужок з тим, що перебуває зовні.

array db 0, 10, 20, 30, 40, 50, 60*;* Нехай у сегменті даних визначений масив.

Послідовність команд:

mov BX,5

mov AL, array [5] ; завантажує в AL елемент масиву з індексом 5, тобто 50.

Той же результат можна отримати й у наступних послідовностях команд:

mov BX,offset array

mov AL,5 [BX]

або

mov AL, [BX] +5

mov AL, [BX+5]

1.6. Базово-індексний режим

Адресується пам'ять (байт або слово). Відносна адреса операнда визначається, як сума вмісту наступних пар регістрів:

зсув [BX] [SI]	(мається на увазі DS: *зсув* [BX] [SI])
зсув [BX] [DI]	(мається на увазі DS: *зсув* [BX] [DI])
зсув [BP] [SI]	(мається на увазі SS: *зсув* [BP] [SI])
зсув [BP] [DI]	(мається на увазі SS: *зсув* [BP] [DI])

У всіх цих випадках можна також записати:

зсув [BX+SI]
[*зсув* +BX+SI]
[BX+SI] +*зсув*

Це надзвичайно розповсюджений спосіб адресації, особливо, при роботі з масивами. У ньому використовуються два регістри, при цьому одним з них

повинен бути базовий (BX або BP), а іншим - індексний (SI або DI). Як правило, в одному з регістрів перебуває адреса масиву, а в іншому - індекс у ньому.

1.7. Базово-індексна адресація зі зсувом.

Адресується пам'ять (байт або слово). Відносна адреса операнда визначається як сума вмісту двох регістрів і зсуву.

Даний спосіб адресації є удосконаленням попередньої. У ньому використовуються ті ж пари регістрів, але отриману з їхньою допомогою результуючу адреса можна ще змістити на значення зазначеної в команді константи. Як й у випадку базово-індексної адресації, константа може представляти собою індекс (тоді в одному з регістрів повинна міститися базова адреса пам'яті), але може бути й базовою адресою. В останньому випадку регістри можуть використовуватися для зберігання складових індексу.

Приведемо формальний приклад розглянутого режиму адресації. Нехай у сегменті даних визначений масив з 24 байт

syms db 'ЙЦУКЕНГШЩЗХЪ'

db 'йцукенгшщзхъ'

Послідовність команд

mov BX,12mov SI,6

mov DL,syms [BX] [SI] *;* завантажить у регістр DL елемент із індексом 6 із другого рядка, тобто код ASCII букви ‘Г’

Той же результат буде отриманий й у наступному варіанті:

mov BX,offset syms

mov SI,6

mov DL,12 [BX] [SI]

2. Порядок виконання роботи

1. За допомогою редактора емулятора EMU 8086 написати програму, вихідний текст якої приводиться в лістингу №1.

2. Створити виконуючий файл типу MZ.

3. Вивчити структуру програми, а також структуру сегмента даних програми: знайти у ньому всі змінні, які визначені в тексті програми.

4. Переробити програму з використанням спрощених директив сегментації так, щоб одержати виконуючий файл типу .COM і порівняти розміри програм.

5. Виконати перші 5 кроків програми, аналізуючи й записуючи стан регістрів на кожному кроці.

6. Занести у CX 00FFh. Визначити по способу адресації комірку пам'яті в сегменті, де відбудуться зміни, записати її адресу.

7. Виконати подальші кроки програми, аналізуючи можливі способи адресації.

8. Підготувати звіт, що повинен містити тексти програм, адреси сегментних регістрів і запис адрес комірок пам'яті проти відповідних команд, а також запис вмісту цих комірок.

9. У звіті повинні міститися відповіді на наступні запитання.

3. Контрольні питання

3.1. Як переслати вміст X в Y?

3.2. Чим відрізняються команди

MOV [si], cx

та

MOV si, cx?

3.3. До якого способу адресації відноситься команда MOV dx, offset message?

3.4. Які сегменти використовуються при наступних варіантах адресації:

[BX] [SI], [BX] [DI], [BP] [SI], [BP] [DI] ?

3.5. Що відбудеться при виконанні інструкції:

MOV AL, DS: 17h?

Чим ця команда відрізняється від наступної:

MOV AL, DS: [17h] ?

3.6. Нехай у сегменті даних визначений масив

Array db 0,15,22,31,44,45,62,67,76,99

3.7. Що виявиться в регістрі AL після виконання команд:

MOV BX, 5

MOV AL, array [BX] ?

Який це спосіб адресації?

3.8. Вказати, які інструкції в програмі (лістинг №1), створеної в даній лабораторній роботі, відносяться до інструкцій:

- ✓ з безпосереднім;
- ✓ з непрямим режимом адресації?

3.9. Вказати спосіб запису звертання прямо до комірки пам'яті по відомій абсолютній адресі?

3.10. Префікси, види префіксів. Префікси заміни сегмента.

3.11.Перелічити регістри непрямої й базової адресації. Описати відмінності.

3.12. Сутність ефективності базової адресації в порівнянні із прямою.

Лістинг №1.

TITLE MOVE2 MOVE2 SEGMENT 'CODE' ASSUME CS: MOVE2, DS: DATA MYPROC PROC OUTPROC:

MOV AX,DATA MOV DS,AX MOV AH,BH MOV AH,X MOV CH,3 MOV AX,3 MOV AX,Y MOV [SI],CX MOV [BP],CX MOV [SI],258 MOV [BP+516],1027 MOV BYTE PTR X,255 MOV BYTE PTR [DI+515],4 MOV WORD PTR [DI+515],4 MOV [DI+BP+515],258 MOV AX, [SI+BX+258] MOV AH,4CH INT 21H MYPROC ENDP MOVE2 ENDS DATA SEGMENT X DB 1 Y DW 2 DATA ENDS END MYPROC

ЛАБОРАТОРНА РОБОТА № 5

КОМАНДИ, ЩО ОБСЛУГОВУЮТЬ РОБОТУ ІЗ КЛАВІАТУРОЮ

Мета роботи: освоїти команди зчитування даних і керування клавіатурою. Вивчити способи роботи із процедурами.

1. Короткі теоретичні відомості

1.1. Засоби BIOS

BIOS надає більше можливостей у порівнянні з DOS для зчитування даних і керування клавіатурою. Наприклад, функціями DOS не можна визначити натискання комбінацій клавіш типу Ctrl-Alt-Enter або натискання двох клавіш Shift одночасно, DOS не може визначити момент відпускання натиснутої клавіші, і нарешті, в DOS немає аналога функції C ungetch (), що поміщає символ у буфер клавіатури, немов його ввів користувач. Все це можна виконати, використовуючи різні функції переривання 16h й операції з байтами стану клавіатури.

INT 16h, AH = 0, 10h, 20h - Зчитування символу з очікуванням

Ввід:	AH = 00h (83/84-key), 10h (101/102-key), 20h (122-key)
Вивід:	AL = ASCII-код символу, 0 або префікс скан-кода AH = скан-код натиснутої кнопки або розширений ASCII-код

Кожній кнопці на клавіатурі відповідає так званий скан-код (див. додаток 1), що відповідає тільки цій кнопці. Цей код надсилається клавіатурою при кожному натисканні й відпусканні кнопки й обробляється BIOS (оброблювачем переривання INT 9). Переривання 16h дає можливість одержати код натискання, не перехоплюючи цей оброблювач. Якщо натиснутій кнопці

відповідає ASCII-символ, то в AH повертається код цього символу, а в AL - скан-код кнопки. Якщо натиснутій кнопці відповідає розширений ASCII-код, в AL повертається префікс скан-кода (наприклад, E0 для сірих кнопок) або 0, якщо префікса немає, а в AH - розширений ASCII-код. Функція 00H обробляє тільки комбінації, що використовують кнопки клавіатури з 84-кнопками, l0h обробляє всі 101 - 105-кнопкові комбінації, 20h - 122-кнопкові. Тип клавіатури можна визначити за допомогою функції 09h переривання 16h, якщо вона підтримується BIOS (чи підтримується ця функція, можна довідатися за допомогою функції C0h переривання 15h).

INT 16h, AH = 1, 11h, 21h - Перевірка символу

Ввід:	AH = 01h (83/84-key), 11h (101/102-key), 21h (122-key)
Вивід:	ZF = 1, якщо буфер порожній ZF = 0, якщо в буфері присутній символ AL = ASCII-код символу, 0 або префікс скан-коду AH = скан-код натиснутої кнопки або розширений ASCII-код

Символ залишається в буфері клавіатури, хоча деякі BIOS видаляють символ з буфера при обробці функції 01h, якщо він відповідає розширеному ASCII-коду, відсутньому на 84-клавішних клавіатурах.

INT 16h, AH = 05h - Помістити символ у буфер клавіатури

Ввід:	AH = 05h CH = скан-код CL = ASCII-код
Вивід:	AL = 00, якщо операція виконана успішно AL = 01h, якщо буфер клавіатури переповнений AH модифікується багатьма BIOS

Зазвичай можна помістити 0 замість скан-коду в CH, якщо функція, що буде виконувати читання з буфера, використовуватиме саме ASCII-код.

Наприклад, наступна програма при запуску з DOS викликає команду DIR.

Приклад №1.1

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov cl,'d'	; заносимо в регістр cl - ASCII-код букви "d"
call ungetch	; переходимо на мітку ungetch: (виклик підпрограми)
mov cl,'i'	; заносимо в регістр cl - ASCII-код букви "i"
call ungetch	; викликаємо підпрограму
mov cl,'r'	; заносимо в регістр cl - ASCII-код букви "r"
call ungetch	; викликаємо підпрограму
mov cl,0Dh	; переведення рядка
Ungetch proc	; мітка початку підпрограми
mov ah,05h	; AH = номер функції
mov ch,0	; CH = 0 (скан-код неважливий)
int 16h	; виклик DOS (переривання)
ret	; функція DOS "завершити роботу процедури"
Ungetch endp	; закінчення підпрограми
end begin	; мітка закінчення коду програми

Оформлення процедур (підпрограм):

Call ungetch

Ungetch proc

Ungetch endp

Ungetch - назва процедури

Call - виклик підпрограми

Proc - procedure - процедура

endp - end procedure - кінець процедури

Слід відзначити, що з однієї підпрограми можна викликати інші. З інших - наступні. Найчастіше з метою спрощення програми й, тим самим, зменшення її розміру, програмісти використовують просте зв'язування команд:

Call мітка

мітка:

Будь-який виклик підпрограм закінчується поверненням в блок коду командою ret **(**ret **-** return **-** повернення**).**

INT 16h, AH = 02h, 12h, 22h **-** зчитати стан клавіатури

Ввід:	AH = 02h (83/84-key), 12h (101/102-key), 22h (122-key)
Вивід:	AL = байт стану клавіатури 1 AH = байт стану клавіатури 2 (тільки для функцій 12h й 22h)

Байт стану клавіатури 1 (цей байт завжди розташований у пам'яті за адресою 0000h: 0417h або 0040h: 0017h):

Біт 7: Ins включений

Біт 6: CapsLock включений

Біт 5: NumLock включений

Біт 4: ScrollLock включений

Біт 3: Alt натиснута (будь-яка Alt для функції 02h, часто тільки ліва Alt для 12h/22h)

Біт 2: Ctrl натиснутий (будь-яка Ctrl)

Біт 1: Лівий Shift натиснутий

Біт 0: Правий Shift натиснутий

Байт стану клавіатури 2 (цей байт завжди розташований у пам'яті за адресою 0000h: 0418h або 0040h: 0018h):

Біт 7: SysRq натиснутий

Біт 6: CapsLock натиснутий

Біт 5: NumLock натиснутий

Біт 4: ScrollLock натиснутий

Біт 3: Права Alt натиснута

Біт 2: Правий Ctrl натиснутий

Біт 1: Лівий Alt натиснутий

Біт 0: Лівий Ctrl натиснутий

Дані байти постійно розташовуються в пам'яті, тому замість виклику переривання часто зручніше просто зчитувати значення напряму. Крім того, у дані байти можна записувати нові значення, і BIOS відповідно змінить стан клавіатури.

Крім зазначених двох байт BIOS зберігає у своїй області даних і весь клавіатурний буфер, до якого також можна звертатися напряму. Буфер займає 16 слів з 0h: 041Eh по 0h: 043Dh включно, причому адреса 0h: 041Ah – адреса початку буфера, тобто адреса, за якою розташовується наступний введений символ, а адреса 0h: 041Ch – адреса кінця буфера, тому якщо ці дві адреси рівні, буфер порожній. Буфер діє як кільце: якщо початок буфера – 043Ch, а кінець – 0420h, то в буфері перебувають три символи за адресами 043Ch, 041Eh й 0420h. Кожен символ зберігається у вигляді слова – того самого, яке повертає функція 10h переривання INT 16h. У деяких випадках буфер розміщується за іншими адресами, тоді адреса його початку зберігається в області даних BIOS за адресою 0480h, а кінця – за адресою 0482h. Прямий доступ до буфера клавіатури трохи швидший, ніж виклик відповідних функцій BIOS, і для додатків, що вимагають максимальної швидкості, таких як ігри або демо-програми, використовують керування клавіатурою на рівні портів вводу-виводу.

1.2. Засоби DOS

Як і у випадку виводу на екран, DOS надає набір функцій для зчитування даних із клавіатури, які використають стандартний пристрій уведення STDIN,

так що можна використати як джерело даних файл або стандартний вивід іншої програми.

Функція DOS 0Ah – Зчитати рядок символів з STDIN у буфер

Ввід:	AH = 0Ah DS: DX = адреса буфера
Вивід:	Буфер містить уведений рядок

Для виклику даної функції треба підготувати буфер, перший байт якого містить максимальне число символів для вводу (1-254), а вміст, якщо воно задано, може використатися як підказка для вводу. При наборі рядка обробляються клавіші Esc, F3, F5, BS, Ctrl-C/Ctrl-Break і т.д., як при наборі команд DOS (тобто Esc починає уведення спочатку, F3 відновлює підказку для уведення, F5 запам'ятовує поточний рядок як підказку, Backspace стирає попередній символ). Після натискання клавіші Enter рядок (включаючи останній символ CR (0Dh)) записується в буфер, починаючи з третього байта. У другий байт записується довжина реально введеного рядка без врахування завершального CR.

Розглянемо приклад програми, що виконує перетворення десяткового числа в шістнадцяткове.

Приклад №1.2

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
start:	; мітка початку коду програми
mov dx,offset messagel	; DS: DX - адреса рядка
mov ah,9	; номер функції DOS в AH
int 21h	; вивести запрошення до вводу message1
mov dx,offset buffer	; DS: DX - адреса рядка
mov ah,0Ah	; номер функції DOS в AH
int 21h	; считати рядок символів у буфер

mov dx,offset crlf	; DS: DX - адреса рядка
mov ah,9	; номер функції DOS в AH
int 21h	; перевод рядка
xor di,di	; DI = 0 - номер байта в буфері
xor ax,ax	; AX = 0 - поточне значення результату
mov cl,blength	
xor ch,ch	; обнуляємо регістр ch
xor bx,bx	; обнуляємо регістр bx
mov si,cx	; SI - довжина буфера
mov cl,10	; CL = 10, множник для MUL
asc2hex:	; мітка початку блоку asc2hex:
mov bl,byte ptr bcontents [di]	
sub bl,'0'	; цифра = код цифри - код символу "0",
jb asc_error	; якщо код символу був менше, ніж код "0",
cmp bl,9	; або більше, ніж "9",
ja asc_error	; вийти із програми з повідомленням про помилку,
mul cx	; інакше: помножити поточний результат на 10,
add ax,bx	; додати до нього нову цифру,
inc di	; збільшити лічильник на 1
cmp di,si	; якщо лічильник+1 менше числа символів -
jb asc2hex	; продовжити (лічильник зчитується від 0)
push ax	; зберегти результат перетворення
mov ah,9	; номер функції DOS в AH
mov dx,offset message2	; DS: DX - адреса рядка
int 21h	; вивести запрошення до вводу message2
pop ax	; забрати зі стека
push ax	; записати в стек
xchg ah,al	; помістити в AL старший байт
call print_al	; вивести його на екран
pop ax	; відновити в AL молодший байт
call print_al	; вивести його на екран
ret	; завершення COM-файлу
asc_error:	; початок блоку asc_error:
mov dx,offset err_msg	; DS: DX - адреса рядка
mov ah,9	; номер функції DOS в AH
int 21h	; вивести повідомлення про помилку
ret	; завершити програму
print_al:	; мітка початку блоку print_al:
mov dh,al	; заносимо в dh значення регістра al

and dh,0Fh	; DH - молодші 4 біти
shr al,4	; AL - старші
call print_nibble	; вивести старшу цифру
mov al,dh	; тепер AL містить молодші 4 біти
print_nibble:	; процедура виводу 4 біт (шістнадцяткової цифри)
cmp al,10	; три команди, що переводять цифру в AL
sbb al,69h	; у відповідний ASCII-код
das	; (див. опис команди DAS)
mov dl,al	; код символу в DL
mov ah,2	; номер функції DOS в AH
int 21h	; вивід символу
ret	; даний RET працює два рази - один раз для повернення із процедури print_nibble, викликаної для старшої цифри й другий раз - для повернення з print_al
messagel db "Десяткове число: $"	; строка, що містить виведені дані
message2 db "Шістнадцятирічне число: $"	; строка, що містить виведені дані
err_msg db "Помилка уведення"	; строка, що містить виведені дані
crlf db 0Dh,0Ah, '$'	; строка, що містить виведені дані
Buffer db 6	; максимальний розмір буфера уведення
blength db?	; розмір буфера після зчитування
bcontents:	; вміст буфера розташовується за кінцем COM-файлу
end start	; мітка закінчення коду програми

Функція 0Ah надає зручний, але обмежений спосіб вводу даних. Найчастіше використовують функції посимвольного вводу, що дозволяють контролювати відображення символів на екрані, реакцію програми на функціональні й керуючі клавіші й т.д.

Функція DOS 01h – Зчитати символ з STDIN з відгуком, очікуванням і перевіркою на Ctrl-Break

Ввід:	AH = 01h
Вивід:	AL = ASCII-код символу або 0. Якщо AL = 0, другий виклик цієї функції поверне в AL розширений ASCII-код символу

При зчитуванні за допомогою цієї функції уведений символ автоматично негайно відображається на екрані (посилає в пристрій STDOUT - так що його можна перенаправляти у файл). При натисканні Ctrl-C або Ctrl-Break виконується команда INT 23h. Якщо натиснуто клавішу, що не відповідає якому-небудь символу (стрілки, функціональні клавіші Ins, Del і т.д.), то в AL повертається 0 і функцію треба викликати ще один раз, щоб одержати розширений ASCII-код (див. додаток 1).

У трьох наступних варіантах цієї функції код символу повертається в AL за однаковим принципом.

Функція DOS 08h – Зчитати символ з STDIN без відгуку, з очікуванням і перевіркою на Ctrl-Break

Ввід:	AH = 08h
Вивід:	AL = код символу

Функція DOS 07h – Считати символ з STDIN без відгуку, з очікуванням і без перевірки на Ctrl-Break

Ввід:	AH = 07h
Вивід:	AL = код символу

Функція DOS 06h – Считати символ з STDIN без відгуку, без очікування й без перевірки на Ctrl-Break

Ввід:	AH = 07h DL = 0FFh
Вивід:	ZF = 1, якщо не була натиснута клавіша, і AL = 00 ZF = 0, якщо клавіша була натиснута. У цьому випадку AL = код символу

1.3 Службові функції DOS для роботи із клавіатурою.

Крім перерахованих функцій використовують деякі службові функції DOS для роботи із клавіатурою.

Функція DOS 0Bh - Перевірити стан клавіатури

Ввід:	AH = 0Bh
Вивід:	AL = 0, якщо не була натиснута клавіша AL = 0FFh, якщо була натиснута клавіша

Дану функцію зручно використовувати перед функціями 01, 07 й 08, щоб не чекати натискання клавіші. Крім того, виклик цієї функції дозволяє перевірити, не зчитуючи символ із клавіатури, чи була натиснута комбінація клавіш Ctrl-Break; якщо це відбулося, виконається переривання 23h.

Функція DOS 0Ch - Очистити буфер і зчитати символ

Ввід:	AH = 0Ch AL = Номер функції DOS (01, 06, 07, 08, 0Ah)
Вивід:	Залежить від викликаної функції

Функція 0Ch очищає буфер клавіатури, так що наступна функція зчитування символу буде чекати ввід із клавіатури, а не використовувати натиснутий раніше й ще не оброблений символ. Наприклад, саме ця функція використовується для зчитування відповіді на питання "Чи впевнений користувач у тому, що він хоче форматувати диск?".

Функції посимвольного вводу без відгуку можна використовувати для інтерактивного керування програмою.

