Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет будівництва, транспорту та енергетики
Кафедра автоматизації виробничих процесів

ПРОГРАМНО-ТЕХНІЧНІ КОМПЛЕКСИ ТА ПРОМИСЛОВІ КОНТРОЛЕРИ

Методичні вказівки для виконання практичних робіт
для другого рівня вищої освіти
зі спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»

Затверджено на засіданні кафедри
"Автоматизація виробничих процесів"
Протокол №1 від 29.08.2024 р.

Кропивницький 2024

ПРОГРАМНО-ТЕХНІЧНІ КОМПЛЕКСИ ТА ПРОМИСЛОВІ КОНТРОЛЕРИ. Методичні вказівки до виконання практичних робіт для другого рівня вищої освіти зі спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» /Укл.: Трушаков Д.В., Федотова М.О. – Кропивницький: ЦНТУ, 2024 - 91 с.

Укладачі: Трушаков Д.В. – канд. техн. наук, доцент;
Федотова М.О. – канд. техн. наук, асистент.

Рецензент: Сербул О.М. – канд. техн. наук, доцент.

Практична робота №1
МІКРОКОНТРОЛЕР INTEL 8051

МЕТА РОБОТИ

Метою роботи є вивчення архітектури мікроконтролерів популярного сімейства 8051 Intel, а також інтегрованого середовища ProView фірми Franklin Software Inc., призначеного для розробки програмного забезпечення для цього сімейства. Робота розрахована на 4 години домашньої підготовки і 4 години занять у лабораторії.

При підготовці до роботи вивчається структура одного з клонів сімейства – мікроконтролера INTEL 8051, його функціональні вузли й особливості їхньої роботи. Перед початком лабораторної роботи проводиться колоквіум. Студенти, що успішно відповіли на поставлені питання, допускаються до лабораторної частини роботи.

У лабораторії на простому прикладі по методу «Швидкий старт» вивчаються етапи технології розробки і налагодження програм, основні прийоми роботи з ProView.Після виконання роботи оформляється звіт із зазначеним нижче змістом.

ЗАВДАННЯ ДЛЯ ДОМАШНЬОЇ ПІДГОТОВКИ

1.1. Вивчить особливості однокристальних мікроконтролерів

Сімейства однокристальних мікроконтролерів. Узагальнені структури. Функціональні можливості й області застосування [1-4].

1.2. Вивчить архітектуру мікроконтролерів сімейства Intel 8051

Однокристальні мікроконтролери серії INTEL 8051. Арифметико-логічний пристрій, пам'ять даних і пам'ять команд, регістри загального призначення і регістри спеціальних функцій, пристрій керування і синхронізації, таймер-лічильник, паралельні порти вводу\виводу, послідовний порт. Система преривань. Розширення пам'яті програм і пам'яті даних. Розширення портів [2, 3].

1.3. Вивчить систему команд мікроконтролерів

Загальні відомості про систему команд. Типи операндів. Способи адресації. Прапори результату. Група команд передачі даних. Арифметичні команди. Команди логічних операцій. Команди операцій з бітами. Група команд передачі керування [2, 3].

1.4. Вивчить розділ «Швидкий старт»

Детально дослідите вихідні тексти програми мовою асемблера.

1.5. Контрольні питання

1. Перелічить характерні риси архітектури однокристальних мікроконтролерів.
2. Вкажіть програмно-доступні вузли INTEL 8051 і призначення регістрів спеціальних функцій.
3. Дайте визначення поняттю «булев» процесор.
4. Назвіть і охарактеризуйте чотири типи інформаційних об'єктів, з якими може оперувати арифметико-логічний пристрій мікроконтролера.

5. Яка ємність адресуємої пам'яті програм і даних у INTEL 8051?
6. Який регістр виконує функції базового регістра при непрямих переходах у програмі?
7. Які операції можуть бути виконані тільки з використанням акумулятора?
8. Які операції можуть бути виконані без участі акумулятора?
9. Який формат має слово стану програми INTEL 8051? Вкажіть призначення прапорів.
10. Які можливості надає наявність декількох банків регістрів загального призначення?
11. Як переключити банк регістрів загального призначення?
12. Який регістр використовується для адресації зовнішньої пам'яті даних?
13. Як сполучити адресні простори пам'яті програм і даних?
14. Охарактеризуйте спосіб адресації елементів стека в мікроконтролері.
15. Яка тривалість виконання команд у мікроконтролері?
16. Охарактеризуйте режими роботи таймера/лічильника в INTEL 8051.
17. Як за допомогою таймера можна виміряти тривалість імпульсу?
18. Як виводиться адреса зовнішньої пам'яті?
19. Яка навантажувальна здатність портів?
20. Перелічить альтернативні функції портів.
21. Охарактеризуйте режими роботи послідовного порту в INTEL 8051.
22. Як змінити швидкість передачі даних через послідовний порт?
23. Для чого використовується дев'ятий біт при передачі даних через послідовний порт?
24. Намалюйте схему преривань у INTEL 8051. Перелічить й охарактеризуйте типи преривань.
25. Для чого потрібний регістр масок переривання? Як змінити пріоритети преривань?
26. Як переводиться мікроконтролер у режим зниженого енергоспоживання?
27. Охарактеризуйте режим завантаження і верифікації програм.
28. Перелічить етапи технології розробки програм для мікроконтролерів.
29. Вкажіть призначення основних модулів інтегрованого середовища ProView.
30. Що таке об'єктний код, які функції виконує компоновувач?
31. Вкажіть основні тенденції розвитку мікроконтролерів.

АРХІТЕКТУРА МІКРОКОНТРОЛЕРА INTEL 8051

У мікропроцесорній техніці виділився самостійний клас інтегральних схем – мікроконтролери, що призначені для вбудовування в прилади різного призначення. Від класу однокристальних мікропроцесорів їх відрізняє наявність вбудованої пам'яті, розвинені засоби взаємодії з зовнішніми пристроями.

Мікроконтролер виконаний на основі високорівневої n-МОП технології. Через чотири програмувальні паралені порти вводу/виводу та один послідовний порт мікроконтролер взаємодіє з зовнішніми пристроями. Основу структурної схеми (мал.1) утворює внутрішня двонаправлена 8-бітна шина, що зв'язує між собою основні вузли і пристрої мікроконтролера: резидентну пам'ять програм (RPM), резидентну пам'ять даних (RDM), арифметико-логічний пристрій (ALU), блок регістрів спеціальних функцій, пристрій керування (CU) і порти вводу/виводу (P0-P3).

Арифметико-логічний пристрій

8-бітний арифметико-логічний пристрій (ALU) може виконувати арифметичні операції додавання, віднімання, множення і ділення; логічні операції І, АБО, виключаєче АБО, а також операції циклічного зрушення, скидання, інвертування і т.п. До входів підключені програмно-недоступні регістри Т1 і Т2, призначені для тимчасового збереження операндів, схема десяткової корекції (DCU) і схема формування ознак результату операції (PSW).

Найпростіша операція додавання використовується в ALU для інкрементування вмісту регістрів, просування регістра-покажчика даних (RAR) і автоматичного обчислення наступної адреси резидентної пам'яті програм. Найпростіша операція вирахування використовується в ALU для декрементування регістрів і порівняння змінних.

Найпростіші операції автоматично утворять «тандеми» для виконання таких операцій, як, наприклад, інкрементування 16-бітних реєстрових пар. У ALU реалізується механізм каскадного виконання найпростіших операцій для реалізації складних команд. Так, наприклад, при виконанні однієї з команд умовної передачі керування по результату порівняння в ALU тричі інкрементується лічильник команд (PC), двічі виробляється читання з RDM, виконується арифметичне порівняння двох змінних, формується 16-бітна адреса переходу і приймається рішення про те, робити чи не робити перехід по програмі. Усі перераховані операції виконуються усього лише за 2 мкс.

Важливою особливістю ALU є його здатність оперувати не тільки байтами, але і бітами. Окремі програмно-доступні біти можуть бути встановлені, скинуті, інвертовані, передані, перевірені і використані в логічних операціях. Ця здатність досить важлива, оскільки для керування об'єктами часто застосовуються алгоритми, що містять операції над вхідними і вихідними булевими змінними, реалізація яких засобами звичайних мікропроцесорів пов`язана з деякими труднощями.

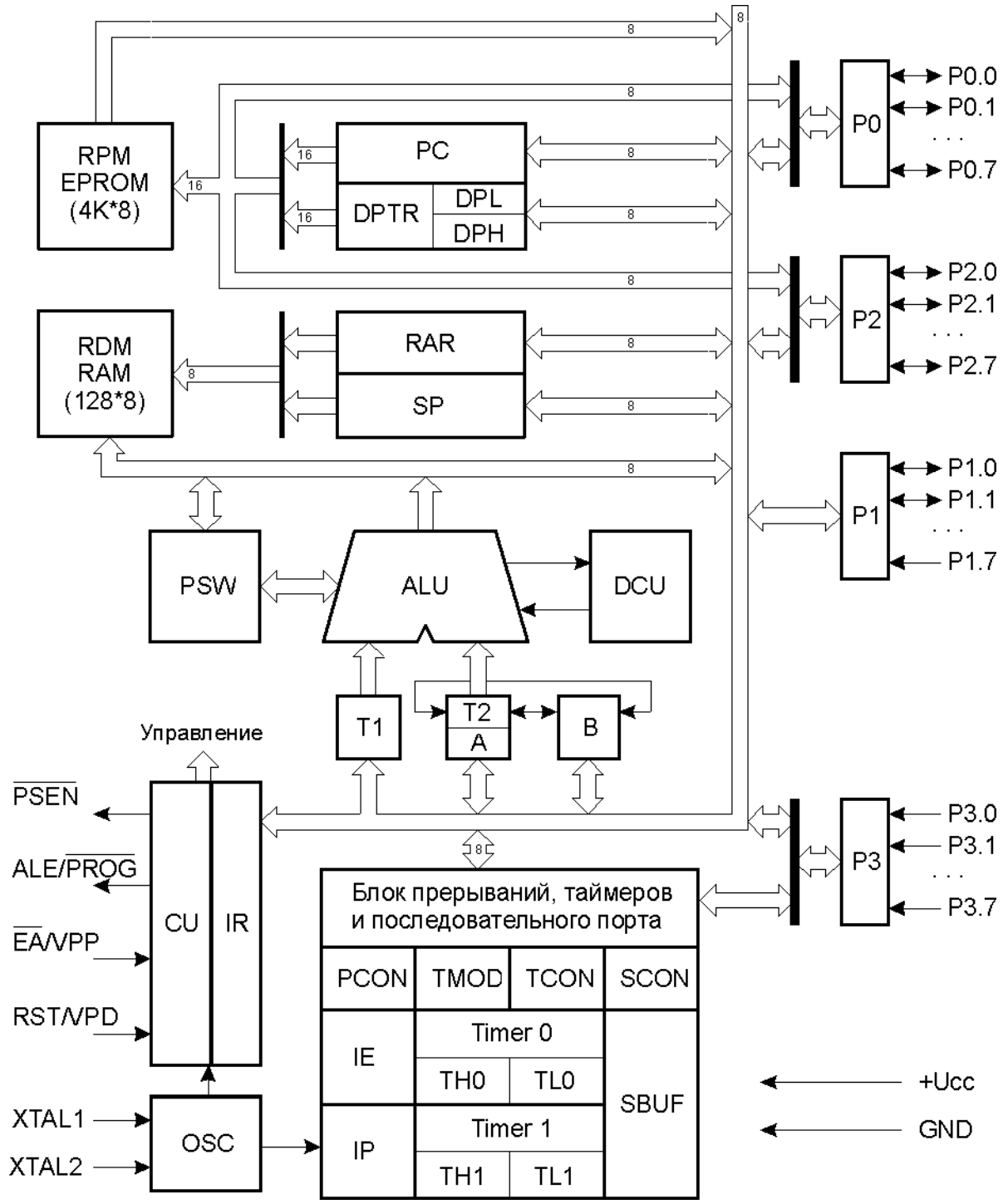


Рис. 1. Структурна схема мікроконтролера INTEL 8051

Таким чином, ALU може оперувати чотирма типами інформаційних об'єктів: булевими (1 біт), цифровими (4 біти), байтними (8 біт) і адресними (16 біт). У ALU виконується 51 різна операція пересилання або перетворення цих даних. Так як використовується 11 режимів адресації (7 для даних і 4 для адрес), то шляхом комбінування операції і режиму адресації базове число команд 111 розширюється до 255 з 256 можливих при однобайтному коді операції.

Резидентна пам'ять програм і даних

Резидентні (розміщені на кристалі) пам'ять програм (RPM) і пам'ять даних (RDM) фізично і логічно розділені, мають різні механізми адресації, працюють під керуванням різних сигналів і виконують різні функції.

Пам'ять програм RPM має ємність 4 Кбайта і призначена для збереження команд, констант, що керують словами ініціалізації, таблиць перекодування вхідних і вихідних перемінних і т.п. Пам'ять має 16-бітну шину адреси, через яку забезпечується доступ з програмного лічильника PC чи з регістра-покажчика даних (DPTR). DPTR виконує функції базового регістра при непрямих переходах по або програмі використовується в операціях з таблицями.

Пам'ять даних RDM призначена для збереження змінних у процесі виконання прикладної програми, адресується одним байтом і має ємність 128 байт. Крім того, до її адресного простору примикають адреси регістрів спеціальних функцій, що перераховані в табл. 1.