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
Begin:	; мітка початку коду програми
call Wait_key	; викликаємо підпрограму
cmp al,27	; порівнюємо значення в al з кодом 27 (ESC)
je Quit_prog	; якщо так – то перехід на мітку Quit_prog
cmp al,0	; код клавіші розширений? (F1-F12 і т.п.)
je Begin	; так - повторити запит...

call Out_char	; викликати процедуру виводу натиснутої клавіші на екран
jmp Begin	; очікувати далі...
Quit_prog:	; мітка, на яку прийде програма у випадку натискання ESC
mov al,32	; помістити в AL <пробіл>
call Out_char	; викликати процедуру виводу символу в AL
int 20h	; вийти...
Wait_key proc	; процедура очікування натиску клавіші користувачем
mov ah,10h	; закінчення підпрограми
int 16h	; переривання DOS
ret	; функція DOS "завершити роботу процедури"
Wait_key endp	; закінчення процедури Wait_key
Out_char proc	; початок процедури out_char
push cx	; зберегти всі регістри, які будуть змінені підпрограмою...
push ax	; зробити це для того, щоб у подальшому не було плутанини
push es	; зберегти сегментний регістр
push ax	; зберегти AX, так як в ньому код натиснутої клавіші.
mov ax,0B800h	; установити ES на сегмент відеобуфера
mov es,ax	
mov di,0	; DI – перший символ першого рядка
mov cx, 2000	; вивести 2000 символів (80 символів у рядку, 25 рядків)
pop ax	; відновити код клавіші
mov ah,31	; кольори символу
Next_sym:	; мітка для циклу
mov es: [di],ax	; занести код клавіші та її колір (колір завжди 31)
inc di	; збільшити покажчик на 2 (перший байт - символ, другий байт - колір)
inc di	
loop Next_sym	; обробка наступного символу
pop es	; відновити збережені регістри й вирівняти стік
pop ax	
pop cx	
ret	; повернення із процедури
Out_char endp	; закінчення процедури Out_char
end Begin	; мітка закінчення коду програми

Програма виконує наступне:

- ✓ Чекає від користувача клавіші;
- ✓ якщо це розширений ASCII (F1-F12, стрілки), то ігнорує її;
- ✓ якщо це не розширений ASCII (A-Z, 0-9 і т.п.) – заповнює екран даним символом;
- ✓ якщо натискаємо ESC (27 або 1Bh), то заповнює екран пробілами (mov al, 32) і здійснює вихід.

2. Завдання для виконання.

2.1. С допомогою редактора емулятора EMU 8086 написати програми, вихідний текст яких приводиться в прикладах даної лабораторної роботи.

2.2. Створити виконуючі файли типу *.COM.

2.3. Пояснити роботу отриманих програм.

2.4. Написати програму для виводу на екран умісту регістра DS (на базі прикладу №2.1). Порівняти результат роботи власної програми й того, що показує відладчик.

2.5. Описати роботу команд DIV, PUSH, POP, SHL, TEST.

2.6. Встановити (знайти адреси і записати), де перебувають числа, поміщені в стек.

2.7. Написати програму для виводу на екран умісту регістра CS (на базі прикладу №3.1).

2.8. Запропонувати інші способи вирішення поставлених завдань.

3. Контрольні питання

3.1. Переваги використання команди SHL замість TEST (приклад №1.1)?

3.2. Чим відрізняються команди

```
SHL dx,1
```

та

```
SHL dx, cl
```

3.3. Як переслати вміст X у стек та отримати його назад?

3.4. Описати методику виводу значення байта в десятковій системі числення.

3.5. Описати методику виводу значення байта в шістнадцятковій системі числення.

3.6. Описати методику виводу двійкового коду числа, записаного в регістр X.

3.7. Стек. Принцип роботи. Команди роботи зі стеком.

3.8. Укажіть відмінності в роботі тандема команд.

```
push DS
pop ES
```

від

```
push DS
pop ES
```

Додаток №1

Основні скан-коди клавіш клавіатури.

Клавіша	Код	Клавіша	Код	Клавіша	Код	Клавіша	Код
Esc	01h	Enter	1Ch	K*	37h	Ins	52h
1!	02h	Ctrl	1Dh	Alt	38h	Del	53h
2 @	03h	A	1Eh	SP	39h	SysRq	54h
3 #	04h	S	1Fh	Caps	3Ah	Macro	56h
4 $	05h	D	20h	F1	3Bh	F11	57h
5%	06h	F	21h	F2	3Ch	F12	58h
6 ^	07h	G	22h	F3	3Dh	PA1	5Ah
7 &	08h	H	23h	F4	3Eh	F13/LWin	5Bh
8 *	09h	J	24h	F5	3Fh	F14/RWin	5Ch
9 (	0Ah	K	25h	F6	40h	F15/Menu	5Dh
0)	0Bh	L	26h	F7	41h	F16	63h
- _	0Ch	;:	27h	F8	42h	F17	64h
= +	0Dh	' "	28h	F9	43h	F18	65h
BS	0Eh	` ~	29h	F10	44h	F19	66h
Tab	0Fh	LShift	2Ah	Num	45h	F20	67h

Q	10h	\ |	2Bh	Scroll	46h	F21	68h
W	11h	Z	2Ch	Home	47h	F22	69h
E	12h	X	2Dh	-	48h	F23	6Ah
R	13h	C	3Eh	PgUp	49h	F24	6Bh
T	14h	V	2Fh	K-	4Ah	EraseEOF	6Dh
Y	15h	B	30h		4Bh	Copy/Play	6Fh
U	16h	N	31h	K5	4Ch	CrSel	72h
I	17h	M	32h	®	4Dh	Delta	73h
O	18h	, <	33h	K+	4Eh	ExSel	74h
P	19h	. >	34h	End	4Fh	Clear	76h
[{	1Ah	/?	35h	I	50h		
] }	1Bh	RShift	36h	PgDn	51h		

ЛАБОРАТОРНА РОБОТА № 6

ВИВІД НА ЕКРАН У ТЕКСТОВОМУ РЕЖИМІ

Мета роботи: ознайомитися з основними засобами виводу текстових даних на екран за допомогою засобів операційної системи DOS, засобами BIOS і засобами безпосереднього (прямого) відображенням у відеобуфер.

1. Засоби DOS.

1.1 Функція DOS 02h.

Функція DOS 02h – Записати символ в STDOUT з перевіркою на Ctrl-Break

Ввід:	AH = 02h DL = ASCII-код символу
Вивід:	Ніякого, відповідно до документації, але насправді: AL = код останнього записаного символу (дорівнює DL, крім випадку, коли DL = 09h (табуляція), тоді в AL повертається 20h).

Дана функція при виводі на екран обробляє деякі керуючі символи – вивід символу BEL (07h) призводить до звукового сигналу, символ BS (08h)

призводить до руху курсору вліво на одну позицію, символ HT (09h) заміняється на кілька пробілів, символ LF (0Ah) переводить курсор на одну позицію вниз, і CR (0Dh) призводить до переходу на початок поточного рядка.

Якщо в ході роботи цієї функції була натиснута комбінація клавіш Ctrl-Break, викликається переривання 23h, що за замовчуванням здійснює вихід із програми.

Простий приклад роботи функції DOS 02h.

Приклад № 1.1

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov dl,< ASCII-код символу >	; занести в регістр dl будь-який ASCII-код символу
mov ah,2	; номер функції DOS "вивід символу"
int 21h	; виклик DOS
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

Дана програма, виводить на екран будь-який ASCII-символ, у встановлену позицію курсору.

Всі функції DOS виводу на екран використовують пристрій STDOUT, стандартний вивід. Це дозволяє перенаправляти вивід програми у файл або на стандартний ввід іншої програми. Наприклад, якщо відкомпілювати наведений приклад (створити файл cod.COM) і написати в командному рядку

cod.COM > cod.out

то на екран нічого видано не буде, а в поточному каталозі з'явиться файл cod.out, що містить ASCII-код символу.

1.2. Функція DOS 06h.

Функція DOS 06h - Записати символ в STDOUT без перевірки на Ctrl-Break

Ввід:	AH = 06h DL = ASCII-код символу (крім FFh)
Вивід:	Ніякого, відповідно до документації, але насправді: AL = код записаного символу (копія DL)

Дана функція не обробляє керуючі символи (CR, LF, HT й BS виконують свої функції при виводі на екран, але зберігаються при перенаправленні виводу у файл) і не перевіряє натискання Ctrl-Break.

Замінимо в прикладі № 1.1 MOV AH,2 на MOV AH,6 і перекомпілюємо даний приклад. Роботу відкомпільованого приклада розглянемо в операційній системі MS-DOS.

1.3. Функція DOS 09h

Функція DOS 09h - Записати рядок в STDOUT з перевіркою на Ctrl-Break

Ввід:	AH = 09h DS: DX = адреса рядка, що закінчується символом $ (24h)
Вивід:	Ніякого, відповідно до документації, але насправді: AL = 24h (код останнього символу)

Дія даної функції повністю аналогічна дії функції 02h, але виводиться не один символ, а цілий рядок (див. лабораторну роботу №2).

1.4 Функція DOS 40h

Функція DOS 40h - Записати у файл або пристрій

Ввід:	AH = 40h BX = 1 для STDOUT або 2 для STDERR DS: DX = адреса початку рядка, CX = довжина рядка
Вивід:	CF = 0, AX = кількість записаних байт

Дана функція призначена для запису у файл, але якщо в регістр BX помістити число 1, функція 40h буде виводити дані на STDOUT, а якщо BX = 2 – на пристрій STDERR. STDERR завжди виводить дані на екран і не перенаправляється у файли. На використанні цієї функції засновані використовувані в C функції стандартного виводу – фактично функція C fputs() просто викликає дане переривання, поміщаючи свій перший аргумент у BX, адреса рядка (другий аргумент) - в DS: DX і довжину - у CX.

Простий приклад роботи функції DOS 40h.

Приклад № 1.2

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ah,40h	; номер функції DOS
mov bx,2	; указуємо пристрій STDERR
mov dx,offset message	; DS: DX - адреса рядка
mov cx,25	; CX - довжина рядка
int 21h	; виклик DOS
ret	; функція DOS "завершити програму"
message db "This function can print $"	; строка, що містить виведені дані.
end begin	; мітка закінчення коду програми

Якщо скомпілювати даний приклад і запустити її командою

dosout.COM > dosout. out

то повідомлення з'явиться на екрані, а файл dosout2.out виявиться порожнім.

Переривання INT 29H

INT 29h: Швидкий вивід символу на екран

Ввід:	AL = ASCII-код символу

Простий приклад роботи переривання INT 29h.

Приклад № 1.3

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ax, < ASCII-код символу >	; заносимо в регістр ax - будь-який ASCII-код символу
int 29h	; виклик переривання DOS - виклик символу;
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

У більшості випадків INT 29h негайно викликає функцію BIOS "вивід символу на екран у режимі телетайпа", таким чином ніяких переваг, крім економії байт при написанні якомога коротших програм, вона не має.

2. Засоби BIOS

Функції DOS виводу на екран дозволяють перенаправляти вивід у файл, але не дозволяють вивести текст у будь-яку позицію екрана й не дозволяють змінити кольори тексту. DOS припускає, що для більш детальної роботи з екраном програми необхідно використати відеофункції BIOS. BIOS - забезпечує доступ до деяких пристроїв, зокрема до відеоадаптера. Всі функції відеосервісу BIOS викликаються шляхом переривання 10h.

2.1 Вибір відеорежиму

BIOS надає можливість перемикання екрана в різні текстові та графічні режими. Режими вирізняються між собою дозволом (для графічних) та кількістю рядків і стовпців (для текстових), а також кількістю можливих кольорів.

2.1.1. Стандартні відеорежими

INT 10h, AH = 00 - Установити відеорежим

Ввід:	AL = номер режиму, в молодших 7 бітах
Вивід:	Зазвичай ніякого, але деякі BIOS (Phoenix й AMI) поміщають в AL 30H для текстових режимів й 20h для графічних

Приклад роботи.

Приклад № 2.1

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ah,00	; установлюємо відеорежим
mov al,5	; установлюємо номер режиму
int 10h	; виклик переривання DOS - виклик відеосервісу;
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

Виклик даної функції приводить до того, що екран переводиться в обраний режим. Якщо старший біт AL не встановлений в 1, екран очищається. Номера текстових режимів - 0, 1, 2, 3 та 7. 0 та 1 - 16-кольорові режими 40x25 (з 25 рядками по 40 символів у рядку), 2 та 3 - 16-кольорові режими 80x25, 7 - монохромний режим 80x25. Існує ще багато текстових режимів з більшою роздільною здатністю (80x43, 80x60, 132x50 і т.д.), але їх номери для виклику через дану функцію різні для різних відеоадаптерів (наприклад, режим 61h - 132x50 для Cirrus 5320 й 132x29 для Genoa 6400). Однак, якщо відеоадаптер підтримує стандарт VESA BIOS Extention, у режими з високою роздільною здатністю можна перемикатися, використовуючи функцію 4Fh.

2.1.2. SuperVGA-відеорежим

INT 10h, AH = 4Fh, AL = 02 - Установити SuperVGA-відеорежим

Ввід:	BX = номер режиму в молодших 13 бітах
Вивід:	AL = 4Fh, якщо ця функція підтримується AH = 0, якщо переключення відбулося успішно AH = 1, якщо відбулася помилка

Якщо біт 15 регістра BX установлений в 1, відеопам'ять не очищається. Текстові режими, які можна викликати з використанням цієї функції: 80x60 (режим 108h), 132x25 (109h), 132x43 (10Ah), 132x50 (10Bh), 132x60 (10Ch).

Відеорежим, що використовується в DOS за замовчуванням - текстовий режим 3.

2.2 Керування положенням курсору

2.2.1. Встановлення положення курсору

INT 10h, AH = 02 - Встановити положення курсору

Ввід:	AH = 02 BH = номер сторінки DH = рядок DL = стовпець

Приклад роботи.

Приклад № 2.2.1

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ah,02	; встановити положення курсору
mov bh,0	; встановити номер сторінки
mov dh,12	; рядок 12
mov dl,29	; стовпець 29
int 10h	; переривання DOS – встановити положення курсору в точку 12, 29
mov ax, < ASCII-код символу >	; занести в регістр ax будь-який ASCII-код символу
int 29h	; виклик переривання DOS - виклик символу;
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

За допомогою даної функції можливо встановити курсор у будь-яку позицію екрана, і подальший вивід тексту буде походити із встановленої позиції. Відлік номера рядка й стовпця ведеться від верхнього лівого кута екрана (символ у лівій верхній позиції має координати 0, 0). Номера сторінок 0 - 3 (для режимів 2 й 3) і 0 - 7 (для режимів 1 й 2) відповідають області пам'яті, зміст якої в цей момент відображається на екрані. Можливо вивести текст у неактивну в даний момент сторінку, а потім перемкнутися на неї, щоб зображення змінилося миттєво.

2.2.2. Зчитування положення й розмір курсору

INT 10h, AH = 03 - Зчитати положення й розмір курсору

Ввід:	AH = 03 BH = номер сторінки
Вивід:	DH, DL = рядок і стовпець поточної позиції курсору CH, CL = перший й останній рядки курсору

Повертає поточний стан курсору на обраній сторінці (кожна сторінка використає власний незалежний курсор).

2.3 Вивід символів на екран

Кожен символ на екрані описується двома байтами - ASCII-кодом символу й байтом атрибута, що вказує кольори символу й фону, а також чи є символ мигаючим.

Атрибут символу.

Біт 7: символ мигає (за замовчуванням) або фон яскравого кольору (якщо його дія була перевизначена відеофункцією 10h).

Біти 6 - 4: кольори фону. Біт 3: символ яскравого кольору (за замовчуванням) або фон мигає (якщо його дія була перевизначена відеофункцією 11h).

Біти 2 - 0: кольори символу.

Кольори кодуються в бітах, як показано в таблиці №2.3.

Таблиця №2.3 Атрибути символів

	Звичайні кольори	Яскравий колір
000b	чорний	темно-сірий
001b	синій	світло-синій
010b	зелений	світло-зелений
011b	блакитний	світло-блакитний
100b	червоний	ясно-червоний
101b	пурпурний	світло-пурпурний
110b	коричневий	жовтий
111b	світло-сірий	білий

2.3.1. Зчитування символу і атрибуту символу в поточній позиції курсору.

INT 10h, AH = 08 - Зчитати символ й атрибут символу в поточній позиції курсору

Ввід:	AH = 08 BH = номер сторінки
Вивід:	AH = атрибут символу AL = ASCII-код символу

2.3.2. Вивід символу із заданим атрибутом на екран

INT 10h, AH = 09 - Вивести символ із заданим атрибутом на екран

Ввід:	AH = 09 BH = номер сторінки AL = ASCII-код символу BL = атрибут символу CX = число повторень символу

За допомогою даної функції можна вивести на екран будь-який символ, включаючи навіть символи CR й LF, які звичайно інтерпретуються як кінець рядка. У графічних режимах CX не повинен перевищувати число позицій, що залишилося до правого краю екрана.

Приклад роботи.

Приклад № 2.2.1

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.

. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ah,09	; помістити номер функції DOS "вивід рядка (9)" у регістр AH.
mov bh,0	; встановити номер сторінки
mov al, < ASCII-код символу >	; рядок 12; занести в регістр al будь-який ASCII-код символу
mov bl, 00011111b	; атрибут символу (білий на блакитному)
mov cx,555	; встановити в лічильник кількість виведених символів
int 10h	; виклик переривання DOS - виклик символу;
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

2.3.3. Вивід символ з поточним атрибутом на екран

INT 10h, AH = 0Ah - Вивести

Ввід:	AH = 0Ah BH = номер сторінки AL = ASCII-код символу CX = число повторень символу

Дана функція також виводить будь-який символ на екран, але як атрибут символу використається атрибут, що мав символ, що перебував раніше в даній позиції.

2.3.4. Вивід символу в режимі телетайпа

INT 10h, AH = 0Eh - Вивести символ у режимі телетайпа

Ввід:	AH = 0Eh BH = номер сторінки AL = ASCII-код символу

Символи CR (0Dh), LF (0Ah), BEL (7) інтерпретуються як керуючі символи. Якщо текст при записі виходить за межі нижнього рядка, екран прокручується нагору. Як атрибут використовується атрибут символу, що перебував у даній позиції.

2.3.5. Вивід рядку символів із заданими атрибутами

INT 10h, AH = 13h - Вивести рядок символів із заданими атрибутами

Ввід:	AH = 13h AL = режим виводу: біт 0 - перемістити курсор у кінець рядка після виводу, біт 1 - рядок містить не тільки символи, але також й атрибути, так що кожен символ описується двома байтами: ASCII-код й атрибут біти 2 - 7 зарезервовані CX = довжина рядка (тільки число символів) BL = атрибут, якщо рядок містить тільки символи DH, DL = рядок і стовпець, починаючи з яких буде виводитися рядка ES: BP = адреса початку рядка в пам'яті

Функція 13h виводить на екран рядок символів, інтерпретуючи керуючі символи CR (0Dh), LF (0Ah), BS (08) і BEL (07). Якщо рядок підготовлений у форматі символ, атрибут - набагато швидше просто скопіювати її у відеопам'ять.

Функції BIOS зручні для перемикання й налаштування відеорежимів, але часто виявляється, що вивід тексту на екран набагато швидше й простіше виконувати просто копіюванням зображення у відеопам'ять.

3. Пряма робота з відеопам'яттю

Усе, що зображено на моніторі - і графіка, і текст, одночасно є присутнім у пам'яті, вбудованої у відеоадаптер. Для того щоб зображення з'явилося на моніторі, воно повинне бути записане в пам'ять відеоадаптера. Для цього відводиться спеціальна область пам'яті, що починається з абсолютної адреси 0B800h: 0000h (для текстових режимів) і закінчується на 0B800h: FFFFh. Усе, що програми записують в дану область пам'яті, негайно пересилається в пам’ять відеоадаптера. У текстових режимах для зберігання кожного символу використаються два байти: байт з ASCII-кодом символу й байт із його атрибутом, так що за адресою 0B800h: 0000h лежить байт з кодом символу, що перебуває у верхньому лівому куті екрана; за адресою 0B800h: 0001h лежить атрибут цього символу; за адресою 0B800h: 0002h лежить код другого символу у верхньому рядку екрана.

Таким чином, будь-яка програма може вивести текст на екран простою командою пересилання даних, не вдаючись до визову спеціальних функцій DOS або BIOS.

Приклад роботи з відеопам'яттю.

Приклад № 3.1

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov ax,0003h	; відеорежим 3 (очищення екрана)
int 10h	; переривання DOS - очищення екрана;
mov ax,0B800h	; завантажити в сегментний регістр ES число 0B800h
mov es,ax	
mov di,0	; завантажити в регістр DI нуль
mov ah,31	; занести в регістр ah атрибут символу
mov al, < ASCII-код символу >	; занести в регістр al будь-який ASCII-код символу
mov es: [di],ax	; занести за адресою 0B800: 0000h атрибут і ASCII-код символу
mov ah,10h	; викликати функцію 10h - щоб можна було зупинити програму до натискання будь-якої клавіші
int 16h	; виклкати переривання 16h - сервіс роботи із клавіатурою BIOS
ret	; функція DOS "завершити програму"
end begin	; мітка закінчення коду програми

При підготовці даних для копіювання у відеопам'ять у даній програмі в рядках (7) і (8) завантажують в сегментний регістр ES число 0B800h, що відповідає сегменту дисплея в текстовому режимі. У рядку (9) завантажують в регістр DI нуль – зсув щодо сегмента 0B800h. У рядках (10) і (11) у регістр AH заноситься атрибут символу (31 – яскраво-білий символ на синьому фоні) і в AL - ASCII-код символу (01 – «смайл») відповідно.