Пам'ять програм, так само як і пам'ять даних, може бути розширена до 64 Кбайт шляхом підключення зовнішніх мікросхем.

Таблиця 1

Блок регістрів спеціальних функцій

	Символ	Найменування		Адреса
*	A	Акумулятор		0E0H
*	B	Регістр-розширник акумулятора		0F0H
*	PSW	Слово стану програми		0D0H
	SP	Регістр-покажчик стека		81H
	DPTR	Регістр-покажчик даних	(DPH)	83H
			(DPL)	82H
*	P0	Порт 0		80H
*	P1	Порт 1		90H
*	P2	Порт 2		0A0H
*	P3	Порт 3		0B0H
*	IP	Регістр пріоритетів переривань		0B8H
*	IE	Регістр маски переривань		0A8H
	TMOD	Регістр режиму таймера/лічильника		89H
*	TCON	Регістр керування/статусу таймера		88H
	TH0	Таймер 0 (старший байт)		8CH
	TL0	Таймер 0 (молодший байт)		8AH
	TH1	Таймер 1 (старший байт)		8DH
	TL1	Таймер 1 (молодший байт)		8BH
*	SCON	Регістр керування прийомопередавачем		98H
	SBUF	Буфер прийомопередавача		99H
	PCON	Регістр керування потужністю		87H

Примітка. *Регістри, імена яких відзначені знаком (*), допускають адресацію окремих бітів.*

Акумулятор, регістри загального призначення і прапори

Акумулятор (A) є джерелом операнда і місцем фіксації результату при виконанні арифметичних, логічних операцій і ряду операцій передачі даних. Крім того, тільки з використанням акумулятора можуть бути виконані операції зрушень, перевірка на нуль, формування прапора паритету і т.п. У розпорядженні користувача маються 8 регістрів загального призначення R0–R7 одного з чотирьох можливих банків. При виконанні багатьох команд у ALU формується ряд ознак операції (прапорів), що фіксуються в регістрі PSW. У табл. 2 приводиться перелік прапорів PSW, даються їхні символічні імена й описуються умови їхнього формування.

Таблиця 2

Формат слова стану програми PSW

Символ	Розряд	Ім'я і призначення
C	PSW.7	Прапор переносу. Встановлюється і скидається апаратно чипрограмно при виконанні арифметичних і логічних операцій
AC	PSW.6	Прапор допоміжного переносу. Встановлюється і скидається тільки апаратно при виконанні команд додавання і вирахування і сигналізує про или перенос позиції в біті 3
F0	PSW.5	Прапор 0. Може бути встановлений, скинутий чи перевірений програмою як прапор, специфікуємий користувачем
RS1	PSW.4	Вибір банку регістрів. Встановлюється і скидається програмно для вибору робочого банку регістрів (табл. 3)
RS0	PSW.3	
OV	PSW.2	Прапор переповнення. Встановлюється і скидається апаратно при виконанні арифметичних операцій
-	PSW.1	Не використовується
P	PSW.0	Прапор паритету. Встановлюється і скидається апаратно в кожнім циклі і фіксує непарне/парне число одиничних бітів в акумуляторі, тобто виконує контроль по парності

Таблиця 3

Вибір робочого банку регістрів

RS1	RS0	Банк	Границі адрес
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Найбільш «активним» прапором PSW є прапор переносу, що бере участь і модифікується в процесі виконання безлічі операцій, включаючи додавання, віднімання і зрушення. Крім того, прапор переносу (C) виконує функції «булева акумулятора» у командах, що маніпулюють з бітами. Прапор переповнення (OV) фіксує арифметичне переповнення при операціях над цілими числами зі знаком і уможливлює використання арифметики в додаткових кодах. ALU не керує прапорами селекції банку регістрів (RS0, RS1), їхнє значення цілком визначається прикладною програмою і використовується для вибору одного з чотирьох реєстрових банків.

У мікропроцесорах, архітектура яких спирається на акумулятор, більшість команд працюють з ним, використовуючи неявну адресацію. У Intel 8051 справа обстоїть інакше. Хоча процесор має у своїй основі акумулятор, він може виконувати безліч команд і без його участі. Наприклад, дані можуть бути передані з будь-якої комірки RDM у будь-який регістр, будь-який регістр може бути завантажений безпосереднім операндом і т.д. Багато логічних операцій можуть бути виконані без участі акумулятора. Крім того, змінні можуть бути інкрементовані, декрементовані і перевірені без використання акумулятора. Прапори і керуючі біти можуть бути перевірені і змінені аналогічно.

Регістри-покажчики

8-бітний покажчик стека (SP) може адресувати будь-яку область RDM. Його вміст інкрементується перш, ніж дані будуть запам`ятовані в стеці в ході виконання команд PUSH і CALL. Зміст SP декрементується після виконання команд POP і RET. Подібний спосіб адресації елементів стека називають передінкрементним/постдекрементним. У процесі ініціалізації мікроконтролера після сигналу RST у SP автоматично завантажується код 07H. Це значить, що якщо прикладна програма не перевизначає стек, то перший елемент даних у стеці буде розташовуватися в комірку RDM з адресою 08H.

Двохбайтний регістр-покажчик даних DPTR звичайно використовується для фіксації 16-бітної адреси в операціях зі звертанням до зовнішньої пам'яті. Командами мікроконтролера регістр-покажчик даних може бути використаний як 16-бітний регістр або як два незалежних 8-бітних регістри (DPH і DPL).

Регістри спеціальних функцій

Регістри із символічними іменами IP, IE, TMOD, TCON, SCON і PCON використовуються для фіксації і програмної зміни керуючих бітів і біта стану схеми переривання, таймера/лічильника, прийомопередавача послідовного порту і для керування енергоспоживанням. Їхня організація буде описана нижче при розгляді особливостей роботи мікроконтролера в різних режимах.

Пристрій керування і синхронізації

Кварцевий резонатор, що підключається до зовнішніх виводів мікроконтролера, керує роботою внутрішнього генератора, що у свою чергу формує сигнали синхронізації. Пристрій керування (CU) на основі сигналів синхронізації формує машинний цикл фіксованої тривалості, рівний 12 періодам резонатора. Більшість

команд мікроконтролера виконується за один машинний цикл. Деякі команди, що оперують з 2-байтними словами або зв'язаними зі звертанням до зовнішньої пам'яті, виконуються за два машинних цикли. Тільки команди ділення і множення вимагають чотирьох машинних циклів. На основі цих особливостей роботи пристрою керування виробляється розрахунок часу виконання прикладних програм.

На схемі мікроконтролера до пристрою керування примикає регістр команд (IR). У його функцію входить збереження коду виконуваної команди.

Вхідні і вихідні сигнали пристрою керування і синхронізації:

- PSEN – дозвіл програмної пам'яті,
- ALE – вихідний сигнал дозволу фіксації адреси,
- PROG – сигнал програмування,
- EA – блокування роботи з внутрішньою пам'яттю,
- VPP – напруга програмування,
- RST – сигнал загального скидання,
- VPD – вивід резервного живлення пам'яті від зовнішнього джерела,
- XTAL – входи підключення кварцевого резонатора.

Паралельні порти вводу/виводу інформації

Усі чотири порти (P0-P3) призначені для вводу або виводу інформації побайтно. Кожен порт містить керовані регістр-засувку, вхідний буфер і вихідний драйвер.

Вихідні драйвери портів 0 і 2, а також вхідний буфер порту 0 використовуються при звертанні до зовнішньої пам'яті. При цьому через порт 0 у режимі тимчасового мультиплексування спочатку виводиться молодший байт адреси, а потім видається чи приймається байт даних. Через порт 2 виводиться старший байт адреси в тих випадках, коли розрядність адреси дорівнює 16 біт.

Усі виводи порту 3 можуть бути використані для реалізації альтернативних функцій, перерахованих у табл. 4. Ці функції можуть бути задіяні шляхом запису 1 у відповідні біти регістра-засувки (P3.0-P3.7) порту 3.

Таблиця 4

Альтернативні функції порту P3

Символ	Розряд	Ім'я і призначення
RD	P3.7	Читання. Активний сигнал низького рівня формується апаратно при звертанні до зовнішньої пам'яті даних
WR	P3.6	Запис. Активний сигнал низького рівня формується апаратно при звертанні до зовнішньої пам'яті даних
T1	P3.5	Вхід таймера/лічильника 1 або тест-вхід
T0	P3.4	Вхід таймера/лічильника 0 або тест-вхід
INT1	P3.3	Вхід запиту переривання 1. Сприймається сигнал низького рівня або зріз
INT0	P3.2	Вхід запиту переривання 0. Сприймається сигнал низького рівня або зріз
TXD	P3.1	Вихід передавача послідовного порту в режимі UART. Вихід синхронізації в режимі регістра зрушення

RXD	P3.0	Вхід приймача послідовного порту в режимі UART. Ввід/вивід даних у режимі регістра зрушення

Порт 0 є двонаправленим, а порти 1-3 - квазідвонаправленими. Кожна лінія портів може бути використана незалежно для або вводу/виводу.

По сигналі RST у регістри-засувки всіх портів автоматично записуються одиниці, що настроюють їх тим самим на режим вводу.

Усі порти можуть бути використані дляорганізації вводу/виводу інформації з двонаправленими лініями передачі. Однак порти 0 і 2 не можуть бути використані для цієї мети у випадку, якщо система має зовнішню пам'ять, зв'язок з який організується через загальну поділювану шину адреси/даних, що працює в режимі тимчасового мультиплексування.

Звертання до портів вводу/виводу можливо з використанням команд, що оперують з байтом, окремим бітом, довільною комбінацією бітів. При цьому в тих випадках, коли порт є одночасно операндом і місцем призначення результату, пристрій керування автоматично реалізує спеціальний режим, що називається «читання-модифікація-запис». Цей режим звертання припускає введення сигналів не з зовнішніх виводів порту, а з його регістра-засувки, що дозволяє виключити неправильне зчитування раніше виведеної інформації. Цей механізм звертання до портів реалізований у командах:

- ANL – логічне І, наприклад, ANL P1,A;
- ORL – логічне АБО, наприклад, ORL P2,A;
- XRL – виключаєче АБО, наприклад, XRL P3,A;
- JBC – перехід, якщо в адресуємому біті одиниця, і наступне скидання біта, наприклад, JBC P1.1, LABEL;
- CPL – інверсія біта, наприклад, CPL P3.3;
- INC – інкремент порту, наприклад, INC P2;
- DEC – декремент порту, наприклад, DEC P2;
- DJNZ – декремент порту і перехід, якщо його вміст не дорівнює нулю, наприклад, DJNZ r, LABEL;
- MOV PX.Y,C – передача біта переносу в біт Y порту X;
- SET PX.Y – установка біта Y порту X;
- CLR PX.Y – скидання біта Y порту X.

Таймер/лічильник

Ускладі мікроконтролера маються реєстрові пари із символічними іменами TH0, TL0 і TH1, TL1, на основі яких функціонують два незалежних програмно-керованих 16-бітних таймери/лічильника подій (T/C0 і T/C1). При роботі як таймер вміст T/C інкрементується в кожному машинному циклі, тобто через кожні 12 періодів резонатора. При роботі як лічильник вміст T/C інкрементується під впливом переходу з 1 у 0 зовнішнього вхідного сигналу, який подається на відповідний (T0, T1) вхід мікроконтролера. Опитування сигналів виконується в кожному машинному циклі. Тому що на розпізнавання переходу потрібно два машинних цикли, то максимальна частота підрахунку вхідних сигналів дорівнює 1/24 частоти резонатора. На тривалість періоду вхідних сигналів обмежень зверху немає. Для гарантованого прочитання

вхідного сигналу, що зчитується, він повинний утримувати значення 1 як мінімум протягом одного машинного циклу.

Для керування режимами роботи і дляорганізації взаємодії таймерів із системою переривання використовуються два регістри спеціальних функцій TMOD і TCON, опис яких приводиться в табл. 5-7. Для обох Т/С режими роботи 0, 1 і 2 однакові. Режими 3 для Т/С0 і Т/С1 різні.

Таблиця 5

Регістр режиму роботи таймера/лічильника

Символ	Розряд	Ім'я і призначення
GATE	TMOD.7 для Т/С1 TMOD.3 для Т/С0	Керування блокуванням. Якщо біт установлений, то таймер/лічильник «х» дозволений доти, поки на вході «INT х» високий рівень і біт керування «TRx» установлений. Якщо біт скинутий, то Т/З дозволяється, як тільки біт керування «TRx» установлюється
C/T	TMOD.6 для Т/С1 TMOD.2 для Т/С0	Біт вибору режиму таймера чи лічильника подій. Якщо біт скинутий, то працює таймер від внутрішнього джерела сигналів синхронізації. Якщо біт установлений, то працює лічильник від зовнішніх сигналів на вході «Тх»
M1	TMOD.5 для Т/С1 TMOD.1 для Т/С0	Режим роботи (див.табл. 6)
M0	TMOD.4 для Т/С1 TMOD.0 для Т/С0	

Таблиця 6

Режими роботи таймера/лічильника

M1	M0	Режим роботи
0	0	«TLx» працює як 5-бітний переддільник
0	1	16-бітний таймер/лічильник. «THx» і «TLx» включені послідовно
1	0	8-бітний автоперезавантажуємий таймер/лічильник. «THx» зберігає значення, що повинне бути перезавантажене в «TLx» щораз по переповненню
1	1	Таймер/лічильник 1 зупиняється. Таймер/лічильник 0: TL0 працює як 8-бітний таймер/лічильник, і його режим визначається керуючими бітами таймера 0. TH0 працює тільки як 8-бітний таймер, і його режим визначається керуючими бітами таймера 1

Режим 0. Переведення будь-якого Т/С у цей режим робить його 8-розрядним таймером, на вхід якого підключений 5-бітний переддільник частоти на 32. У цьому режимі таймерний регістр має розрядність 13 біт. При переході зі стану «всі одиниці» у

стан «усі нулі» установлюється прапор переривання від таймера TF1. Вхідний синхросигнал таймера 1 дозволений (надходить на вхід T/C), коли керуючий біт TR1 встановлений у 1 і, або керуючий біт GATE (блокування) дорівнює 0, або на зовнішній вхід запиту переривання INT1 надходить рівень 1.