У рядку (12) заносять за адресою 0B800: 0000h (тобто перший символ у першому рядку дисплея - верхній лівий кут) атрибут й ASCII-код символу (31 й 01 відповідно).

4. Завдання для виконання.

4.1. За допомогою редактора емулятора EMU 8086 написати програми, приклади яких наведені в даній лабораторній роботі.

4.2. Створіть файли типа MZ і *.COM.

4.3. Пояснити структуру відкомпільованих програм.

4.4. Одержати завдання у викладача (один з п'яти варіантів табл. №4.1) написати програму виводу на екран рядка 'Hello'.

Таблиця №4.1

№ вар.	Функція виводу (DOS)	Функція виводу (BIOS)	Відеопам'ять
	02h	Ah=02h	‘Hello’
	06h	Ah=08h	
	09h	Ah=09h	
	40h	Ah=0Ah	
	29h	Ah=13h	

Примітка. У прикладах, у яких можливе задання різних параметрів виводу (кольори символу, фону; номер рядка, стовпця, сторінки й т.д.) вивід на екран слова "hello" виконуються з параметрами відмінними від стандартних.

4.5. Написати програму роботи переключення SuperVGA-відеорежимів (згідно варіантів табл. №4.2)

Таблиця №4.2

№ вар.	SuperVGA-відеорежим
	108h
	109h
	10Ah
	10Bh
	10Ch

4.6. Підготувати звіт, що повинен містити тексти програм, а також вказати опис роботи команд програм.

4.7. Звіт повинен містити відповіді на наступні контрольні питання.

5. Контрольні питання

5.1. Перелічити функції виводу на екран засобами операційної системи DOS.

5.2. Принцип роботи функції DOS 02h.

5.3. Вказати основні керуючі символи виводу на екран.

5.4. Яким чином здійснити вивід програми у файл?

5.5. Пояснити відмінність функції DOS 02h від 06h.

5.6. Переривання int 29h. Переваги використання.

5.7. За допомогою яких функцій можна встановити потрібний відеорежим (текстовий, кольоровий, монохромний)?

5.8. Відзначити основні деталі установки super VGA-відеорежимів.

5.9. Вказати функції й переривання керування положенням курсору.

5.10. Перелічити функції зчитування положення й розміру курсору.

5.11. Вивід символів на екран засобами BIOS.

5.12. Пряма робота з відеопам'яттю. Принципи роботи з відеопам'яттю.

5.13. Вказати переваги виводу на екран за допомогою безпосередньої роботи з відеопам'яттю.

5.14. Область пам'яті відеоадаптера.

5.15. Вказати код третього символу у верхньому рядку екрана для роботи з відеопам'яттю.

5.16. Якщо в прикладі № 1.2 довжину рядка указати більшу, ніж зазначена, що в цьому випадку буде виводитися на екран?

ЛАБОРАТОРНА РОБОТА № 7

КОМАНДИ ЛОГІЧНИХ ОПЕРАЦІЙ

Мета роботи: ознайомитися з роботою команд логічних операцій: and, or, xor, test, not та реалізацією їхньої роботи на практиці.

1. Короткі теоретичні відомості.

Логічні операції є важливим елементом у проектуванні мікросхем і мають багато загального в логіку програмування. Команди AND, OR, XOR й TEST є командами логічних операцій. Дані команди використовують для скидання й установки біт і для арифметичних операцій у коді ASCII. Всі ці команди обробляють один байт або одне слово в регістрі або в пам'яті, і встановлюють прапорці CF, OF, PF, SF, ZF.

Команда AND

Команда AND (Логічне І) здійснює логічне (побітове) множення першого операнда з другим. Вихідне значення першого операнда (приймача) втрачається з заміщенням результатом множення. У якості першого операнда команди AND можна вказувати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обидва операнда одночасно як комірки пам'яті. Операнди можуть бути байтами або словами. Команда впливає на прапорці SF, ZF й PF.

Правила побітового множення:

Перший операнд - біти 0101	Біти результату 0001
Другий операнд - біти 0011	

Приклад 1

```
mov AX,0FFEh
and AX,5555h; AX=0554h
```

Приклад 2

```
mov ax,00101001b
add ax,11110111b ; ax=00100001b
```

Команда OR

Команда OR (Логічне АБО) виконує операцію логічного (побітового) додавання двох операндів. Результат заміщає перший операнд (приймач); другий операнд (джерело) не змінюється. У якості першого операнда можна вказувати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обидва операнди одночасно як комірки пам'яті. Операнди команди OR можуть бути байтами або словами. Команда впливає на прапорці OF, SF, ZF, PF й CF, при цьому прапори CF й OF завжди скидаються в 0.

Правила побітового додавання:

Перший операнд - біти 0101	Біти результату 0111
Другий операнд - біти 0011	

Приклад 1

```
mov AX,000Fh
mov BX,00F0h
or AX,BX; AX=00FFh, BX=00F0h
```

Приклад 2

```
mov AX,00101001b
mov BX,11110111b
or AX,BX ; mov dx,11111111b
```

Приклад 3

```
mov AX,000Fh
or AX,8001h ; AX=800Fh
```

Команда XOR

Команда XOR (Логічне ВИКЛЮЧНЕ АБО) виконує операцію логічного (побітового) виключного АБО над своїми двома операндами. Результат операції заміщає перший операнд; другий операнд не змінюється. Кожен біт результату встановлюється в 1, якщо відповідні біти операндов різні, і скидається в 0, якщо відповідні біти операндов збігаються.

У якості першого операнда команди XOR можна вказувати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обидва операнди одночасно як комірки пам'яті. Операнди можуть бути байтами або словами. Команда впливає на прапори OF, SF, ZF, PF й CF, причому прапорці OF й CF завжди скидаються, а інші прапорці встановлюються залежно від результату.

Правила побітового або XOR:

Перший операнд - біти 0101	Біти результату 0110
Другий операнд - біти 0011	

Приклад 1

```
mov AX,0Fh
xor AX,0FFFFh; AX=FFF0h
```

Приклад 2

```
mov AX,00101001b
mov BX,11110111b
xor ax,bx; 11011110b
```

Приклад 3

```
mov SI,0AAAAh
mov BX,5555h
xor SI,BX ; SI=FFFFh, BX=5555h
```

Приклад 4

```
xor BX, BX ; Обнулення BX
```

Команда TEST

Команда TEST (Логічне порівняння) виконує операцію логічного множення І над двома операндами і, залежно від результату, установлює прапорці SF, ZF й PF. Прапорці OF й CF скидаються, а AF має невизначене значення. Стан прапорців можна потім проаналізувати командами умовних переходів. *Команда TEST не змінює жоден з операндів.*

В якості першого операнда команди TEST можна вказувати регістр (крім сегментного) або комірку пам'яті, у якості другого - регістр (крім сегментного), комірку пам'яті або безпосереднє значення, однак не допускається визначати обидва операнди одночасно як комірки пам'яті. Операнди можуть бути байтами або словами й представляти числа зі знаком або без знака.

Правила побітового множення:

Перший операнд - біти 0101	Біт результату 0001
Другий операнд - біти 0011	

Прапорець SF установлюється в 1, якщо в результаті виконання команди утворилося число із установленим знаковим бітом.

Прапорець ZF установлюється в 1, якщо в результаті виконання команди утворилося число, що лише з двійкових нулів.

Прапорець PF установлюється в 1, якщо в результаті виконання команди утворилося число з парною кількістю двійкових одиниць у його бітах.

Приклад 1

```
test AX,1
jne label2: ; Перехід, якщо біт 0 в AX установлений
je label1: ; Перехід, якщо біт 0 в AX скинутий
```

Даний приклад порівнює два значення (рядок (6)), якщо одне із двох значень дорівнює нулю тоді переходимо на мітку label1 (рядок (10)) далі виконуються команди, що слідують після цієї мітки, у випадку якщо одне із двох значень дорівнює нулю – перехід на мітку label2.

Приклад 2

```
test SI,8
jne bityes ; Перехід, якщо біт 3 в SI установлений
je bitno ; Перехід, якщо біт 0 в AX скинутий
```

Приклад 3

```
test DX,0FFFFh
jz null ; Перехід, якщо DX=0
jnz smth; Перехід, якщо DX не 0
```

Команда NOT

Команда NOT (інверсія, доповнення до 1, логічне заперечення) виконує інверсію бітів зазначеного операнда, заміняючи 0 на 1 і навпаки. У якості операнда можна вказувати регістр (крім сегментного) або комірку пам'яті розміром як у байт, так і у слово. Не допускається використати в якості операнда безпосереднє значення. Команда не змінює прапорці процесора.

Правила побітової інверсії:

Операнд - біти 0 1	Біти результату 1 0

Приклад 1

```
mov AX,0FFFFh
not AX; AX=0000h
```

Приклад 2

```
mov SI,5551h
not SI; SI=AAAEh
```

Характерні приклади роботи команд логічних операцій.

Для наступних незв'язаних прикладів, припустимо, що:

AL містить 1100 0101

BH містить 0101 1100:

1. AND AL,BH; Встановлює в AL 0100 0100
2. OR BH,AL; Встановлює в BH 1101 1101
3. XOR AL,AL; Встановлює в AL 0000 0000
4. AND AL,00; Встановлює в AL 0000 0000
5. AND AL,0FH; Встановлює в AL 0000 0101
6. OR CL,CL; Встановлює прапорці SF й ZF

Приклади 3 й 4 демонструють спосіб очищення регістра. У прикладі 5 обнуляються ліві чотири біти регістра AL.

Можна застосувати команду OR для наступних цілей:

1. OR CX,CX; Перевірка CX на нуль

JZ; Перехід, якщо нуль

2. OR CX,CX; Перевірка знака в CX

JS; Перехід, якщо негативно

2. Завдання для виконання.

Запустити емулятор EMU8086.

Користуючись правилами оформлення асемблерних програм, набрати код прикладу й запустити код на виконання.

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov CX,<число1 >	; завантажити в CX число1 <будь-яке число1>
mov BX,<число2>	; завантажити в BX число2 <будь-яке число2>
test cx,bx	; логічно порівняти числа в регістрах cx та bx
jne label2	; якщо одне зі значень не дорівнює 0, то перейти на мітку label2
je label1	; якщо одне зі значень дорівнює 0, то перейти на мітку label1
ret	; функція DOS "завершити програму" (не виконується)
label1:	; початок блоку мітки Label1
mov ah,9	; помістити номер функції DOS "вивід рядка (9)" у регістр AH.
mov dx,offset string	; поміщає в регістр DX зсув мітки String відносно початку сегмента даних
int 21h	; функція DOS "вивід рядка"
ret	; функція DOS "завершити програму"
String db 'одне із чисел дорівнює 0$'	; строка, що містить виведені дані.
label2:	; початок блоку мітки Label2
mov ah,9	; помістити номер функції DOS "вивід рядка (9)" у регістр AH.
mov dx,offset string1	; помістити в регістр DX зсув мітки String1 відносно початку сегмента даних
int 21h	; функція DOS "вивід рядка"
ret	; функція DOS "завершити програму"
string1 db 'не рівні 0$'	; строка, що містить виведені дані.
end begin	; мітка закінчення коду програми

Проаналізувати роботу коду прикладу. За допомогою довідки емулятора EMU8086 проаналізувати роботу команд (jne, je, js, jz).

Одержати завдання у викладача (один із чотирьох варіантів (команди and, or, xor, test, not) і, користуючись правилами оформлення асемблерних програм, написати три програми що характеризують роботу їх із числами (двійкової, десяткової, шістнадцяткової систем числення) відповідно до наведених прикладів.

Результати роботи продемонструвати викладачеві.

3. Контрольні питання

3.1. Призначення команд логічних операцій.

3.2. Команда and, основне призначення.

3.3. Команда or, основне призначення.

3.4. Команда xor, основне призначення.

3.5. Команда test, основне призначення.

3.6. Команда not, основне призначення.

3.7. Альтернативна робота команд (test, xor, and).

3.8. У чому полягає робота зв'язки команд jne, je?

3.9. У чому полягає робота зв'язки команд js, jz?

3.10. Що відбудеться, якщо в прикладі №1.4.1.1 змінити рядок (8) на наступний (jne label1)?

ЛАБОРАТОРНА РОБОТА №8

ОЗНАЙОМЛЕННЯ З РОБОТОЮ ЦИКЛІВ

Мета роботи: ознайомитися зі структурою й реалізацією циклів у програмі.

1. Короткі теоретичні відомості.

Цикли, що дозволяють виконати деякі ділянки програми багаторазово, у будь-якій мові є однієї з найбільш уживаних конструкцій. У системі команд МП 86 цикли реалізують, головним чином, за допомогою команди loop (петля), хоча є й інші способи організації циклів. У більшості випадків число кроків у циклі визначається вмістом регістра CX, тому максимальне число кроків становить 64К.

Організація циклічних переходів, як на мовах високого рівня, так і мовою assembler являє собою чудовий засіб, що дозволяє значно зменшити код програми.

У загальному виді будь-який цикл записується в асемблері як умовний перехід.

Організація циклу за допомогою команди LOOP (Перший спосіб)

Команда loop (англ. петля) виконує декремент умісту регістра CX (лічильнику), і якщо воно не дорівнює 0, здійснює перехід на зазначену мітку вперед або назад у тому ж програмному сегменті в діапазоні - 128... + 127 байт. Зазвичай мітка міститься перед першою пропозицією тіла циклу, а команда loop є останньою командою циклу. Уміст регістра CX розглядається як ціле число без знака, тому максимальне число повторень групи включених у цикл команд становить 65536 (якщо перед входом у цикл CX=0). Команда не впливає на прапорці процесора.

Команда	Призначення	Процесор
LOOP мітка	Організація циклів	8086

Найпростіший приклад організації циклічного переходу (з лічильником у регістрі cx) мовою Assembler:

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить і дані.
org 100h	; початок COM-файлу

begin:	; мітка початку коду програми
mov cx,10	; завантажити в (регістр-лічильник) CX кількість повторів (відлік буде йти від 10 до 0)
Label1:	; створити мітку (Label - мітка).
mov ah,9	; помістити номер функції DOS "вивід рядка (9)" у регістр AH.
mov dx,offset String	; помістити в регістр DX зсув мітки String відносно початку сегмента даних
int 21h	; функція DOS "вивід рядка"
loop Label1	; оператор loop зменшує на одиницю CX й, якщо він не дорівнює нулю, переходить на мітку Label1 (рядок 6)
ret	; функція DOS "завершити програму"
string db 'privet $'	; строка, що містить виведені дані.
end begin	; мітка закінчення коду програми

У рядку (5) завантажуємо в CX кількість повторів (відлік буде йти від 10 до 0). У рядку (6) створюємо мітку (Label - мітка). Далі (рядки (7) - (9)) виводимо повідомлення. І в рядку (10) оператор loop зменшує на одиницю CX і, якщо він не дорівнює нулю, переходить на мітку Label1 (рядок (6)). Таким чином, рядок буде виведений на екран десять разів. Коли програма перейде на рядок (11), регістр CX буде дорівнює нулю.

Організація циклу за допомогою команди JMP (Другий спосіб)

Команда jmp передає керування в зазначену крапку того ж або іншого програмного сегмента. Адреса повернення не зберігається. Команда не впливає на прапорці процесора.

Команда jmp має п'ять різновидів:

перехід прямий короткий (у межах - 128... + 127 байтів);

перехід прямий ближній (у межах поточного програмного сегмента);

перехід прямий далекий (в інший програмний сегмент);

перехід непрямий ближній;

перехід непрямий далекий.

Всі різновиди переходів мають ту саму мнемоніку jmp, хоча коди операцій відрізняються. У багатьох випадках транслятор може визначити вид переходу по контексту, у тих же випадках, коли це неможливо, варто використати атрибутні оператори (short - прямій короткий перехід; near ptr - прямій ближній перехід; far ptr - прямій далекий перехід; word ptr - непрямий ближній перехід; dword ptr - непрямий далекий перехід).

Команда	Призначення	Процесор
JMP мітка	Безумовний перехід	8086

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
label1:	; створити мітку
mov ah,9	; поміщаємо номер функції DOS "вивід рядка (9)" у регістр AH.
mov dx,offset String	; помістити в регістр DX зсув мітки String відносно початку сегмента даних
int 21h	; функція DOS "вивід рядка"
jmp Label1	; перехід на рядок з міткою Label1
add cx,12	; додати до значення регістра cx число 12 (дана команда не виконується)
dec cx	; зменшити значення регістра cx на 1 (дана команда не виконується)
ret	; функція DOS "завершити програму"
string db "PRIVET",13,10,'$'	; строка, що містить виведені дані.
end begin	; мітка закінчення коду програми

У результаті роботи програми буде зациклений блок рядків (6) - (10) (Вивід рядка PRIVET багато разів). Рядки (10) - (11).

Організація циклу за допомогою команд DEC й JNZ (Третій спосіб)

З допомогу цих операторів можна створювати цикли, які будуть працювати швидше оператора Loop. Комбінована робота команд DEC й JNZ зменшує вміст регістра CX на 1 і виконує перехід на мітку, якщо в CX не дорівнює нулю.

Команда DEC, крім того, установлює прапорець нуля у регістрі прапорців в стан 0 або 1. Команда JNZ потім перевіряє дану установку.

Аналогічно командам JMP й LOOP операнд у команді JNZ містить значення відстані між кінцем команди JNZ й адресою переходу (Label1), що додається до командного покажчика. Ця відстань повинна бути в межах від -128 до +127 байт.

Наступний приклад буде працювати так само, як і Приклад №1.1, але швидше.

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov cx,10	; завантажити в (регістр-лічильник) CX кількість повторів (відлік буде йти від 10 до 0)
Label1:	; створити мітку (Label - мітка).
mov ah,9	; помістити номер функції DOS "вивід рядка (9)" у регістр AH.
mov dx,offset String	; помістити в регістр DX зсув мітки String відносно початку сегмента даних
int 21h	; функція DOS "вивід рядка"
dec cx	; оператор DEC зменшує на одиницю CX й, якщо він не дорівнює нулю, переходить на мітку Label1
jnz Label1	; умовний перехід на рядок з міткою Label1
ret	; функція DOS "завершити програму"
string db 'priver ',13,10, '$'	; строка, що містить виведені дані
end begin	; мітка закінчення коду програми

2. Завдання для самостійної роботи.

Написати програму, що виводить на екран всі ASCII-символи (16 рядків по 16 символів у рядку).

. model tiny	; модель пам'яті в якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov cx,256	; задаємо значення лічильника (256 символів)
mov dl,0	; перший символ - з кодом 00
mov ah,2	; номер функції DOS "вивід символу"
cloop: int 21h	; виклик DOS
inc dl	; збільшення DL на 1 - наступний символ
test dl,0Fh	; якщо DL не кратний 16
jnz continue_loop;	; продовжити цикл,
push dx	; інакше: зберегти поточний символ
mov dl,0Dh	; вивести CR
int 21h	; виклик DOS
mov dl,0Ah	; вивести LF
int 21h	; виклик DOS
pop dx	; відновити поточний символ
continue_loop:	; мітка
loop cloop	; продовжити цикл
ret	; завершення COM-файлу
end begin	; мітка закінчення коду програми

В даному прикладі за допомогою команди LOOP оформляється цикл, що виконується 256 разів (значення регістра CX на початку циклу). Регістр DL містить код символу, що дорівнює нулю на початку циклу й збільшується щораз на 1 командою INC DL. Якщо значення DL відразу після збільшення на 1 є кратним 16, воно тимчасово зберігається в стеку й на екран виводяться символи CR й LF, що виконують перехід на початок нового рядка. Перевірка виконується командою TEST DL,0Fh - результат операції AND над DL й 0Fh

буде нулем, тільки якщо молодші чотири біти DL дорівнюють нулю, що й відповідає кратності шістнадцяти.

3. Зміст звіту.

3.1. Титульний аркуш.

3.2. Індивідуальний варіант завдання.

3.3. Тестові набори даних і передбачувані результати.

3.4. Текст програми до налагодження.

3.5. Список помилок, виявлених при налагодженні.

3.6. Результати виконання тестів.

3.7.Роздруківка лістингу компіляції налагодженої програми із коментарем роботи кожного рядка.

4. Завдання для виконання.

4.1. Виконати всі приклади, що містяться в описі даної лабораторної роботи.

4.2. Проаналізувати роботу програми прикладу для практики.

4.3. Вивчити умови організації циклічних переходів мовою Асемблера.

4.4. Написати програму, що виводить на екран слово "!!!!!!!!!! Hello!!!!!!!!!!" використовуючи команди циклічних переходів (3 варіанти).

4.5. Одержати завдання у викладача (один з п'яти варіантів табл. №4.1) і, користуючись правилами оформлення асемблерних програм, створити програму, що виводить на екран слово D разів.

4.6. Програму асемблювати у файл типу *.COM або *. exe (на вибір).