Встановлення біта GATE у 1 дозволяє використовувати таймер для виміру тривалості імпульсного сигналу, який подається на вхід запиту переривання.

Таблиця 7

Регістр керування/статусу таймера

Символ	Розряд	Ім'я і призначення
TF1	TCON.7	Прапор переповнення таймера 1. Встановлюється апаратно при переповненні таймера/лічильника. Скидається при обслуговуванні переривання апаратно
TR1	TCON.6	Біт керування таймера 1. Встановлюється/скидається програмою дляпуску/останова
TF0	TCON.5	Прапор переповнення таймера 0. Встановлюється апаратно. Скидається при обслуговуванні переривання
TR0	TCON.4	Біт керування таймера 0. Встановлюється/скидається програмою дляпуску/останова таймера/лічильника
IE1	TCON.3	Прапор фронту переривання 1. Встановлюється апаратно, коли детектується зріз зовнішнього сигналу INT1. Скидається при обслуговуванні переривання
IT1	TCON.2	Біт керування типом переривання 1. Встановлюється/скидається програмно для специфікації запиту INT1 (зріз/низький рівень)
IE0	TCON.1	Прапор фронту переривання 0. Встановлюється по зрізі сигналу INT0. Скидається при обслуговуванні переривання
IT0	TCON.0	Біт керування типом переривання 0. Встановлюється/скидається програмно для специфікації запиту INT0 (зріз/низький рівень)

Режим 1. Робота будь-якого T/C у цьому режимі така ж, як і в режимі 0, за винятком того, що таймерний регістр має розрядність 16 біт.

Режим 2. У цьому режимі робота організована таким чином, що переповнення (перехід зі стану «всі одиниці» у стан «усі нулі») 8-бітного лічильника TL1 приводить не тільки до встановлення прапора TF1, але й автоматично перезавантажує в TL1 зміст старшого байта (TH1) таймерного регістра, що попередньо було задано програмним шляхом. Перезавантаження залишає вміст TH1 незмінним. У режимі 2 T/C0 і T/C1 працюють зовсім однаково.

Режим 3. У цьому режимі T/C0 і T/C1 працюють по-різному. T/C1 зберігає незмінним свій поточний вміст. Іншими словами, ефект такий ж, як і при скиданні керуючого біта TR1 у нуль. У цьому режимі TL0 і TH0 функціонують як два незалежних 8-бітних лічильники. Роботу TL0 визначають керуючі біти T/C0 (C/T,

GATE, TR0), вхідний сигнал INT0 і прапор переповнення TF0. Роботу TH0, що може виконувати тільки функції таймера (підрахунок машинних циклів мікроконтролера), визначає керуючий біт TR1. При цьому TH0 використовує прапор переповнення TF1.

Режим 3 використовується в тих випадках, коли потрібно наявність додаткового 8-бітного таймера чи лічильника подій. Можна вважати, що в режимі 3 мікроконтролер має у своєму складі три таймери/лічильника. У тому випадку, якщо T/C0 використовується в режимі 3, T/C1 може бути включений, виключений, чи переведений у свій власний режим 3, може бути використаний послідовним портом як генератор частоти передачі, чи, нарешті, може бути використаний у будь-якім застосуванні, що не вимагає переривання.

Послідовний порт

Через універсальний асинхронний прийомопередавач UART (Universal Asynchronous Receiver-Transmitter) відбувається передача інформації, представленої послідовним кодом (молодшими бітами вперед), у повному дуплексному режимі обміну. До складу UART, який називають часто послідовним портом, входять приймаючий і передавальний, що зрушують регістри, а також спеціальний буферний регістр (SBUF) прийомопередавача.

1.1.1. Регістр SBUF

Являє собою два незалежних регістри: буфер приймача і буфер передавача. Завантаження байта в SBUF негайно викликає початок процесу передачі через послідовний порт. Коли байт зчитується з SBUF, це значить, що його джерелом є приймач послідовного порту. Запис байта в буфер приводить до автоматичного перепису байта в регістр передавача, що зрушує і ініціює початок передачі байта. Наявність буферного регістра приймача дозволяє сполучати операцію читання раніше прийнятого байта з прийомом чергового байта. Якщо домоменту закінчення прийому байта попередній байт не був лічений, то він буде загублений.

Послідовний порт може працювати в чотирьох різних режимах.

Режим 0. Інформація передається і приймається через вхід приймача RXD. Приймаються і передаються 8 біт даних. Через зовнішній вихід передавача TXD видаються імпульси зрушення, що супроводжують кожен біт. Частота передачі дорівнює 1/12 частоти резонатора.

Режим 1. Через TXD передаються з RXD чи приймаються 10 біт: старт-біт (0), 8 біт даних і стоп-біт (1). Швидкість прийому/передачі – величина змінна і задається таймером.

Режим 2. Через TXD передаються з RXD чи приймаються 11 біт: старт-біт, 8 біт даних, програмувальний дев'ятий біт і стоп-біт. При передачі дев'ятий біт може використовуватися для підвищення імовірності передачі шляхом контролю по парності й у нього можна помістити значення ознаки паритету з PSW. Частота прийому/передачі вибирається програмно і може дорівнювати 1/32 чи 1/64 частоти резонатора в залежності від SMOD.

Режим 3. Збігається з режимом 2, але частота прийому/передачі є величиною змінної і задається таймером.

1.1.2. Регістр SCON

Регістр призначений для керування режимом роботи UART. Регістр містить керуючі біти і дев'ятий біт прийнятих чи переданих даних RB8 і TB8, а також біти переривання прийомопередавача RI і TI. Функціональне призначення бітів зазначене в табл. 8 і 9.