Таблиця №4.1

№ вар.	Виведені дані	Формула розрахунку	A	B	C
	Циклічний перехід	D=A+B+C	101	345	121

№ вар.	Виведені дані	Формула розрахунку	A	B	C
	Hello world	D=A-B+C	578	152	149
	Good Bye	D=A+B-C	333	223	16
	Група	D=A-B+C	1502	834	1
	Лабораторна робота	D=A-B-C	1056	33	125

5. Контрольні питання

5.1. Організація циклу за допомогою команди loop.

5.2. Значимість регістра cx.

5.3. Максимальне число повторень команд циклу обумовленого регістром cx.

5.4. Організація циклу за допомогою команди jmp.

5.5. Різновиду команди jmp.

5.6. Організація циклу за допомогою команд dec й jnz.

ЛАБОРАТОРНА РОБОТА №9

СПОСОБИ Й МЕТОДИ ВИВОДУ ЧИСЕЛ

Мета роботи: засвоїти методи виводу чисел у двійковому, шістнадцятирічному і десятковому коді.

1. Короткі теоретичні відомості

1.1. Вивід двійкового коду числа, записаного в регістр DH.

Методика виконання.

Послідовно проаналізувати біти числа. У даній роботі обмежимося байтом, що будемо зберігати в DH. Якщо біт нульової (скинутий), то потрібно вивести '0', якщо встановлений, то '1'.

Алгоритм вирішення завдання:

- ✓ проаналізувати значення одного біта;
- ✓ вивести значення біта;
- ✓ перейти до наступного біта. І так 8 разів – в циклі.

Аналіз біта: при аналізі значення програмісти звичайно використають команду TEST, але в даній лабораторній роботі буде використано наступну SHL.

Команда SHL здійснює зсув вліво всіх бітів операнда. Старший біт операнда надходить у прапор CF. Якщо команда записана у форматі

SHL операнд, 1

зсув здійснюється на 1 біт. У молодший біт операнда завантажується 0. Якщо команда записана у форматі

SHL операнд, CL

зсув здійснюється на число бітів, зазначене в регістрі-лічильнику CL, при цьому в процесі послідовних зрушень старші біти операнда, пройшовши через прапорець CF, втрачаються, а молодші заповнюються нулями.

У якості операнда команди SHL можна вказувати будь-який регістр (крім сегментного) або комірку пам'яті розміром, як у байт, так й у слово. Не допускається використати в якості операнда безпосереднє значення.

Кожийе зсув вліво еквівалентний множенню знакового числа на 2, тому команду SHL зручно використати для зведення операнда в ступінь 2.

Команда впливає на прапорці OF, SF, ZF, PF й CF.

Реакцію на значення прапорця можна отримати за допомогою команди

JNC <мітка>

Здійснюється перехід на мітку, якщо прапор CF дорівнює нулю, інакше виконується команда, що слідує безпосередньо після команди.

1.2. Вивід значення байта в шістнадцятковій системі числення

Алгоритм вирішення завдання.

Припустимо, що байт, значення якого потрібно вивести, перебуває в регістрі DH, і є таблиця символів "0123456789ABCDEF". Байт складається із двох шістнадцяткових цифр. З врахуванням цього завдання можна вирішити так: потрібно вивести на екран два символи із цієї таблиці. Спочатку - символ з номером, що дорівнює старшому напівбайту числа, а потім з номером, що дорівнює молодшому напівбайту.

Для вирішення завдання потрібно виконати два етапи:

- записати в AL символ з потрібним номером. Скористатися регістровим не прямим режимом адресації зі зміщенням. Для цього значення кожного напівбайта необхідно записувати в BX;

- записати в BX значення напівбайта.

2. Завдання для виконання.

Запустити емулятор EMU8086.

Користуючись правилами оформлення асемблерних програм, набрати код прикладу і запустити на виконання.

2.1. Вивід двійкового коду числа, записаного в регістр DH.

Приклад №1.1

. model tiny	; модель пам'яті, у якій об'єднані сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.
org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov DH,<55>	; занести в регістр dh будь-який ASCII-код символу
mov AH,2	; помістити номер функції DOS "вивід рядка (2)" у регістр AH.
mov CX,8	; ініціалізація змінної циклу
@1: mov DL,'0'	; занести в DL код символу '0'
shl DH, 1	; зрушення на 1 біт
jnc @2	; перехід, якщо '0'
inc DL	; використати той факт, що код символу '1' на одиницю більше коду символу '0'
@2: int 21h	; виклик переривання DOS - виклик символу;
LOOP @1	; переходимо на мітку @1
int 21h	; виклик переривання DOS - виклик символу;
end begin	; мітка закінчення коду програми

Примітка: в описаному способі аналізу значення вихідного числа втрачається. Інакше варто використати команду ROL, але 8 разів для байта.

2.2. Вивід значення байта в шістнадцятковій системі числення

Приклад №2.1

. model tiny	; модель пам'яті, у об'єднані якій сегменти коду, даних і стека.
. code	; сегмент коду, що містить дані.

org 100h	; початок COM-файлу
begin:	; мітка початку коду програми
mov dh, 10	; занести в регістр dh число 10
mov bl, dh	; занести в регістр bl число 10
xor bh, bh	; обнулення вх
and bl, 0F0h	; здійснити логічне (побітове) множення dh на 0f0h.
shr bl, 4	; зсув вправо на 4 біти
mov al, table [bx]	; занести в регістр al значення рядка даних
int 29h	; виклик переривання DOS - виклик символу
mov bl, dh	; занести в регістр bl значення регістра dh
and bl, 0Fh	; логічне (побітове) множення dh на 0fh.
mov al, table [bx]	; заносимо в регістр al значення рядка даних
int 29h	; виклик переривання DOS - виклик символу;
mov al, 13	; занести в регістр al число 13
int 29h	; виклик переривання DOS - виклик символу;
mov al, 10	; занести в регістр al число 10
int 29h	; виклик переривання DOS - виклик символу
ret	; функція DOS "завершити програму"
table db '0123456789ABCDEF'	; строка, що містить виведені дані.
end begin	; мітка закінчення коду програми

3. Завдання для виконання.

3.1. Виконати всі приклади, що містяться в описі даної лабораторної роботи.

3.2. Проаналізувати роботу програми приклада для практики.

3.3. Вивчити умови організації переривань мовою Асемблера.

3.4. Програму скомпілювати у файл типу *.COM або *. exe (на вибір).

4. Контрольні питання

4.1. Організація переривань за допомогою команди int.

4.2. Значимість регістра dh.

4.3. Функція завершення програми переривання ret.

4.4. Використання команди ROL.

4.5. Використання команди SHR.

ЛАБОРАТОРНА РОБОТА №10

ПРОГРАМНА СИМУЛЯЦІЯ РОБОТИ МІКРОКОНТРОЛЕРІВ

Мета роботи: засвоїти основні прийоми складання принципових схем та прикладних програм мікроконтролера Intel 8051 в програмному середовищі Multisim.

1. Завдання для виконання роботи

Скласти схему підключення семисегментного індикатора до мікроконтролера 8051 в програмному середовищі Multisim. Написати та протестувати програму для обчислення арифметичного виразу:

$$Y = (X1+X2) * (X3–X4) / 2$$

2. Опис процесу складання схем та створення програм в середовищі імітації роботи електронних схем Multisim.

Запускаємо програму Multisim (рис. 2.1):

меню Пуск -> Программы -> National Instruments -> Circuit Design Suite 11.0 -> Multisim 11.0

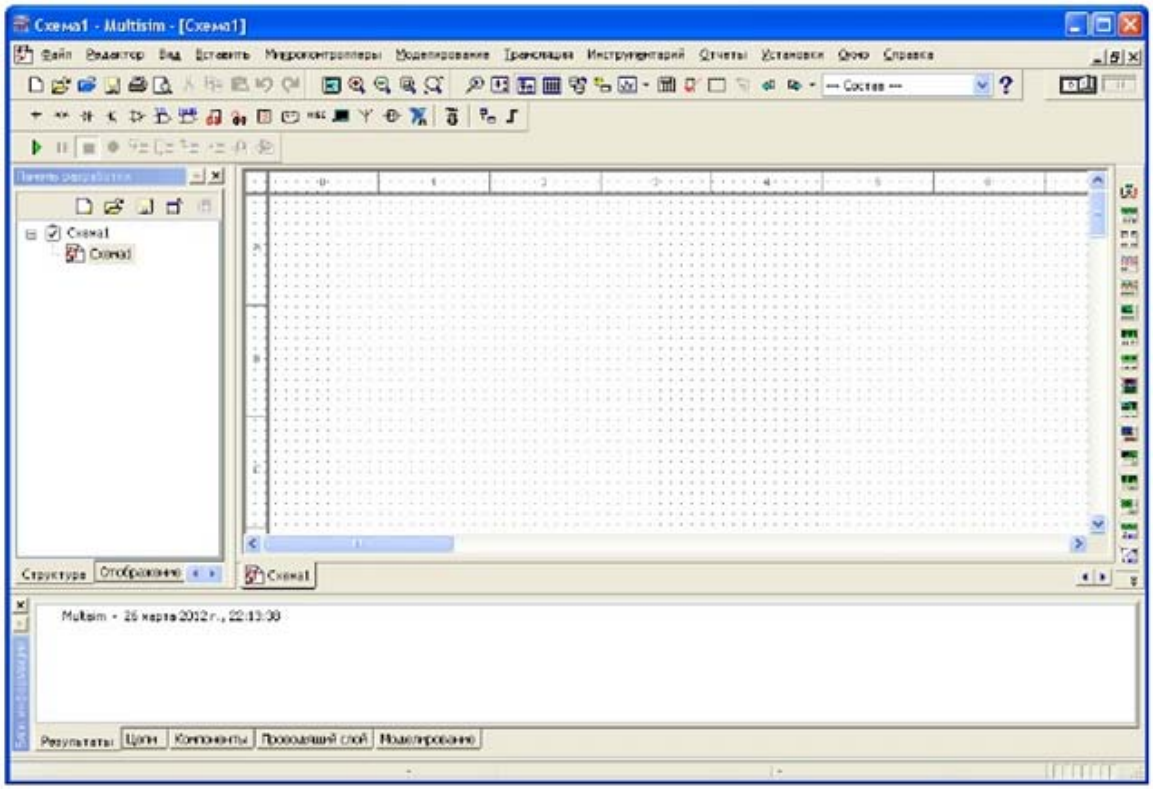


Рис.2.1 Загальний вигляд робочого простору

На вкладці компонент бібліотеки натискаємо на кнопку “Микроконтроллеры”

Вибираємо компонент 8051, як зображено на рис. 2.2.

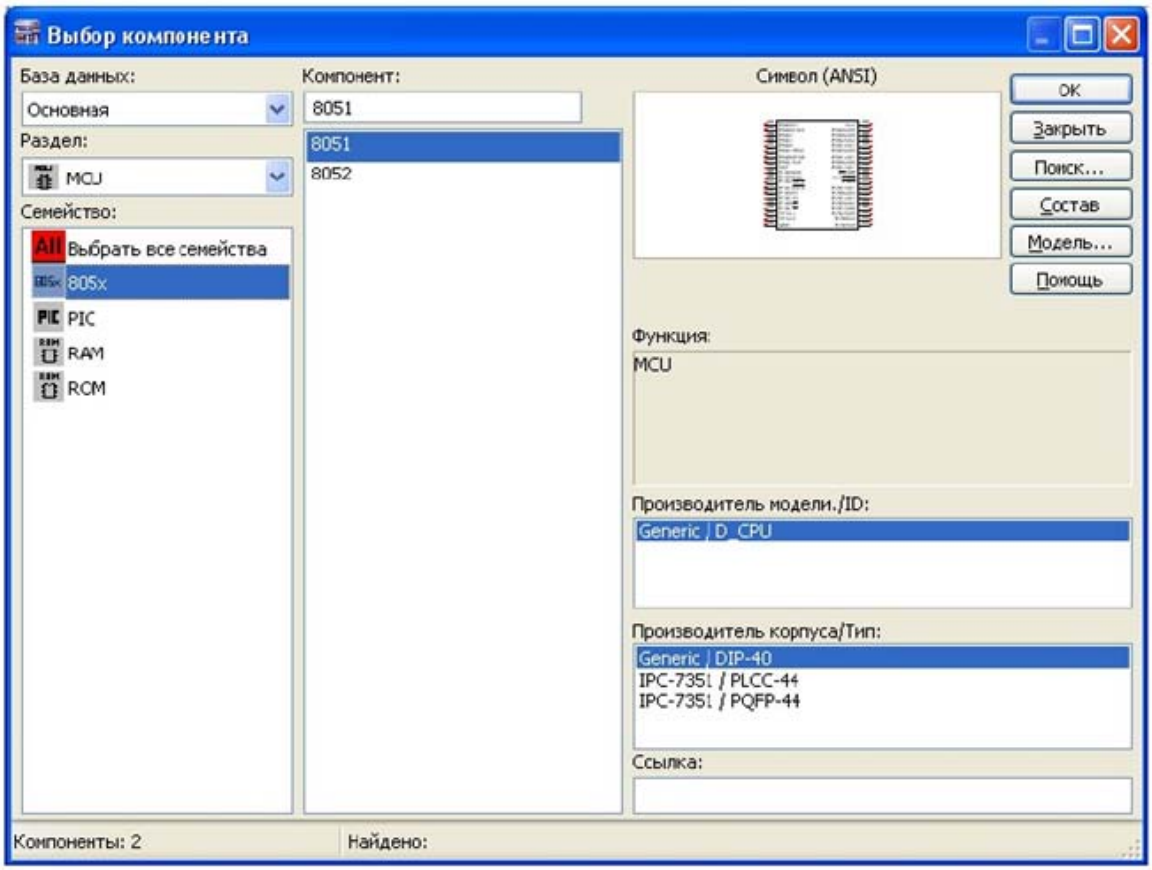


Рис. 2.2 Вибір мікроконтролера із бібліотеки

Натискаємо ОК, ставимо компонент в будь-яке місце на білому полі. Коли компонент встановили, з’являється наступне діалогове вікно, див. рис. 2.3.

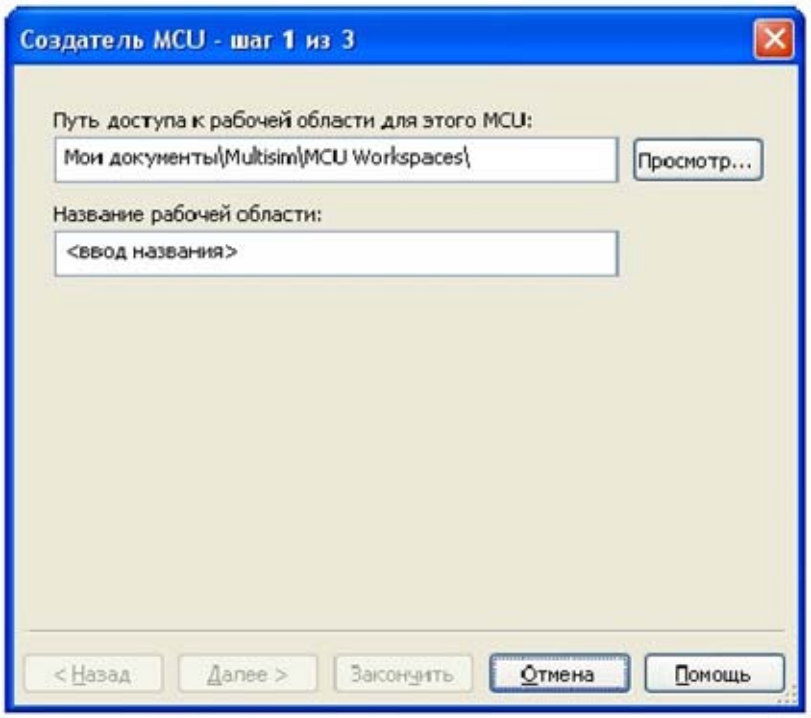


Рис. 2.3 Вікно №1створювача мікроконтролера

Вибираємо шлях де збережемо робочу область та її назву латинськими літерами і натискуємо кнопку “Далее >”. З’являється наступне вікно, див. рис. 2.4.

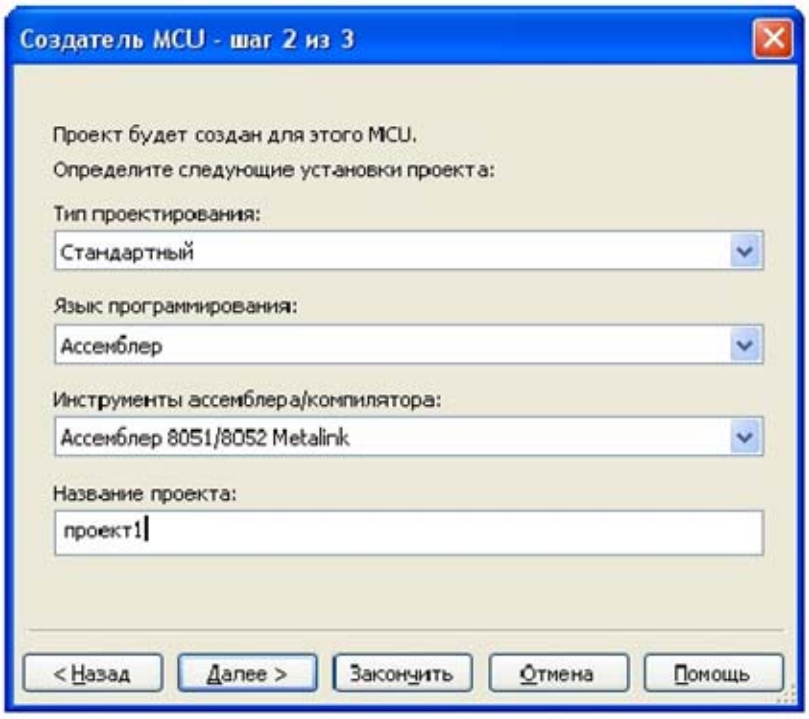


Рис. 2.4 Вікно №2 створювача мікроконтролера

В списку “Язык программирования” вибираємо “Ассемблер”, задаємо назву проекту та натискаємо на клавішу “Далее >”, з’являється наступне вікно, див. рис. 2.5.

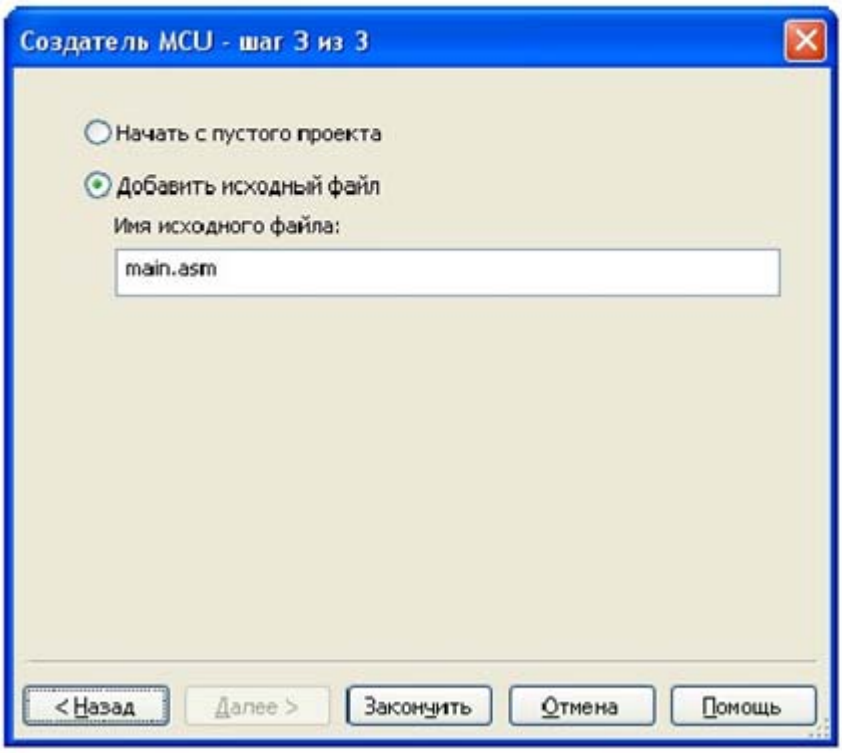


Рис. 2.5 Вікно №3 створювача мікроконтролера

Залишаємо ім’я файлу по замовченню та натискаємо кнопку “Закончить”.

Добавити до мікроконтролера живлення та аналогову землю, як зображено на рис. 2.6.

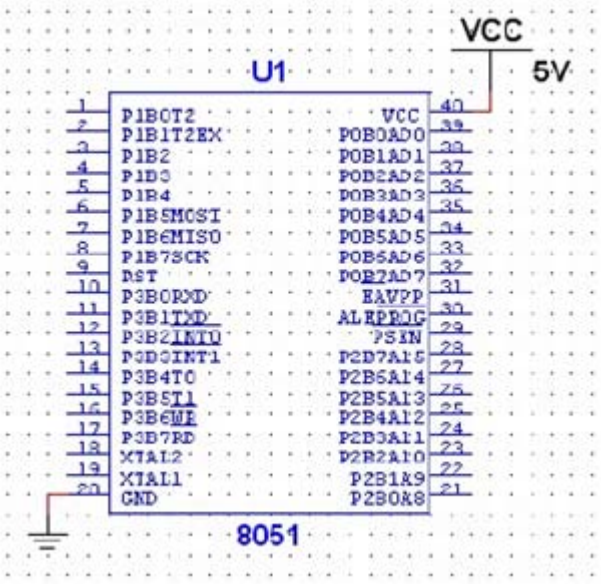


Рис. 2.6. Загальний вигляд мікроконтролера на принциповій схемі

Для цього вибираємо компоненти VCC та GROUND в розділі Sources в вікні “Выбор компонентов” або кнопку “Источники” на панелі інструментів, див. рис.2.7.

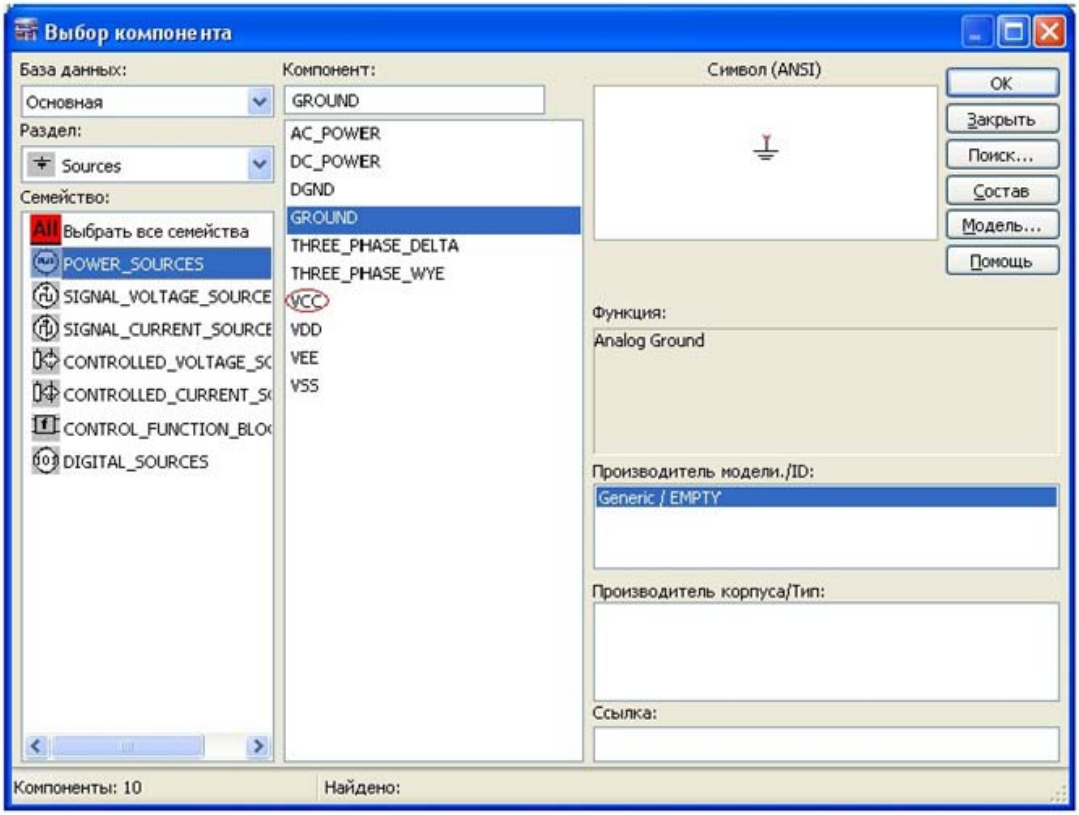


Рис. 2.7. Вибір із бібліотеки елементів живлення

Під’єднаємо до порту P0 мікроконтролера семисегментні індикатори, як показано на схемі, див. рис 2.8.

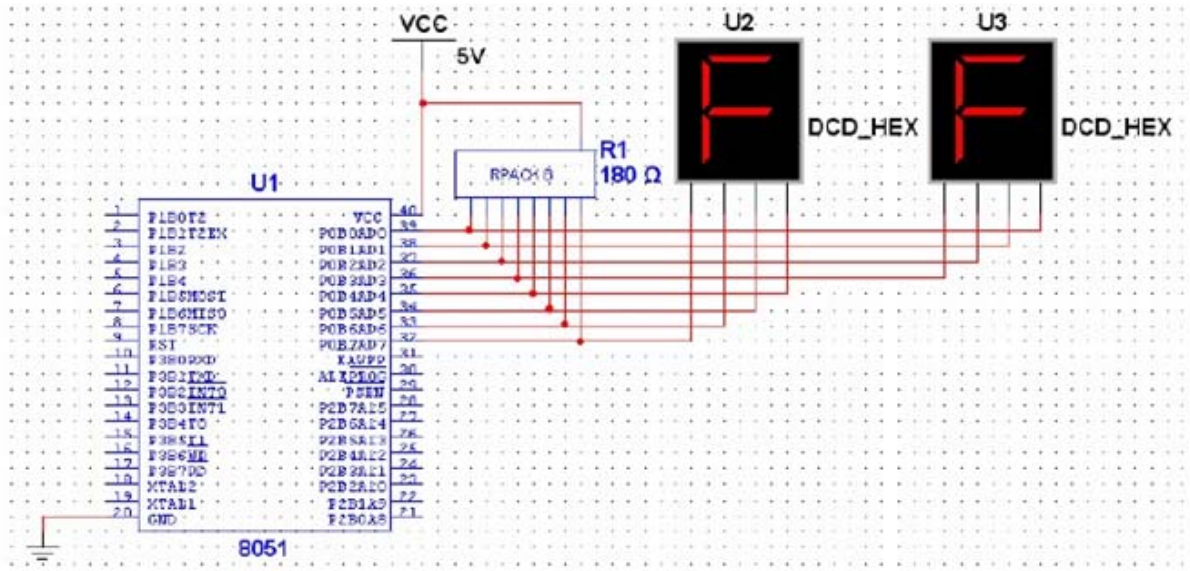


Рис. 2.8. Принципова схема

Індикатори вставляємо, натиснувши на кнопку “Индикаторы” на панелі.

Вибираємо їх в меню, як зображено на рис. 2.9.

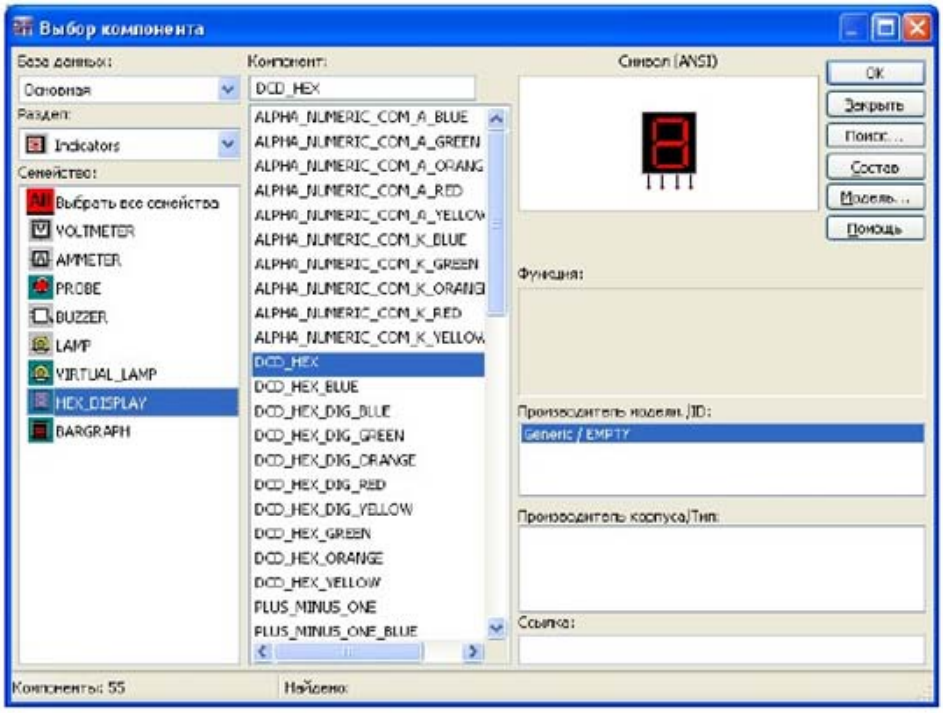


Рис. 2.9. Вибір із бібліотеки індикатора HEX

Магазин резисторів RPACK8 вставляємо натиснувши на кнопку “Пассивные компоненты” на панелі меню

Вибираємо їх в меню, як зображено на рис. 2.10.

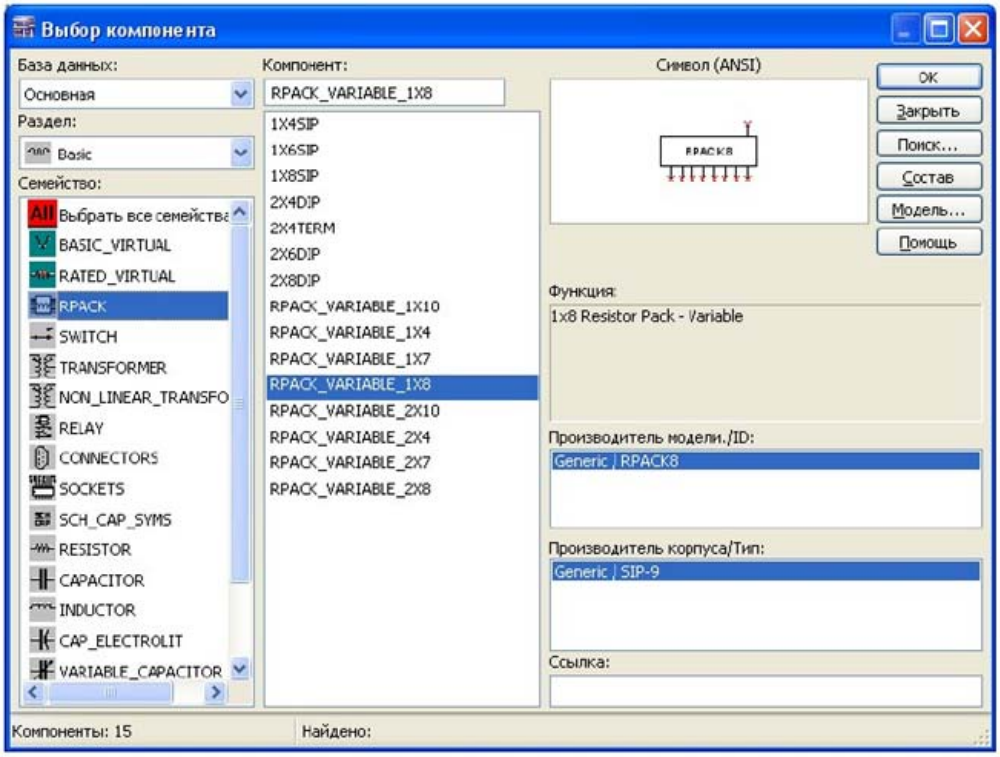


Рис. 2.10. Вибір із бібліотеки індикатора HEX

Далі напишемо текст програми на мові асемблера.

Спочатку розмістимо операнди в резидентній пам’яті даних:

Операнд	Адреса	Початкове значення
X1	30h	5
X2	35h	1
X3	40h	6
X4	45h	2
Y	50h	-

В лівій частині вікна знаходиться “Панель разработки”.

В “Панель разработки” представлено дерево проекту, див. рис. 2.11.

Рис. 2.11. Вікно дерева проекту МК

Відкриваємо файл main.asm в дереві проекту. Код програми буде записаний між директивою $MOD51 та END:

```
$MOD51 ; This includes 8051 definitions for the Metalink assembler
; Please insert your code here.
; в цьому місці має бути програмний код
END
```

Вікно пам’яті має вигляд, як на рис 2.12.

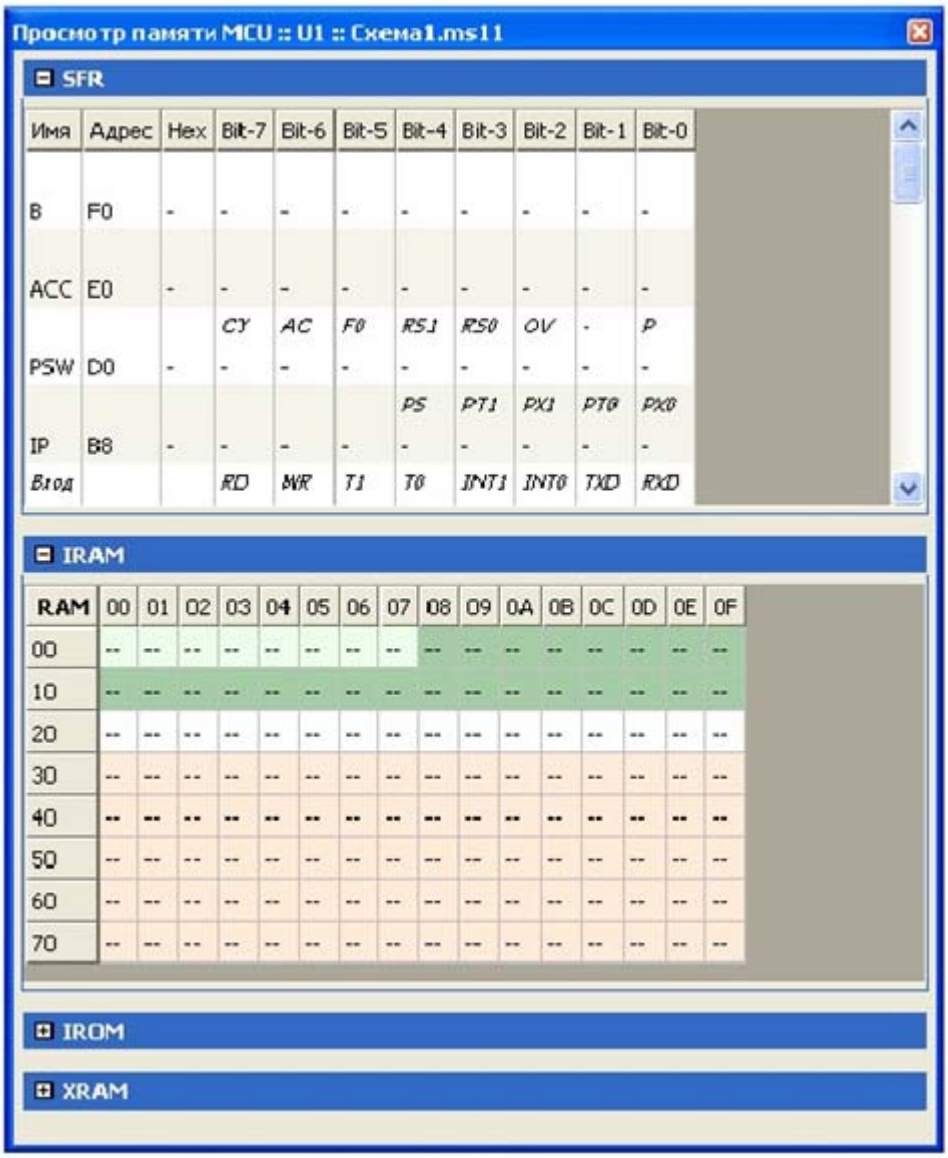


Рис. 2.12. Вікно пам'яті МК

В відкритій вкладці SFR можливо продивитися стан акумулятора, інших регістрів та портів, а в відкритій вкладці IRAM – значення в комірках пам'яті на кожному кроці програми.

Покрокове виконання програми здійснюється натисканням на панелі "Моделирования" кнопку "Вход" :

При покроковому виконанні програми з’являється вікно “Отладка (U1)” в якому жовта стрілка показує, яка команда програми буде зараз виконуватися.

При натисканні кнопки, ця команда виконується.

При цьому стрілка переходить на наступний рядок програми, див. рис. 2.13.

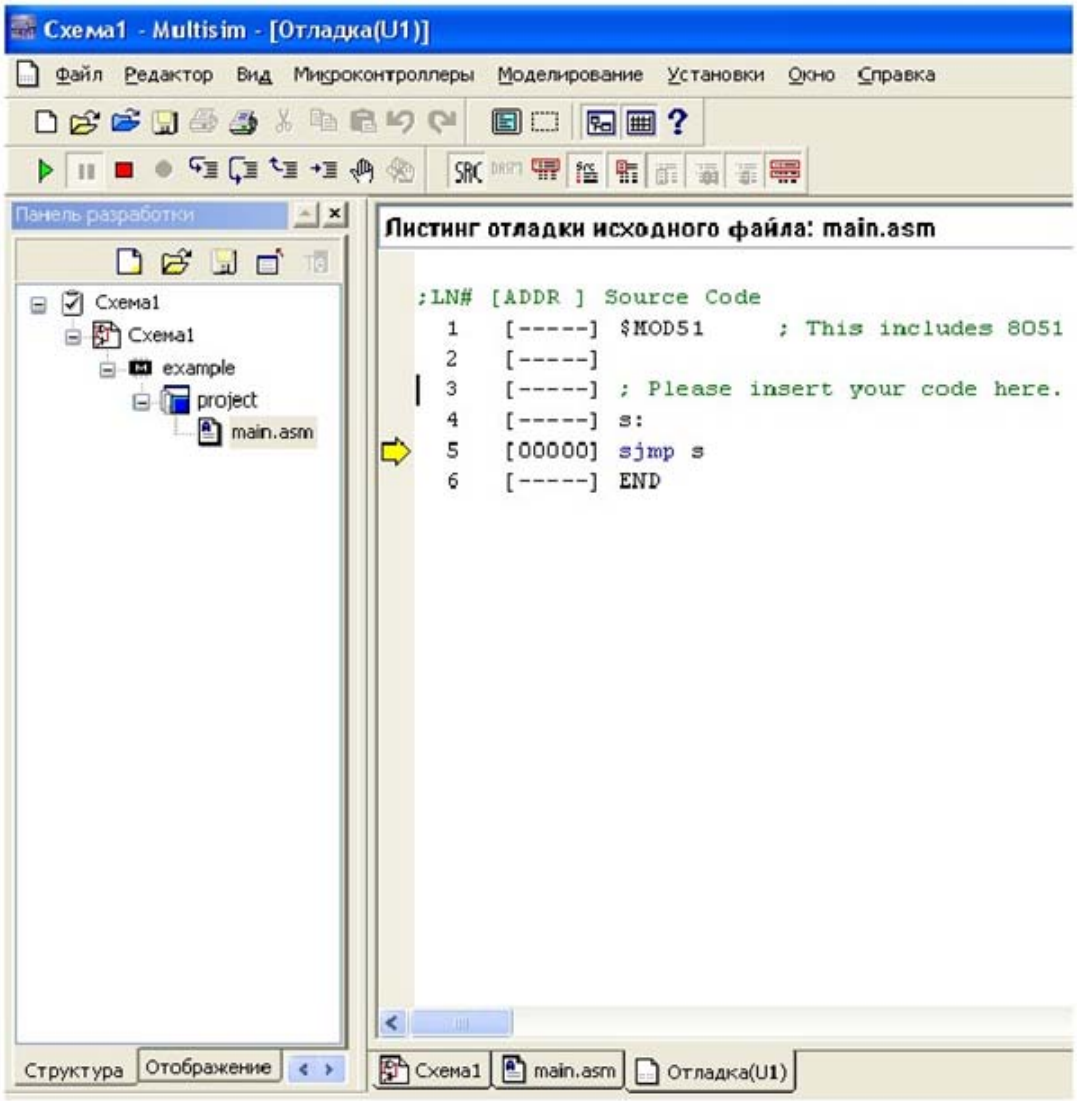


Рис. 2.13. Вікно відлагодження програми МК

3. Порядок виконання роботи

1. Запустити програму Multisim.

2. Розмістити на схемі компонент 8051.

3. Задати шлях на диску до робочої області та її назву. В якості мови програмування задати «Ассемблер». Задати назву проекту та ім’я вихідного файлу.

4. Скласти електронну схему, яку описано вище.

5. Скласти програму на мові асемблера. Розмістити операнди в резидентній пам’яті даних та код програми в пам’яті програм.

6. Підготувати середовище для налагодження програми - розташувати на екрані вкладки SFR та IRAM, скомпілювати програму, переконатися у відсутності помилок.

7. Виконати програму крок за кроком з початку до кінця. Після кожної команди відмічати значення регістрів, що змінилися.

8. Змінити початкові значення змінних та повторити ще двічі попередній пункт.

9. Оформити звіт про роботу.

4. Вміст звіту про роботу

1. Назва, мета та завдання на виконання роботи.

2. Зображення мікроконтролера 8051 та схема підключення до нього семисегментного індикатора.

3. Лістинг програми мікроконтролера.

4. Вихідні дані, протокол покрового виконання, кінцевий результат – 3 варіанти.

5. Висновки і рекомендації.

5. Контрольні питання.

1. Порядок складання електронних схем в Multisim.

2. Порядок розробки прикладної програми.

3. Компіляція та відлагодження програми.

4. Запуск програми на виконання, анімація схеми.