Таблиця 8

Регістр управління/статусу UART

Символ	Розряд	Ім'я і призначення
SM0	SCON.7	Біти керування режимом роботи UART. Встановлюються/скидаються програмно (табл. 9)
SM1	SCON.6	
SM2	SCON.5	Біт керування режимом UART. Встановлюється програмно для заборони прийому повідомлення, у якому дев'ятий біт дорівнює 0
REN	SCON.4	Біт дозволу прийому. Встановлюється/скидається програмно для дозволу/заборони прийому послідовних даних
TB8	SCON.3	Передача біта 8. Встановлюється/скидається програмно для завдання дев'ятого переданого біта в режимі UART - 9 біт
RB8	SCON.2	Прийом біта 8. Встановлюється/скидається апаратно для фіксації дев'ятого прийнятого біта в режимі UART - 9 біт
TI	SCON.1	Прапор переривання передавача. Встановлюється апаратно при закінченні передачі байта. Скидається програмно після обслуговування переривання
RI	SCON.0	Прапор переривання приймача. Встановлюється апаратно при прийомі байта. Скидається програмно після обслуговування переривання

Таблиця 9

Режим роботи UART

SM0	SM1	Режим роботи UART
0	0	Зрушуючий регістр розширення вводу/виводу
0	1	UART - 8 біт. Змінювана швидкість передачі
1	0	UART - 9 біт. Фіксована швидкість передачі
1	1	UART - 9 біт. Змінювана швидкість передачі

Прикладна програма шляхом завантаження в два старших розряди SCON визначає режим роботи UART. В усіх режимах передача ініціюється будь-якою командою, де SBUF зазначений як одержувач байта. Прийом до UART у режимі 0

відбувається за умови RI=0 і REN=1. У режимах 1-3 прийом починається з приходом старт-біта, якщо REN=1.

У TB8 програмно встановлюється значення дев'ятого біта даних, що буде переданий у режимі 2 чи 3. У RB8 фіксується в режимах 2 і 3 дев'ятий прийнятий біт даних. У режимі 1, якщо SM2=0, у біт RB8 заноситься стоп-біт. У режимі 0 RB8 не використовується.

Прапор преривання передавача TI встановлюється апаратно наприкінці періоду передачі восьмого біта даних урежимі 0 і на початку періоду передачі стоп-бита в режимах 1-3. Підпрограма обслуговування цього преривання повинна скидати біт TI.

Прапор преривання приймача RI встановлюється апаратно наприкінці періоду прийому восьмого біта даних у режимі 0 і в середині періоду прийому стоп-біта до режимах 1-3. Підпрограма обслуговування преривання повинна скидати біт RI.

1.1.3. Робота UART у мультиконтролерних системах

У системах децентралізованого керування, що використовуються для керування і регулювання в топологічно розподілених об'єктах, виникає задача обміну інформацією між безліччю мікроконтролерів, об'єднаних у локальну обчислювально-керуючу мережу. Як правило, локальні мережі на основі Intel 8051 мають магістральну архітектуру з поділюваним моноканалом (коаксіальний кабель, кручена пара, оптичне волокно), по якому здійснюється обмін інформацією між контролерами.

Біт SM2 у SCON дозволяє простими засобами реалізувати міжконтролерний обмін. Механізм обміну побудований на тім, що в режимах 2 і 3 програмувальний дев'ятий біт даних при прийомі фіксується в біті RB8. UART може бути запрограмований таким чином, що при одержанні стоп-біта преривання від приймача буде можливе тільки за умови RB8=1. Ведучий контролер усім відомим передає широкомовне повідомлення з байтом-ідентифікатором абонента, що відрізняється від байтів даних тільки тим, що в його дев'ятому біті міститься 1. Відомі за цією ознакою викликають підпрограми порівняння байта-ідентифікатора з кодом власної мережної адреси. Адресуємий контролер скидає свій SM2 і готується до прийому блоку даних. Інші відомі мікроконтролери залишають незмінними свої SM2=1 і передають керування основній програмі. При SM2=1 інформаційні байти в мережі преривання не викликають.

У режимі 1 автономного мікроконтролера SM2 використовується для контролю істинності стоп-біта. У режимі 0 SM2 не використовується і повинний бути скинутий.

1.1.4. Швидкість прийому/передачі

Швидкість залежить від режиму роботи UART. Урежимі 0 частота залежить тільки від резонатора: $f_0=f_{рез}/12$. За один машинний цикл передається один біт.

У режимах 1-3 швидкість залежить від значення керуючого біта SMOD у регістрі спеціальних функцій PCON (табл. 10).

Урежимі 2 частота передачі $f_2=(2^{SMOD}/64)f_{рез}$.

У режимах 1 і 3 у формуванні частоти передачі крім керуючого біта SMOD бере участь таймер 1. При цьому частота передачі залежить від частоти переповнення (OVT1) і визначається в такий спосіб: $f_{1,3}=(2^{SMOD}/32)f_{OVT1}$. Переривання від таймера 1 у цьому випадку повинне бути заблоковано. Сам T/C1 може працювати і як таймер, і як лічильник подій у кожнім із трьох режимів. Однак найбільше зручно використовувати

режим таймера з автоперезавантаженням (старша тетрада TMOD=0010B). При цьому частота передачі визначається вираженням $f_{1,3}=(2^{SMOD}/32)(f_{рез}/12)(256-(TH1))$. У табл. 11 приводиться опис способів настроювання Т/С1 для одержання типових частот передачі даних через UART.

Таблиця 10

Регістр керування потужністю PCON

Символ	Розряд	Найменування і функція
SMOD	PCON.7	Подвоєна швидкість передачі. Якщо біт встановлений у 1, то швидкість передачі вдвічі більше, ніж при SMOD=0
-	PCON.6-4	Не використовуються
GF1	PCON.3	Прапори, специфікуємі користувачем (прапори загального призначення)
GF0	PCON.2	
PD	PCON.1	Біт зниженої потужності. При встановленні в 1 мікроконтролер переходить у режим зниженого енергоспоживання
IDL	PCON.0	Біт холостого ходу. Якщо біт встановлений у 1, то мікроконтролер переходить урежим холостого ходу

Примітка. *При одночасному записі 1 у PD і IDL біт PD має перевагу. Скидання PCON виконується шляхом завантаження в нього коду 0XXX0000.*

Таблиця 11

Настроювання таймера 1 для керування частотою роботи UART

Частота прийому/ передачі (BAUD RATE)		Частота резонатора, МГц	SMOD	Таймер/лічильник 1		
				C/T	Режим (MODE)	Число, що перезаванта жується
Режим 0, макс.:	1 МГц	12	X	X	X	X
Режим 2, макс.:	375 кгц	12	1	X	X	X
Режими 1,3:	62,5 кгц	12	1	0	2	0FFH
	19,2 кгц	11,059	1	0	2	0FDH
	9,6 кгц	11,059	0	0	2	0FDH
	4,8 кгц	11,059	0	0	2	0FAH
	2,4 кгц	11,059	0	0	2	0F4H
	1,2 кгц	11,059	0	0	2	0E8H
	137,5 Гц	11,059	0	0	2	1DH
	110 Гц	6	0	0	2	72H
	110 Гц	12	0	0	1	0FEEBH

Система преривань

Зовнішні преривання INT0 і INT1 (мал. 2) можуть бути викликані рівнем чи переходом сигналу з 1 у 0 на входах мікроконтролера в залежності від значень керуючих бітів IT0 і IT1 у регістрі TCON. Від зовнішніх преривань установлюються прапори IE0 і IE1 у регістрі TCON, що ініціюють виклик відповідної підпрограми обслуговування преривання. Скидання цих прапорів виконується апаратно тільки в тому випадку, якщо преривання було викликано по переходу (зрізу) сигналу. Якщо ж переривання викликане рівнем вхідного сигналу, то скиданням прапора IE керує відповідна підпрограма обслуговування преривання шляхом впливу на джерело преривання з метою зняття їм запиту.