5. Типи команд арифметичних операцій.

6. Регістр ознак, команди, що викликають зміну регістра ознак.

7. Аналіз файлу лістингу програми.

6. Рекомендована література

1. Введение в Multisim. Трехчасовой курс. National Instruments Россия, СНГ, Балтия - 42 с.

2. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990. - 224 с.

ЛАБОРАТОРНА РОБОТА №11

ПРОГРАМНЕ УПРАВЛІННЯ СВІТЛОДІОДНИМИ ІНДИКАТОРАМИ СТАТИЧНОГО ТИПУ

Мета роботи: засвоїти основні прийоми програмного управління світлодіодами та семисегментними індикаторами за допомогою мікроконтролера.

1 Завдання для виконання роботи

Скласти схему підключення світлодіодів та семисегментного індикатора до мікроконтролера 8051. На семисегментному індикаторі вивести цифри шістнадцяткової системи числення: від 0 до F та окремі алфавітні символи.

2 Опис систем відображення статичної інформації

Простими приладами відображення інформації в цифрових пристроях є світлодіоди та цифрові індикатори.

У напівпровідникових світлодіодах використовується властивість p-n

переходу випромінювати світло у видимій частині спектру при протіканні через нього прямого струму (Iпр=5-20мА, Uпр=2-3В). Варіанти включення індикаторів на показані на рис.2.1.

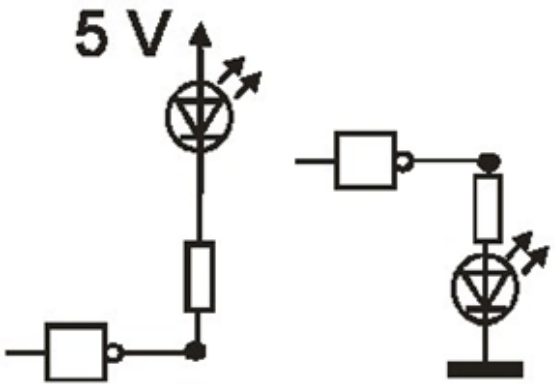


Рис.2.1. Включення одиничних індикаторів

Для відображення цифрової інформації найбільшого поширення набули семисегментні індикатори, в яких зображення цифри складають з семи лінійних світлодіодних сегментів розташованих у вигляді цифри 8.

На основі світлодіодів і семисегментних індикаторів будуються підсистеми відображення інформації. При побудові підсистем відображення інформації розрізняють два підходи - динамічну та статичну схему побудови підсистеми індикації.

Статична індикація полягає в постійному підсвічуванні індикаторів HL1-n від одного джерела інформації, див. рис.2.2, де представлені наступні позначення: **DA** – дешифратор адреси, необхідний для вибірки відповідного регістра;

R1-R4 - регістри, в яких тимчасово зберігається значення коди числа для відображення (відповідний регістр вибирається DA);

DC1-DC4 – семи сегментні дешифратори, що перетворюють двійковий код в семи сегментний код;

HL1-HL4 – семи сегментні індикатори;

ШД – шина даних, по ній здійснюється передача даних на індикацію.

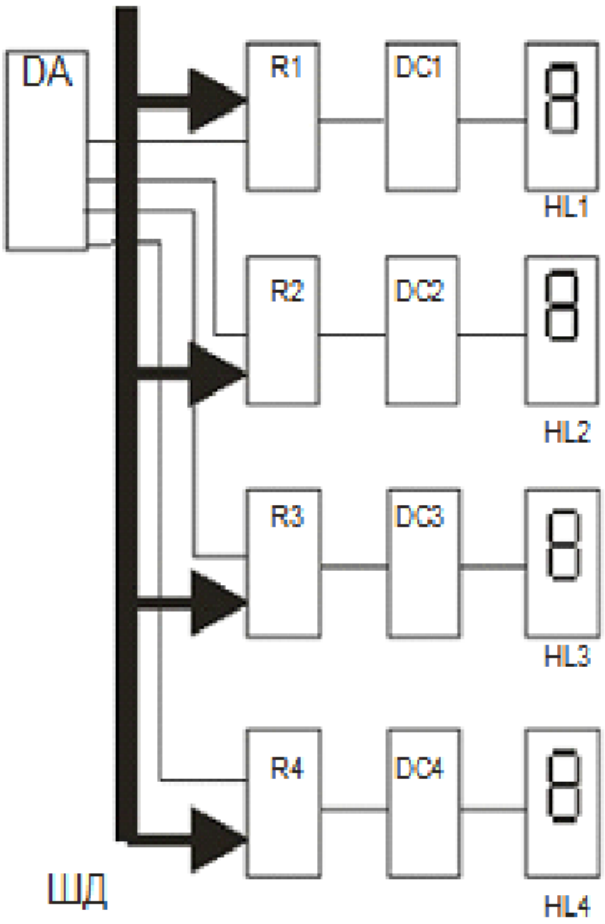


Рис.2.2 Структурна схема статичної індикації

Індикатори із загальним анодом (див. рис.2.3) активуються логічними нулями, а із загальним катодом – одиницями.

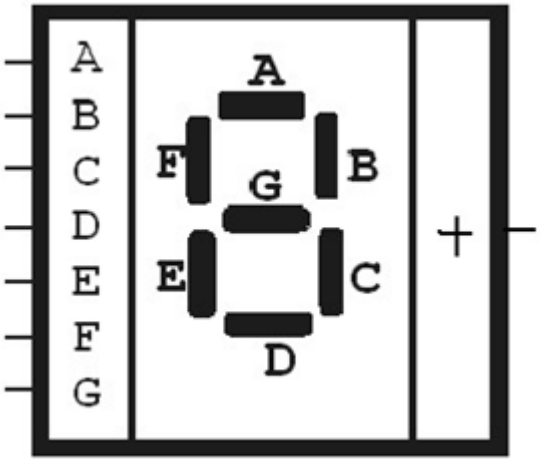


Рис.2.3. Схема семисегментного індикатора із загальним анодом

В бібліотеці Multisim (див. рис. 2.4) міститься багато таких індикаторів.

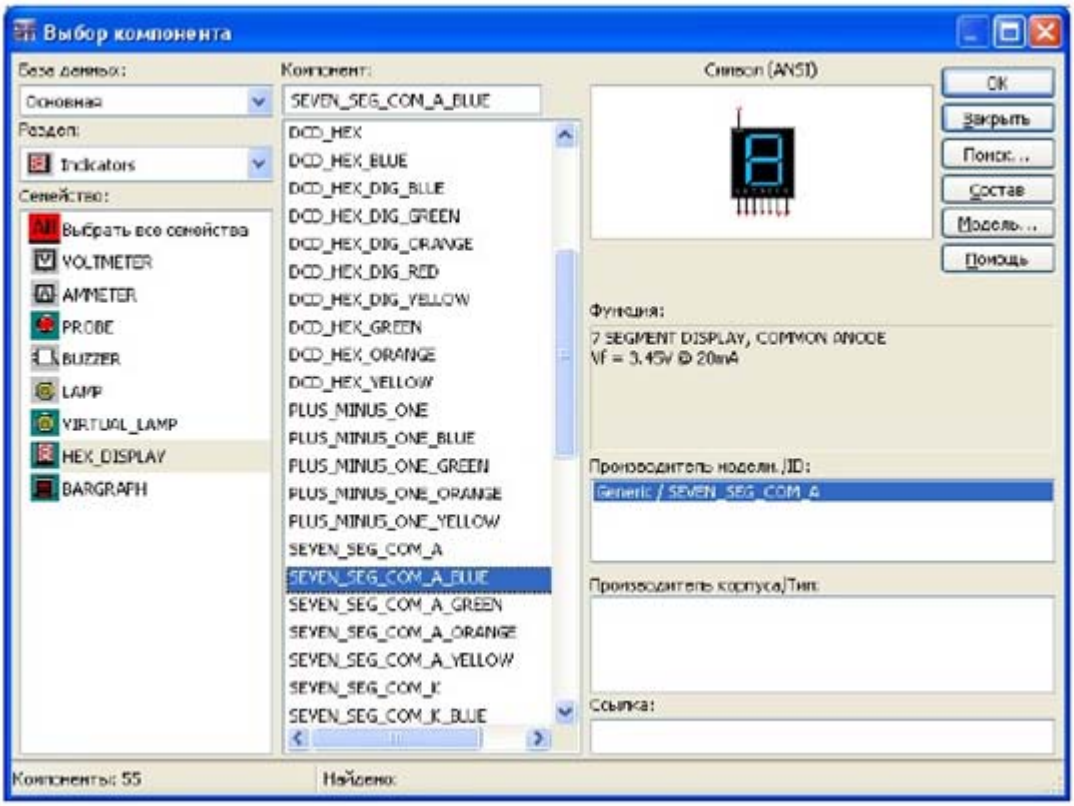


Рис.2.4. Вибір індикатора з бібліотеки

Малопотужні індикатори можна під’єднати і безпосередньо до портів мікроконтролера, використовуючи додаткове живлення через резистори.

Остаточна схема підключення індикатора в Multisim має вигляд, представлений на рис. 2.5.

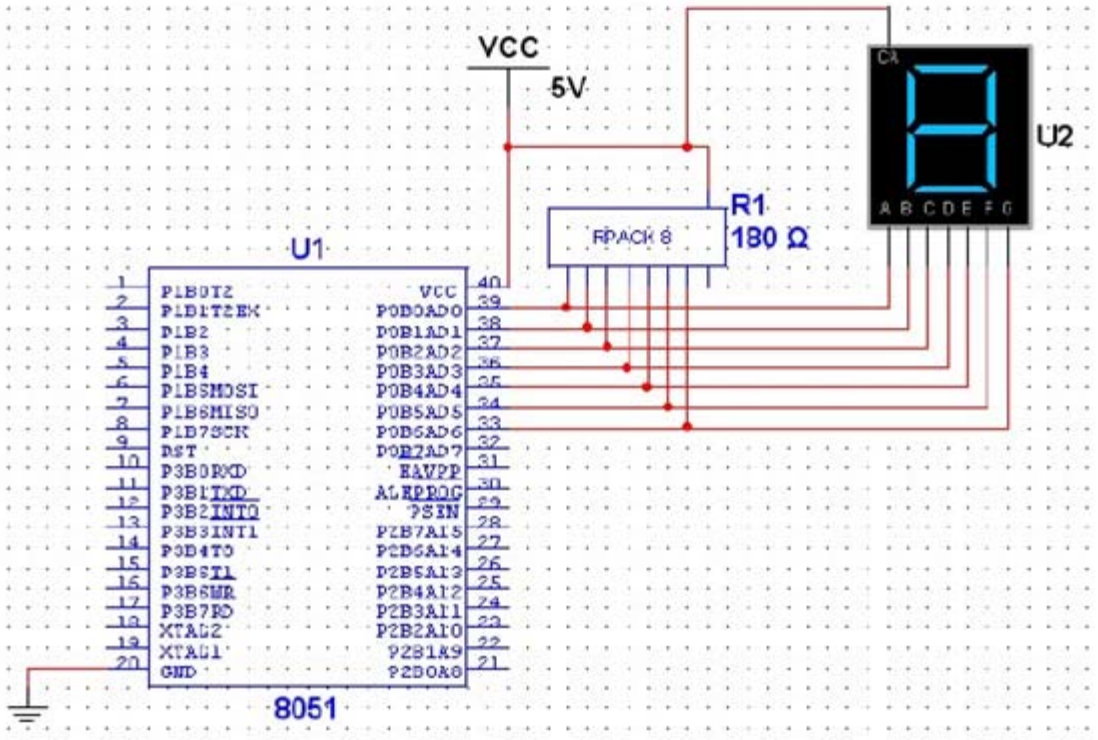


Рис.2.5. Варіант схеми статичної індикації

Приклад програми для статичної індикації.

```
ORG 0
Continue:
  mov A,#0
  mov DPTR,#0A004h
  movx @DPTR,A          ; відмінити гасіння знакомісць C_інд
  mov A,#04h            ; записати у Акумулятор число 04
  mov DPTR,#0A000h      ; встановити у DPTR адресу лівої
                        ; пари знакомісць C_інд
  movx @DPTR,A          ; засвітити число 04
  mov DPTR,#0B000h      ; установити у DPTR адресу правої
```

```
                    ; пари знакомісць C_інд
                     ;засвітити число 04
  movx @DPTR,A          ; виклик підпрограми затримки
  CALL ZAD
  mov A,#00001111b
  mov DPTR,#0A004h
  movx @DPTR,A          ; погасити всі знакомісця C_інд
  CALL ZAD
  jmp Continue
ZAD:                    ; перехід на початок програми
                    ; підпрограма затримки
   mov R1,#0FFh
C2: mov R2,#0FFh
C4: djnz R2, C4
   djnz R1, C2
    ret
END                     ; вихід з підпрограми
```

З частотою 1 Гц число «04» відображається на статичному індикаторі. Статична індикація реалізована на чотирьох статичних семисегментних індикаторах HG1 (розряди HG1.0, HG1.1, HG1.2, HG1.3). Звернення до них виробляється, як до елементів пам'яті з адресами A000h (ліва пара знакомісць), B000h (права пара знакомісць).

3. Порядок виконання роботи

1. В програмі Multisim розмістити на схемі компонент 8051.

2. Задати шлях на диску до робочої області та її назву. В якості мови програмування задати «Ассемблер». Задати назву проекту та ім’я вихідного файлу.

3. Скласти електронну схему, як показано на рис 11.5.

4. Проаналізувати підпрограму затримки, що наведена у прикладі.

5. Скласти програму на мові асемблера, яка по черзі виводить цифри шістнадцяткової системи числення (варіант задає викладач).

6. Підготувати середовище для налагодження програми. Скомпілювати програму, переконатися у відсутності помилок.

7. Виконати програму крок за кроком з початку і до кінця. Після кожної команди відмічати значення цифри, що змінилася.

8. Оформити звіт про роботу.

4. Вміст звіту про роботу

1. Назва, мета та завдання на виконання роботи.

2. Зображення мікроконтролера 8051 та схема підключення до нього семисегментного індикатора.

3. Лістінг програми мікроконтролера.

4. Вихідні дані, протокол покрового виконання, кінцевий результат.

5. Висновки і рекомендації.

5. Контрольні питання

1. Які найпростіші прилади відображення інформації використовуються в цифрових пристроях?

2. Поясніть схеми включення одиничних індикаторів.

3. Суть статичної індикації.

4. Принцип формування коду управління семисегментним індикатором.

5. В чому полягає різниця управління індикатором при наявності та відсутності апаратного дешифратора?

6. Рекомендована література

1. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990. - 224 с.

2. Кушков В.М. Мікропроцесорна техніка: Курс лекцій для студ. напряму 6.050202 «Автоматизація та комп'ютерно-інтегровані технології» ден. та заоч. форм навч. – К.: НУХТ. 2011. - 148 с.

ЛАБОРАТОРНА РОБОТА №12

ПРОГРАМНЕ УПРАВЛІННЯ АНАЛОГОВО_ЦИФРОВИМ ПЕРЕТВОРЮВАЧЕМ

Мета роботи: засвоїти основні прийоми підключення аналогово-цифрового перетворювача до мікроконтролера Intel 8051 та його програмування, використовуючи програмне забезпечення Multisim.

1. Завдання для виконання роботи

Скласти схему підключення аналогово-цифрового перетворювача до мікроконтролера 8051 в програмному середовищі Multisim. Написати програму для мікроконтролера, яка зчитує цифровий сигнал з аналогово-цифрового перетворювача та зображає його цифрове значення на двох шістнадцяткових індикаторах.

2. Опис роботи аналогово-цифрового перетворювача

Аналогово-цифровий перетворювач знаходиться в бібліотеці в розділі Mixed в сімействі ADC_DAC компонент ADC, див. рис. 2.1.

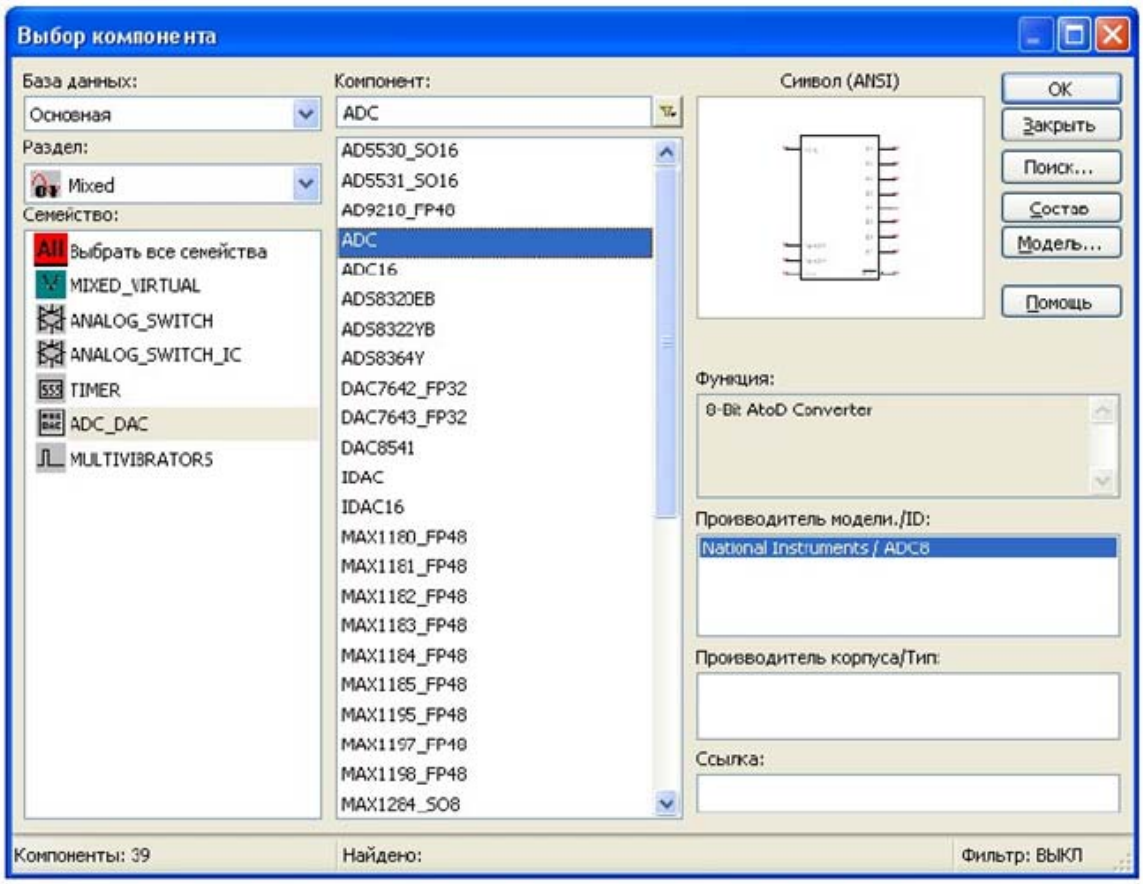


Рис. 2.1. Вибір аналогово-цифрового перетворювача із бібліотеки

Аналогово-цифровий перетворювач має вигляд, зображений на рис. 2.2.

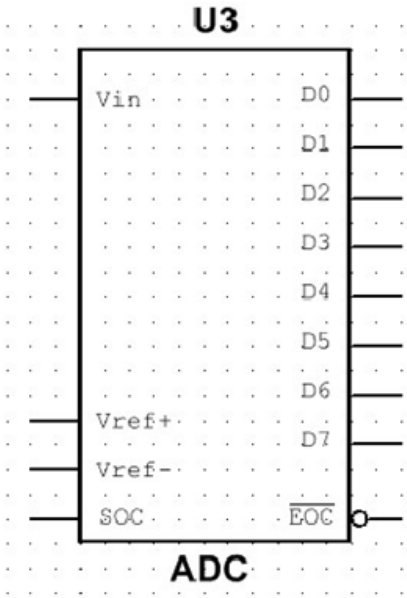


Рис. 2.2. Зовнішній вигляд аналогово-цифрового перетворювача

Виводи аналогово-цифрового перетворювача мають наступні функціональні призначення:

- **D0-D7** – цифровий сигнал, що подається на мікроконтролер;

- **Vin** – аналоговий сигнал, який потрібно перетворити в цифровий;

- **Vref+** – додатна клема живлення аналогово-цифрового перетворювача. На цей вивід підключається компонент VCC;

- **Vref-** – від'ємна клема живлення аналогово-цифрового перетворювача. На цей вивід підключається компонент GROUND;

- **SOC** – лінія, що відповідає за роботу аналогово-цифрового перетворювача.

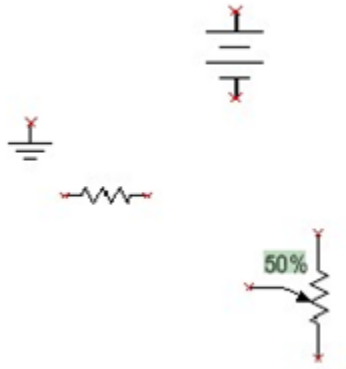
3. Порядок виконання роботи

1. Запустити програму Multisim.

2. Розмістити на схемі компонент 8051.

3. Задати шлях на диску до робочої області та її назву. В якості мови програмування задати «Ассемблер». Задати назву проекту та ім'я вихідного файлу.

4. Скласти електронну схему. На вивід Vin подавати сигнал в якому змінюється напруга. Для цього необхідно скласти електричне коло з наступних компонентів:



- батареї постійного струму DC_POWER;
- аналогової землі GROUND;
- резистора RESISTOR_RATED;
- потенціометра POTENTIOMETR_RATED.

На вивід SOC подавати 1 – початок зчитування аналогового сигналу. Після занесення значень цифрового сигналу з ліній D0-D7 в акумулятор мікроконтролера на вивід SOC подавати 0 – кінець зчитування аналогового

сигналу.

5. Записати код програми у відповідності до завдання та перевірити працездатність програми.

6. Оформити звіт про роботу.

4. Вміст звіту про роботу

1. Назва, мета та завдання на виконання роботи.

2. Зображення мікроконтролера 8051 та схема підключення до нього аналогово-цифрового перетворювача.

3. Лістинг програми мікроконтролера.

4. Зображення результатів виконання програми (для 5 різних значеннь аналогового сигналу відобразити 5 значень в шістнадцятирічному коді на цифрових індикаторах).

5. Висновки і рекомендації.

5. Контрольні питання

1. Функціональне призначення виводу Vin аналогово-цифрового перетворювача.

2. Функціональне призначення виводу SOC аналогово-цифрового перетворювача.

3. Аналіз файлу лістингу програми.

6. Рекомендована література

1. Фрунзе А.В. Микроконтроллеры? Это же просто! Т. 1. – М.: ООО “ИД СКИМЕН”, 2002. – 336 с.

ЛАБОРАТОРНА РОБОТА №13

ПРОГРАМНЕ УПРАВЛІННЯ РОБОТОЮ ДИСПЛЕЯ

Мета роботи: засвоїти основні прийоми підключення дисплея до мікроконтролера Intel 8051 та його програмування використовуючи програмне забезпечення Multisim.

1. Завдання для виконання робот.

Скласти схему підключення дисплею до мікроконтролера 8051 в програмному середовищі Multisim. Написати програму на мові асемблера, яка б виводила текстові рядки на екран дисплея (при натисканні на кнопку А на дисплеї висвічується напис – Start, при відпусканні кнопки А висвічується напис – Stop).

2 Опис роботи дисплея

Зовнішній вигляд схеми дисплея зображений на рис. 2.1.

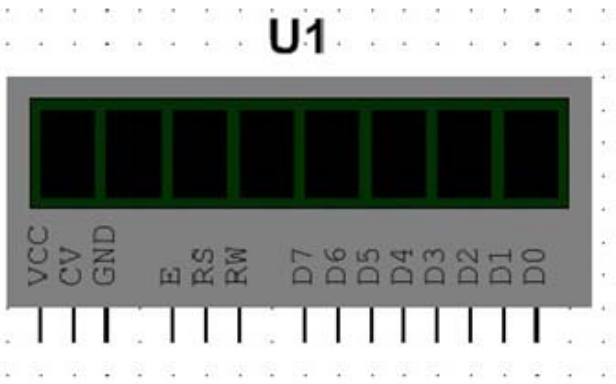


Рис. 2.1. Вигляд дисплея

Схема дисплея знаходиться в бібліотеці в розділі Advanced_Peripherals в сімействі LCDS компонент LCD_DISPLAY_08x1 (див. рис. 2.2).

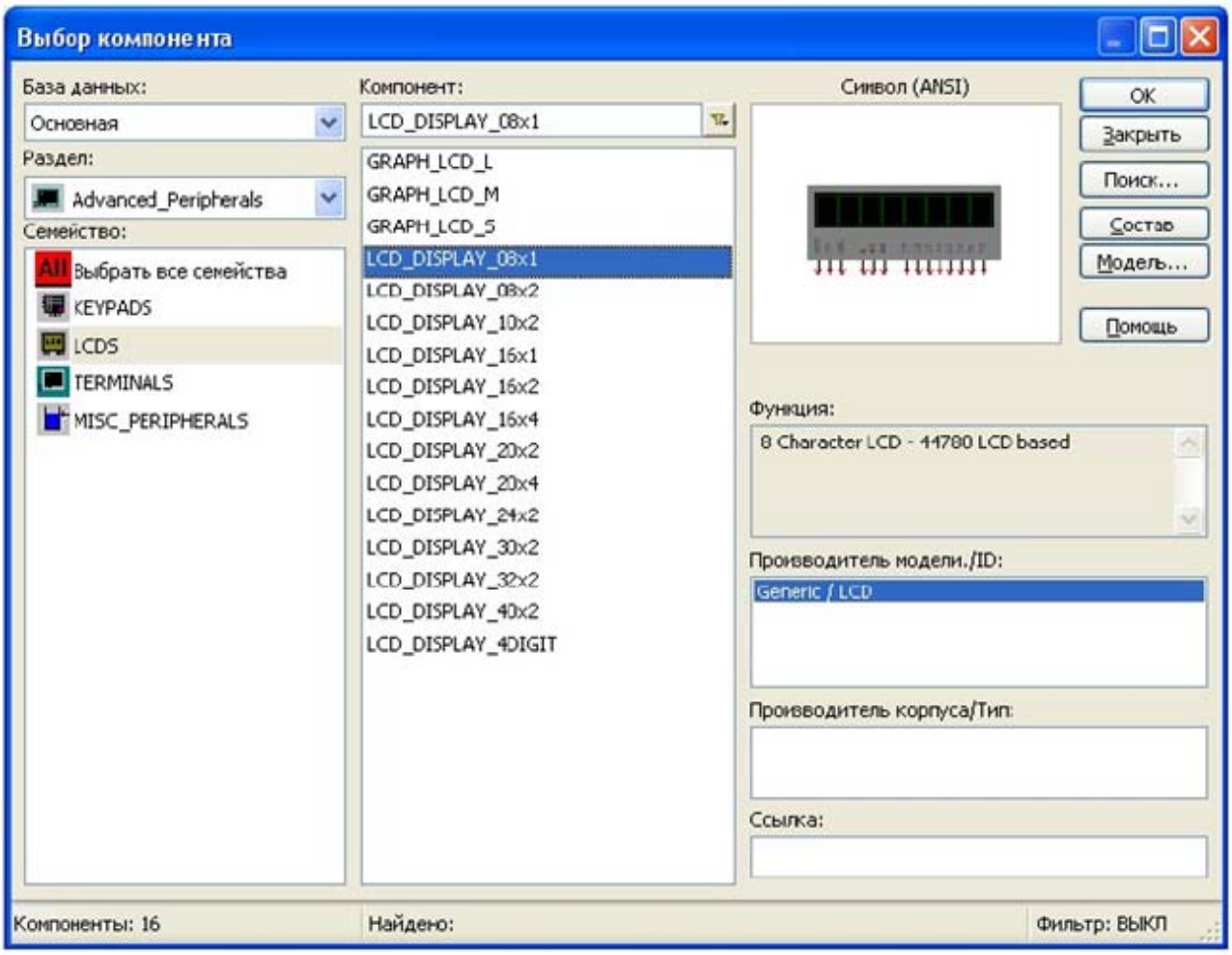


Рис. 2.2. Вибір дисплея із бібліотеки

На дисплеї максимально може загорітися 8 символів в один ряд на це вказує позначення 08x1 в назві компонента.

Виводи дисплея мають наступні функціональні призначення:

- **D0-D7** – лінії даних. По цим лініям на дисплей подаються дані або команди;

- **VCC** – живлення дисплею. На цей вивід підключається компонент VCC;

- **GND** – аналогова земля. На цей вивід підключається компонент GROUND;

- **E** (Enable) – лінія, що відповідає за початок та кінець передачі даних або команд;

- **RS** (Register Select) – при встановлені значення цієї лінії в 0 означає, що

по лініям D0-D7 передаються дані у вигляді команд дисплею, якщо на лінії встановити значення 1, то дані, що передаються по лініям D0-D7 будуть зображені на дисплеї;

- **RW** (Read/Write) – при встановлені значення цієї лінії в 0 означає запис даних на дисплей, при встановлені 1 – зчитування даних з дисплею.

Отже підключивши лінії D0-D7 до одного з портів мікроконтролера, а лінії E, RS, RW та кнопку A до виводів інших портів, як зображено на рис. 2.3, можна приступати до написання програми.

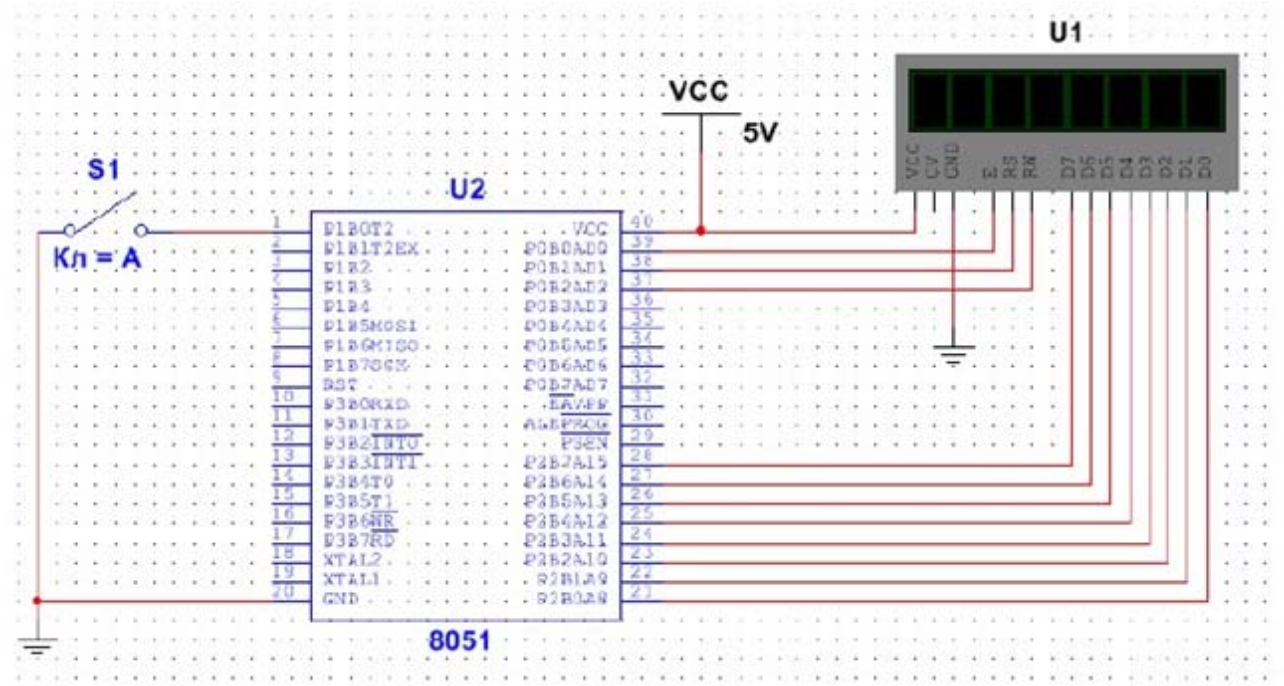


Рис. 2.3. Схема підключення дисплея до мікроконтролера Intel 8051

При написанні програми слід дотримуватися наступної послідовності:

1. Встановити значення виходу мікроконтролера, до якого підключений вивід E в 1 – початок передачі.

2. Встановити значення виходу мікроконтролера, до якого підключений вивід RS в 1 – зображати передані дані на дисплей.

3. Встановити значення виходу мікроконтролера, до якого підключений вивід RW в 0 – записувати передані дані на дисплей.

4. Передати в порт до якого підключені лінії D0-D7 код символу в шістнадцятковій системі числення.

5. Встановити значення виходу мікроконтролера, до якого підключений вивід E в 0 – кінець передачі.

На дисплей виводяться тільки латинські літери та арабські цифри. Шістнадцяткові коди літер та цифр зображені в таблиці ASCII на рис. 2.4.

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

Рис. 2.4. Таблиця кодів ASCII

3. Порядок виконання роботи

1. Запустити програму Multisim.

2. Розмістити на схемі компонент 8051.

3. Задати шлях на диску до робочої області та її назву. В якості мови програмування задати «Ассемблер». Задати назву проекту та ім’я вихідного файлу.

4. Скласти електронну схему, яку представлено на рис. 2.3.

5. Написати код програми у відповідності до завдання та перевірити працездатність програми.

6. Оформити звіт про роботу.

4. Вміст звіту про роботу

1. Назва, мета та завдання на виконання роботи.

2. Зображення дисплея та схема підключення його до мікроконтролера 8051.

3. Лістинг програми мікроконтролера.

4. Зображення результатів виконання програми.

5. Висновки і рекомендації.

5. Контрольні питання

1. Функціональне призначення виводу E дисплея.

2. Функціональне призначення виводу RS дисплея.

3. Функціональне призначення виводу RW дисплея.

4. Аналіз файлу лістингу програми.

6. Рекомендована література

1. Магда Ю.С. Микроконтроллеры серии 8051: практический подход. – М.: ДМК Пресс, 2008. – 228 с.

ЛАБОРАТОРНА РОБОТА № 14

КЕРУВАННЯ ФАЙЛАМИ

Мета роботи: засвоїти керування файлами в Multisim.

1. Короткі теоретичні відомості

MCU-модуль системи «Multisim» призначений для розробки й дослідження схем пристроїв на основі мікроконтролерів (MCU) [1]. При розробці схеми на основі мікроконтролера MCU-модуль використає наступний набір файлів:

- файл схеми з розширенням .ms10 для опису структури схеми;

- файл робочого простору MCU з розширенням .mcuws включає інформацію про всі проекти, що втримуються в робочому просторі MCU;

- файл проекту MCU з розширенням .mcupr містить інформацію про всі файли даного проекту;

- вихідний файл із розширенням .asm являє собою програму користувача мовою асемблера;

- файл включення з розширенням .inc є програмою мовою асемблер, що використається програмою користувача;

- вихідний файл із розширенням .c являє собою програму користувача мовою С.

Кожен файл робочого простору MCU розташований у папці робочого простору, при цьому назви папки й файлу збігаються. Всі інші файли, що ставляться до даної схеми, розташовуються усередині папки робочого простору MCU.

Кожен проект має свою папку, розташовану в межах папки робочого простору. Кожен файл проекту розташований у папці проекту й назви папки й файлу проекту збігаються.

Файли на мовах асемблера й С розташовуються в межах папки проекту.

На рис. 1.1 наведений фрагмент схеми (Circuit1), що містить мікроконтролер (U1). При цьому на панелі Design Toolbox показана структура розташування відповідних файлів.

У цьому випадку робочий простір MCU містить два проекти - project1 й project2, при цьому проект project1 містить ряд файлів, а проект project2 є порожнім.

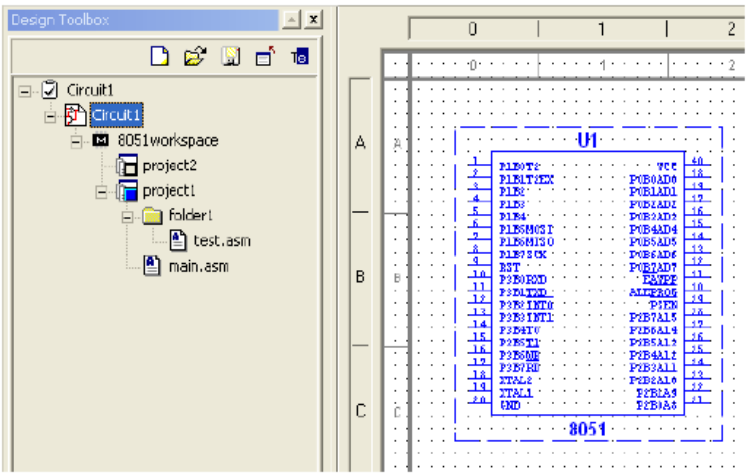


Рис. 1.1. Фрагмент схеми, що містить мікроконтролер

Структура розташування файлів для схеми Circuit1 наведена на рис.1.2.

Інформація про розташування робочого простору MCU зберігається у файлі опису структури схеми (*.ms10), що дозволяє працювати з файлами робочого простору при кожнім відкриванні схеми.

Для організації робочого простору MCU необхідно почати наступні кроки:

- вибрати пункт меню Place/COMponent для виклику діалогового вікна вибору компонентів (Select a COMponent), задати групу модуля MCU (MCU Module Group) і потрібне сімейство (805x або PIC), вибрати мікроконтролер і помістити його на схему. При цьому з'явиться діалогове вікно MCU Wizard;

- у вікні MCU Wizard варто задати найменування робочого простору MCU, указати його розташування й нажати клавішу Next;

- указати інформацію, що ставиться до проекту, задаючи тип проект (Project type), мова програмування (Programming language), вид транслятора (Accembler/COMpiler tool) і найменування проекту (Project name). При цьому при завданні типу проекту може бути обраний або стандартний варіант (Standart), коли використається вихідний текст програми користувача, або варіант завантаження зовнішнього шістнадцяткового файлу програми (Load External Hex File). Як мова програмування може бути прийнята мова асемблера або мова С. Після задання інформації про проект необхідно нажати клавішу Next;

- вибрати джерело програми користувача. При цьому можна створити порожній проект (Create empty project), якщо на даному етапі не буде створюватися програма користувача або додати файл програми (Add source file). В останньому випадку необхідно задати ім'я файлу програми, складеної мовою асемблера або С. Після вибору джерела програми необхідно нажати клавішу Finish.

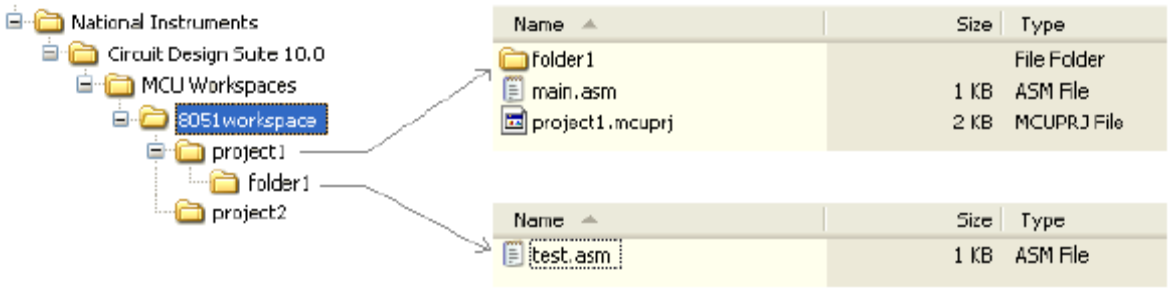


Рис. 1.2. Структура розташування файлів

Після створення робочого простору MCU є можливість для цього простору видаляти або додавати файли й проекти.

1.1. Менеджер коду MCU-модуля

Діалогове вікно менеджера коду модуля MCU (MCU Code Manager) дозволяє організувати роботу з файлами робочого простору MCU, а також задати установки для кожного проекту MCU.

Для виклику менеджера коду необхідно вибрати пункт меню MCU/ MCU8051U1/ MCU Code Manager (див. рис.1.3).

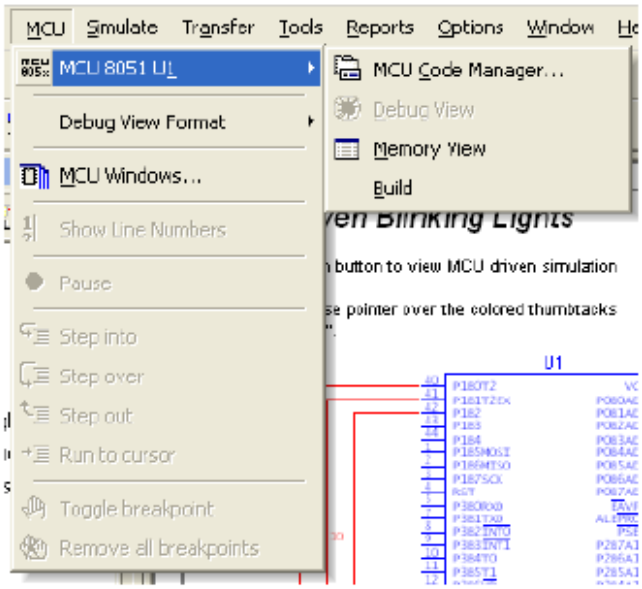


Рис. 1.3. Менеджера коду

Зображення діалогового вікна наведене на рис. 1.4.