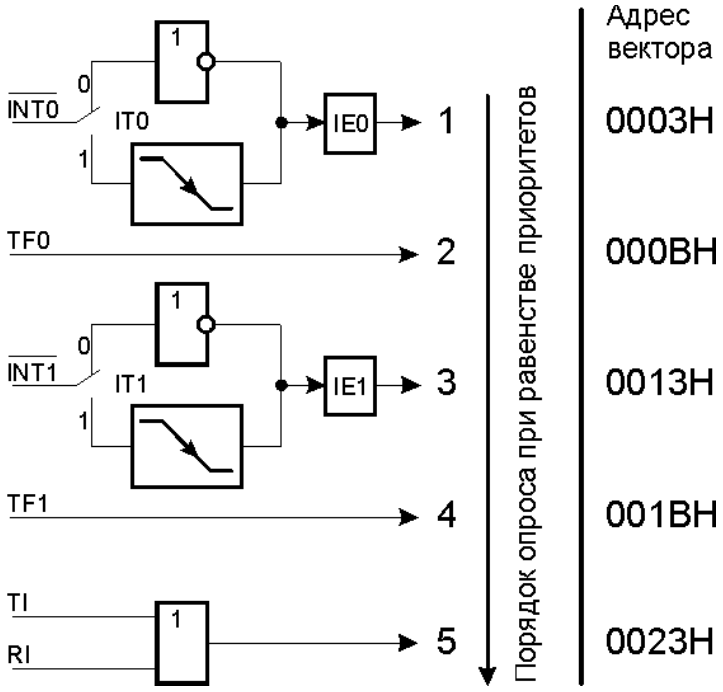


Рис. 2. Схема преривань

Прапори запитів преривання від таймерів TF0 і TF1 скидаються автоматично при передачі керування підпрограмі обслуговування. Прапори запитів преривання RI і TI установлюються UART апаратно, але скидатися повинні програмою. Преривання можуть бути викликані чи скасовані програмою, тому що всі перераховані прапори програмно доступні.

У блоці регістрів спеціальних функцій є два регістри, призначених для керування режимом преривань і рівнями пріоритету. Формати цих регістрів, що мають символічні імена IE і IP описані в табл. 12 і 13 відповідно.

Таблиця 12

Регістр масок преривання IE

Символ	Розряд	Ім'я і призначення
EA	IE.7	Зняття блокування преривань. Скидається програмно для заборони всіх преривань незалежно від станів IE4-IE0

-	IE.6, 5	Не використовуються
ES	IE.4	Біт дозволу преривання від UART. Встановлення/скидання програмою для дозволу/заборони преривань від прапорів TI, RI
ET1	IE.3	Біт дозволу преривання від таймера 1. Встановлення/скидання програмою для дозволу/заборони преривань від таймера 1
EX1	IE.2	Біт дозволу зовнішнього преривання 1. Встановлення/скидання програмою для дозволу/заборони переривань
ET0	IE.1	Дозвіл преривання від таймера 0. Працює аналогічно IE.3
EX0	IE.0	Дозволу зовнішнього преривання 0. Працює аналогічно IE.2

Таблиця 13

Регістр пріоритетів преривання IP

Символ	**Розряд**	**Ім'я і призначення**
-	IP.7-5	Не використовуються
PS	IP.4	Біт пріоритету UART. Встановлення/скидання програмою для призначення перериванню від UART вищого/нижчого пріоритету
PT1	IP.3	Біт пріоритету таймера 1. Встановлення/скидання програмою для призначення перериванню від таймера 1 вищого/нижчого пріоритету
PX1	IP.2	Біт пріоритету зовнішнього преривання 1. Встановлення/скидання програмою для призначення перериванню INT1 вищого/нижчого пріоритету
PT0	IP.1	Біт пріоритету таймера 0. Працює аналогічно IP.3
PX0	IP.0	Пріоритет зовнішнього преривання 0. Працює аналогічно IP.2

Можливість програмної встановлення/скидання будь-якого керуючого біта в цих двох регістрах робить систему преривань винятково гнучкої.

Прапори преривань опитуються в кожнім машинному циклі. Ранжирування преривань по пріоритеті виконується протягом наступного машинного циклу. Система переривань сформує апаратно виклик LCALL відповідної підпрограми обслуговування, якщо вона не заблокована однією з умов:

- у даний момент обслуговується запит преривання рівного чи більш високого рівня пріоритету;
- поточний машинний цикл – не останній у циклі виконуваної команди;
- виконується команда RETI чи будь-яка команда, зв'язана зі звертанням до регістрів IE чи IP.

Примітка. *Якщо прапор преривання був установлений, але по одній з перерахованих умов не одержав обслуговування і до моменту закінчення блокування вже був скинутий, то запит переривання губиться.*

По апаратно сформованому коді команди LCALL система преривання поміщає в стек зміст програмного лічильника PC і завантажує в PC адреса вектора преривання відповідної підпрограми обслуговування. По цій адресі повинна бути розташована команда безумовного переходу JMP до початкової адреси підпрограми обслуговування преривання. Ця підпрограма в разі потреби повинна починатися командами запису в стек PUSH слова стану програми PSW, акумулятора A, розширника акумулятора B, покажчика даних DPTR і т.д. і закінчуватися командами відновлення зі стека POP. Підпрограми обслуговування преривання обов'язково завершуються командою RETI, по якій у програмний лічильник перезавантажується зі стека збережена адреса повернення в основну програму. Команда RET також повертає керування, але при цьому не знімає блокування переривання.