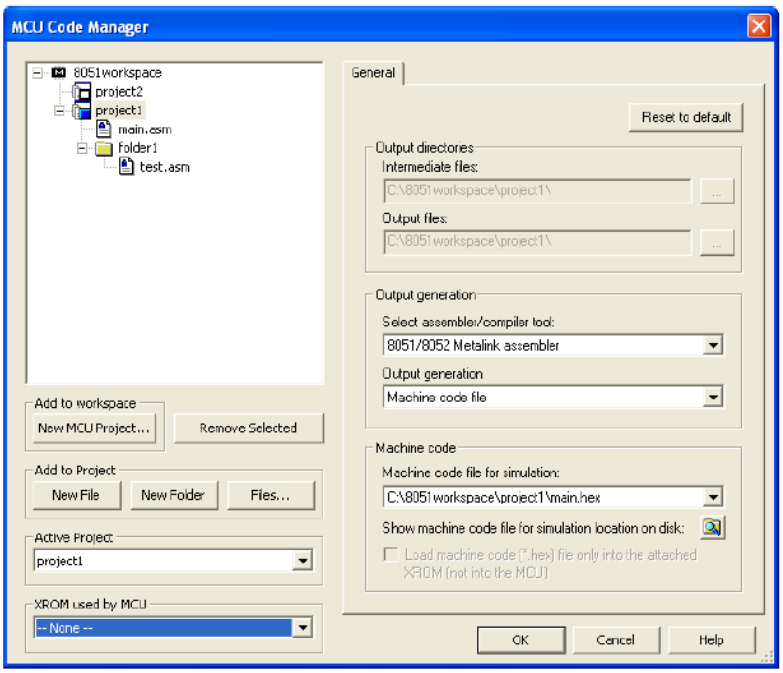


Рис. 1.4. Діалогове вікно проекту

Використовуючи діалогове вікно менеджера коду, можливо додавати до робочого простору MCU або видаляти з нього файли й проекти. При цьому для виклику додаткових вікон використаються клавіші New MCU Project (новий MCU-проект), New File (новий файл), New Folder (нова папка) і Files (файли). Додатково для роботи з файлами й проектами робочого простору MCU може бути використана панель Design Toolbox. При цьому за допомогою лівої кнопки миші вибирається потрібний файл, проект або робочий простір, а потім правою кнопкою викликається додаткове меню.

Діалогове вікно менеджера коду дозволяє також вибрати новий **активний проект** із ряду існуючих у робочому просторі MCU.

Ще одне призначення діалогового вікна пов'язане із **завданням установок** проектам MCU-модуля.

Кожен проект має свої установки, які можуть відрізнятися від установок інших проектів навіть у межах одного робочого простору. Установки залежать від типу використовуваного транслятора. При цьому «Multisim» дозволяє задавати для трансляції програм користувача або асемблери (8051 Metalink Assembler або Microchip PIC Assembler), або компілятори (Hi-Tech 8051 C Lite

COMpiler або Hi-Tech Picc Lite COMpiler). Установки для проекту включають завдання інформації про тип використовуваного транслятора, настроюваннях транслятора і про місце розташування формованих файлів.

Після завдання установок проекту можна переходити до **побудови робочого простору MCU.** У процесі такої побудови активується обраний транслятор і формуються відповідні вихідні файли (файл у машинних кодах, файл лістингу та інші).

Для побудови робочого простору варто вибрати пункт меню MCU/ MCU 8051U1/Build. Результати побудови відображаються на панелі Spreadsheet View у нижній частині екрана. Приклад відображення результатів побудови наведений на рис. 1.5.

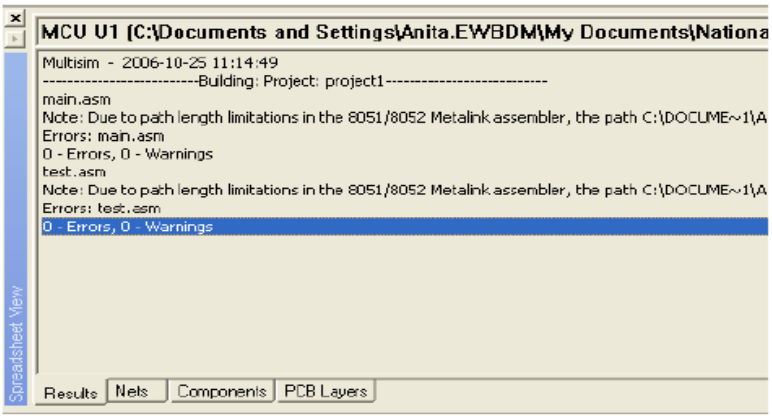


Рис. 1.5. Приклад відображення результатів побудови

У випадку успішної побудови формується вихідний файл (*.hex або *.lib). В іншому випадку виводиться повідомлення про **помилки** із вказівкою їхнього типу й місця розташування у вихідній програмі.

Використовуючи подвійний клік мишею на лінії з номером помилки, можна відкрити файл вихідної програми, при цьому курсор буде поміщений на відповідну лінію. Після корегування програми варто повторити побудову. У випадку успішної побудови робочого простору MCU можна переходити до моделювання роботи схеми, вибираючи пункт меню Simulate/Run.

1.2. Редактор вихідного тексту програми MCU-модуля

Для того щоб відкрити для перегляду або редагування файл вихідного тексту програми варто зробити подвійний клік мишею на імені файлу, відображеному на панелі Design Toolbox. Кольори для відображення тексту програми використовуються редактором у такий спосіб:

- блакитний текст позначає ключові слова програми;

- зелені кольори використається в коментарях;

- чорні кольори застосовується у всіх інших випадках (задання даних, відображення помилок і т.д.). При наявності граматичних помилок ключові слова відображаються не блакитним, а чорним кольором.

Приклад зображення вихідного тексту програми показаний на рис. 1.6.

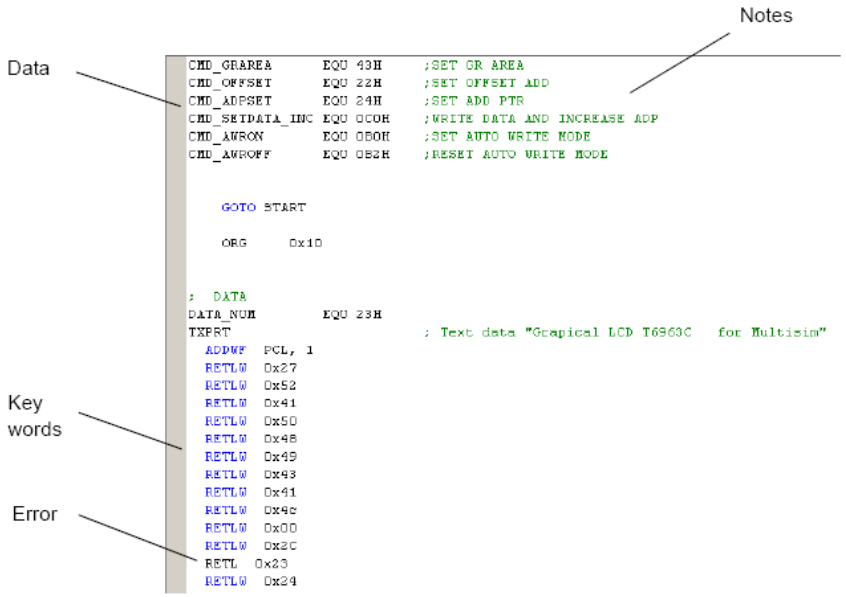


Рис. 1.6. Вихідний текст програми

Для редагування тексту можна використати стандартний пункт меню Edit з операціями копіювання (Copy), видалення (Delete), вставки (Paste), скасування останньої дії (Undo) і повтору останньої дії (Redo).

Для збереження відредагованого файлу необхідно вибрати пункт меню File/Save, а для печатки - File/Print.

Після редагування файлу необхідно виконати операцію побудови робочого простору, вибираючи пункт меню MCU/ MCU8051U1/Build.

1.3. Засоби налагодження MCU-модуля

Після трансляції вихідної програми в шістнадцятковий файл користувачеві доступна наступна інформація:

- вихідний текст програми, що транслювався MCU-модулем;

- файл лістингу програми. Якщо вихідний текст був представлений мовою З, то файл лістингу є компіляцією програми з мови С на мову асемблера. Якщо ж вихідний текст був складений мовою асемблера, то файл лістингу збігається з файлом вихідного тексту, відмінності можливі тільки при використанні в програмі макросів;

- дезасембльований текст програми, створений MCU-модулем на основі шістнадцяткового файлу. Даний файл подібний до файлу лістингу. Відмінності полягають у використанні біля дезасембльованих абсолютних значеннях замість символічних імен (виключення становлять тільки команди переходу jump/go to, що використають мітки).

Після побудови MCU-проекту (команда Build) можна для налагодження програми використати додаткові вікна.

Для того щоб відкрити вікна налагодження, необхідно вибрати пункт меню MCU/ MCU8051U1/Debug View.

У цьому випадку користувачеві доступне вікно з дезасембльованим файлом (Project disassembly) і вікно з файлом лістингу (Source file debug listing). Для перемикання вікон використається щиглик лівої кнопки миші на назві вікна. Вікно з дезасембльованим файлом також може бути викликано, використовуючи клік правою кнопкою миші на файлі *.asm панелі Design Toolbox.

Для вибору інформації, відображуваної в відладочному вікні, необхідно використати пункт меню MCU/Debug View Format. При цьому користувачеві стають доступні наступні кнопки:

- кнопка вихідного тексту програми (Source Code), що дозволяє виконувати налагодження програми із застосуванням її вихідного тексту;

- кнопка дезасемблювання (Disassembly) вибирає для відображення або файл лістингу, або дезасемблюваний файл;

- кнопка подання програми альтернативною мовою як коментар (Show secondary Language as COMments) дозволяє відображати на іншій мові текст програми як коментар. При цьому, якщо вихідним є файл лістингу або дезасемблюваний файл, то як коментар приводиться вихідний текст програми й навпаки;

- кнопка відображення номерів рядків (Show line Numbers) дає можливість відображати номера рядків програми;

- кнопки відображення адрес пам'яті (Show Memory Addresses in Debug View й Show Memory Addresses in Assembly Code) використаються для додаткового відображення адрес комірок пам'яті;

- кнопка відображення шістнадцяткових кодів (Show Hex Opcodes in Assembly Code) дозволяє включити в зображення шістнадцяткові коди команд;

- кнопка відображення міток (Show Jump/Goto Labels) дає можливість зображувати мітки, використовувані командами Jump й Goto;

- кнопка відображення заголовка (Show Headings Above Code) дозволяє використати заголовок для відображення файлу лістингу або дезасемблюваного файлу.

Після побудови проекту користувач може задати **точки зупинки програми (Breakpoints).** Для завдання крапок можуть використатися вихідний файл програми, файл лістингу або дезасемблюваний файл. Точка зупинки відзначається подвійним кліком лівої кнопки миші, при цьому курсор повинен

бути попередньо встановлений ліворуч від обраної команди. Забрати крапку зупинки можна повторним подвійним кліком лівої кнопки миші.

Точки зупинки дають можливість призупинити виконання програми після запуску процесу моделювання (Simulate/Run). Подібна зупинка дозволяє проаналізувати поточну інформацію в регістрах й у комірках пам'яті мікроконтролера.

Крім налагодження програми з використанням точок зупинки може бути використаний також і **покроковий режим налагодження.** Для роботи в покроковому режимі використаються кнопки Step into та Step over.

При цьому кожне натискання кнопки Step into приводить до виконання однієї чергової команди програми. Кнопка Step over функціонує аналогічно. Відмінність полягає тільки у виконанні команди виклику підпрограми CALL. В одному випадку здійснюється перехід до першої команди підпрограми, а в іншому - підпрограма виконується повністю.

В обох випадках поточна виконувана команда відзначається жовтою стрілкою в лівій частині вікна.

До налагоджувальних вікон MCU-модуля також ставляться **вікна, що відображають уміст регістрів і комірок пам'яті** мікроконтролера. Дані вікна відкриваються вибором пункту меню MCU/ MCU8051U1/Memory View. Користувачеві доступне вікно регістрів спеціальних функцій (SFR), вікно внутрішнього ОЗУ мікроконтролера (IRAM), вікно внутрішнього ПЗУ мікроконтролера (IROM), а також вікно зовнішньої оперативної пам'яті (XRAM). Зображення вікон наведене на рис. 1.7.

Вміст регістрів і комірок пам'яті може бути переглянутим перед виконанням програми, після її завершення, при досягненні чергової точки зупинки, а також після виконання чергової команди в покроковому режимі. При необхідності вміст регістра може бути змінений, для чого використається подвійний клік лівої кнопки миші, що дозволяє вибрати потрібний регістр або осередок, з наступним записом нового значення в тім же форматі.

У тому випадку, якщо схема досліджуваного пристрою, крім мікроконтролера, містить додаткову зовнішню пам'ять (постійну або оперативну), є можливість перегляду й зміни вмісту даної пам'яті.

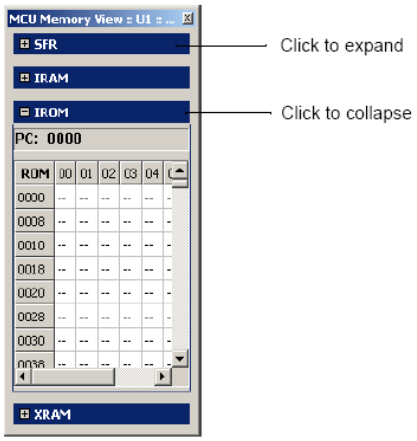


Рис. 1.7. Вікно пам'яті мікроконтролера

ЛАБОРАТОРНА РОБОТА № 15

ПРИСТРОЇ ВВОДУ/ВИВОДУ МІКРОПРОЦЕСОРНИХ СИСТЕМ

Мета роботи: ознайомлення із пристроями уведення/виводу мікропроцесорних систем, органами керування й режимами роботи пристроїв.

1 Короткі теоретичні відомості

1.1 Аналого-цифровий перетворювач

Для вибору АЦП варто звернутися до сімейства ADC-DAC групи Mixed бази даних «Multisim».

Як приклад розглянемо 8-розрядний АЦП (див. рис. 1.1) .

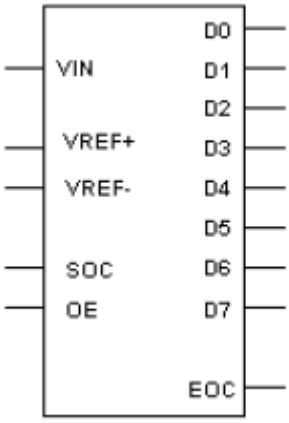


Рис. 1.1. 8-розрядний АЦП

На вхід VIN подається вимірювана напруга. Діапазон зміни VIN лежить у межах від VREF- до VREF+, при цьому до входів VREF- і VREF+ підключається джерело постійної напруги.

Для запуску АЦП необхідно подати сигнал високого рівня (логічну одиницю) на вхід SOC. Час перетворення становить 1мкс. У плині роботи АЦП на виході готовності EOC установлюється низький рівень напруги (логічний нуль). Поява високого рівня на виході EOC сигналізує про завершення аналого-цифрового перетворення. Результат перетворення можна зчитати з виходів даних АЦП D7...D0, подаючи логічну одиницю на вхід OE.

При подачі на OE логічного нуля на виходах даних установлюється третій (високоомний) стан.

Залежність коду, формованого АЦП, від вхідної напруги є лінійною. При цьому нульовій вхідній напрузі відповідає код нуля (00000000 - у двійковому поданні), а максимальному вхідному сигналу (U=VREF+ - VREF-) відповідає код 255 (11111111 - у двійковому поданні).

Для підключення АЦП до мікроконтролера необхідно виходи даних D7...D0 приєднати до порту вводу, а одну з ліній порту виводу використати для запуску АЦП, послідовно видаючи на цю лінію логічний нуль, одиницю й знову нуль. На входи VREF- і VREF+ необхідно подати постійну напругу, що

визначає діапазон вхідного сигналу VIN. Вихід EOC АЦП необхідно з'єднати із входом OE.

1.2 Цифроаналоговий перетворювач

Для одержання інформації про цифроаналогові перетворювачі необхідно звернутися до сімейства ADC_DAC групи Mixed бази даних «Multisim». Як приклад розглянемо 8-розрядний ЦАП (див. рис. 1.2).

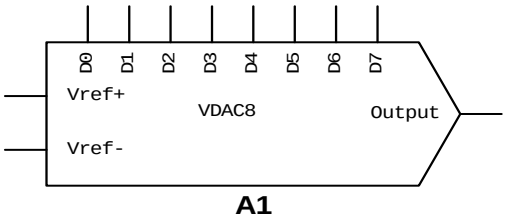


Рис. 1.2. 8-розрядний ЦАП

На входи D7...D0 ЦАП подається утворений двійковий код. До входів VREF+ й VREF- підключається джерело постійної напруги, що задає діапазон зміни вихідної напруги OUTPUT.

Залежність вихідної напруги ЦАП від вихідного коду є лінійною, при цьому мінімальному коду 00000000 відповідає нульова вихідна напруга, а максимальному коду 11111111 - максимальна напруга U=(VREF+) - (VREF-).

Для підключення ЦАП до мікроконтролера необхідно лінії D7...D0 приєднати до порту виводу, а на входи VREF+ й VREF- подати постійну напругу, що визначає діапазон вихідного сигналу ЦАП.

1.3 Аналоговий комутатор

Для вибору аналогового комутатора варто звернутися до сімейства ANALOG_SWITCH групи MIXED бази даних «Multisim».

Як приклад розглянемо 16-канальний комутатор ADG406BN (див. рис. 1.3).

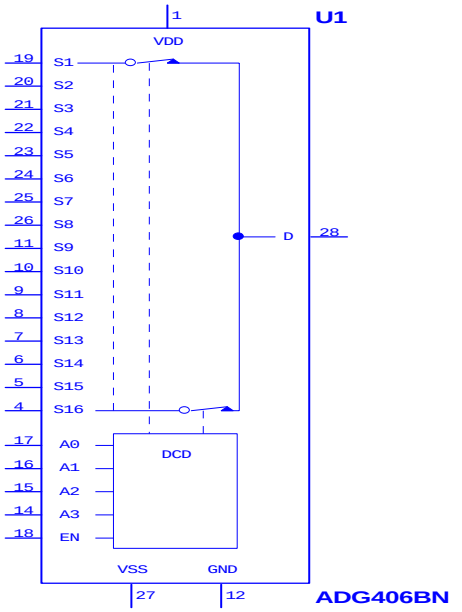


Рис. 1.3. 16-канальний комутатор ADG406BN

До входів V_{DD+} й V_{ss-} підключається джерело постійної напруги, що визначає діапазон вхідних сигналів S1…S16... Адреса A3…A0 визначає, який із вхідних сигналів S підключається на вихід D комутатора. При цьому код 0000 відповідає вхідному сигналу S_1, а код 1111 – сигналу S16. Загальний дозвіл комутації дається сигналом дозволу EN=1. При подачі на EN логічного нуля вихід D відключається від всіх входів S.

Для підключення комутатора до мікроконтролера треба входи адреси A3...A0 і вхід дозволу EN приєднати до порту виводу.

1.4 Клавіатура

Вибрати клавіатуру можна, звернувшись до сімейства Keypads групи Advanced Peripherals бази даних “Multisim».

Як приклад розглянемо клавіатуру 4x4 на 16 клавіш (рис.1.4).

Клавіатура містить 16 ключів, розташованих у вузлах матриці 4x4, при цьому натискання клавіші приводить до замикання відповідного ключа.

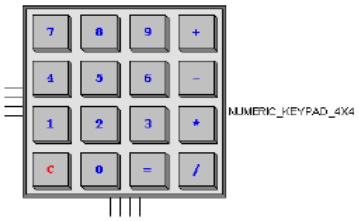


Рис. 1.4. Клавіатура 4x4 на 16 клавіш

Є 4 вхідні лінії вибору рядка клавіатури й 4 вихідні лінії, що відповідають стовпцям клавіатури.

Клавіатура підключається до мікроконтролера в такий спосіб. Входи вибору рядки приєднуються до порту виводу, а виходи стовпців - до порту вводу.

Для визначення натиснутої клавіші використається так званий режим сканування, коли мікроконтролер по черзі видає одиницю на лінії вибору рядка (коди 0001, 0010, 0100, 1000). Кожен вивід коду супроводжується вводом сигналів від стовпців. При цьому наявність одиниці в одному з розрядів вхідного коду свідчить про натиснуту клавішу. Ідентифікація клавіші здійснюється з урахуванням аналізу вихідного й вхідного коду.

У процесі моделювання роботи схеми для імітації натискання клавіші необхідно використати щиглика лівої кнопки миші на даній клавіші.

2 Завдання для виконання

Для виконання роботи необхідно:

1. Запустити «Multisim» і створити проект MCU.

2. Зібрати схему заданого послідовного пристрою уведення/виводу для мікропроцесорних систем. Підключити до схеми клавіатуру й цифро-аналоговий перетворювач.

3. Виконати дослідження заданої викладачем серійної послідовної схеми, а також пристрою на її основі. Для тестування використати генератор.

Проаналізувати тимчасові діаграми на виході цифро-аналогового перетворювача. Роздрукувати схему й діаграми.

3.Контрольні питання

1. Які відмінні риси ЦАП?
2. Яке призначення аналогового комутатора?
3. Які особливості портів мікропроцесора?
4. Яким образом підключається клавіатура до мікропроцесора?
5. Особливості програмного коду на асемблері для керування послідовними портами мікропроцесора.

4. Бібліографічний список

1. Комп'ютери: Довідкове керівництво /Під ред. М. Хелмса. – М.:Мир, 1986. – 416 с.