ВВЕДЕННЯ В PROVIEW

ProView фірми Franklin Software Inc. – інтегроване середовище розробки програмного забезпечення для однокристальних мікроконтролерів сімейства Intel 8051 і його клонів. Вона містить у собі усе, що потрібно для створення, редагування, компіляції, трансляції, компонування, завантаження і налагодження програм:

- стандартний інтерфейс Windows,
- полнофункціональний редактор вихідних текстів з виділенням синтаксичних елементів кольором,
- організатор проекту,
- транслятор з мови С,
- асемблер,
- відладчик,
- вбудовану довідкову систему.

Середовище розробки подібне Visual C++ Microsoft і Borland C++ для Windows. Користувачі, знайомі з кожним з цих виробів, будуть почувати себе в ProView, як вдома.

Перший етап розробки програми – запис її вихідного тексту на якій-небудь мові програмування.

Потім виробляється компіляція чи трансляція його в коди із системи команд мікроконтролера, використовуючи транслятор чи асемблер. Транслятори й асемблери – прикладні програми, що інтерпретують текстовий файл, що містить вихідний текст програми, і створюють об'єктні файли, що містять об'єктний код.

Після компонування об'єктних модулів настає етап налагодження програми, усунення помилок, оптимізації і тестування програми.

ProView поєднує всі етапи розробки прикладної програми в єдиний рекурсивний процес, коли в будь-який момент часу можливе швидке повернення до будь-якому попередньому етапу.

ProView має наступні компоненти.

Оптимізуючий крос-компілятор C51

Мова С - універсальна мова програмування, що забезпечує ефективність коду, елементи структурного програмування і має багатий набір операторів. Універсальність, відсутність обмежень реалізації роблять мову С зручним і ефективним засобом програмування для широкої розмаїтості задач. Безліч прикладних програм може бути написане легше й ефективніше мовою С, чим на інших більш спеціалізованих мовах.

C51 - повна реалізація стандарту ANSI (Американського національного інституту стандартів), наскільки це можливо для архітектури Intel 8051. C51 генерує код для всього сімейства мікроконтролерів Intel 8051. Транслятор сполучить гнучкість програмування мовою С з ефективністю коду і швидкодією асемблера.

Використання мови високого рівня С має наступні переваги над програмуванням на асемблері:

- глибокого знання системи команд процесора не потрібно, елементарне знання архітектури Intel 8051 бажано, але не необхідно;

- розподіл регістрів і способи адресації керуються цілком транслятором;
- краща читаність програми, використовуються ключові слова і функції, що більш властиві людській думці;
- час розробки програм і їхнього налагодження значно коротше в порівнянні з програмуванням на асемблері;
- бібліотечні файли містять багато стандартних підпрограм, що можуть бути включені в прикладну програму;
- існуючі програми можуть багаторазово використовуватися в нових програмах, використовуючи модульні методи програмування.

Макроасемблер A51

Асемблер A51 сполучимо з ASM51 Intel для всього сімейства мікроконтролерів Intel 8051. Асемблер транслює символічну мнемоніку в переміщуваний об'єктний код, що має високу швидкодію і малий розмір. Макрозасоби прискорюють розробку і заощаджують час, оскільки загальні послідовності можуть бути розроблені тільки один раз. Асемблер підтримує символічний доступ до всіх елементів мікроконтролера і перебудовує конфігурацію для кожного різновиду Intel 8051.

A51 транслює вихідний файл асемблера в переміщуваний об'єктний модуль. При налагодженні чи при включеній опції «Include debugging information» цей об'єктний файл буде містити повну символічну інформацію для відладчика/ чи імітатора внутрісхемного емулятора.

Компонувач L51

Компонувач поєднує один чи кілька об'єктних модулів в одну програму, що виконується. Компонувач розміщає зовнішні і загальні посилання, призначає абсолютні адреси переміщуваним сегментам програм. Він може обробляти об'єктні модулі, створені транслятором C51, асемблером A51, транслятором PL/M-51 Intel і асемблером ASM51 Intel.

Компонувач автоматично вибирає відповідні бібліотеки підтримки і зв'язує тільки необхідні модулі з бібліотек. Встановлення за замовчуванням для L51 обрані так, щоб вони підходили для більшості прикладних програм, але можна визначити і замовлені встановлення.

Відладчик/симулятор WinSim51

Відладчик вихідних текстів використовується з транслятором C51, асемблером A51, транслятором PL/M-51 Intel і асемблером ASM51 Intel. Відладчик/симулятор дозволяє моделювати більшість особливостей Intel 8051 без наявності апаратних засобів. Можна використовувати його для перевірки і налагодження прикладної програми перш, ніж будуть виготовлені апаратні засоби. При цьому моделюється широка розмаїтість периферійних пристроїв, включаючи послідовний порт, зовнішнє введення - вивід і таймери.

ШВИДКИЙ СТАРТ

«Швидкий старт» – це звичайний прийом розроблювачів сучасних програмних засобів. Ціль полягає в тому, щоб, не поглиблюючи поки в подробиці, дати новачку чи досить досвідченому користувачу перше представлення про програмний засіб, дати можливість швидко одержати конкретний результат. Повне представлення, знання й уміння з'являється пізніше в процесі роботи і вивчення довідкових матеріалів.

Як приклад візьмемо найпростішу програму, з яким починають вивчення мов програмування багато поколінь студентів. «Hello World» - програма з папки \Fsi\Examples\Hello\, що видає в послідовний порт (UART) мікроконтролера рядок символів «Hello World» («Привіт Світ»). Весь вихідний текст програми міститься у файлі hello.c:

```
/***************************/
/* YOUR FIRST 8051 PROGRAM */
/***************************/

#include <reg51.h> /* special function register declarations
            */
                /* for the intended 8051 derivative        */
#include <stdio.h> /* prototype declarations for I/O
functions*/

/****************/
/* main program */
/****************/
void main (void)  {  /* execution starts here after stack init
*/
SCON  = 0x50;     /* SCON: mode 1, 8-bit UART, enable rcvr  */
   TMOD |= 0x20; /* TMOD: timer 1, mode 2, 8-bit reload    */
   TH1 = 0xf3;        /* TH1:  reload value for 2400 baud
*/
   TR1 = 1;       /* TR1:  timer 1 run                 */
TI  = 1;     /* TI:   set TI to send first char of UART*/

printf ("Hello World\n");/* the 'printf' function call */

while (1) {       /* An embedded program does not stop and  */
;  /* ... */ /* never returns.  We've used an endless  */
   }              /* loop.  You may wish to put in your own */
}                 /* code were we've printed the dots (...) */
```

Першніжпочатирозробкупроекту, скопіюйтепапку \Fsi\Examples\Hello\ усвоюособиступапку. У цій папці знаходиться усього лише один файл hello.c.

Запуск ProView і створення файлу проекту

ProView запускається зі стартового меню Windows подібно іншим додаткам (рис.3). Якщо необхідно запустити програму з командного рядка, її синтаксис має вид: PV32 [projectfile], де projectfile - ім'я файлу проекту з розширенням [.PRJ].

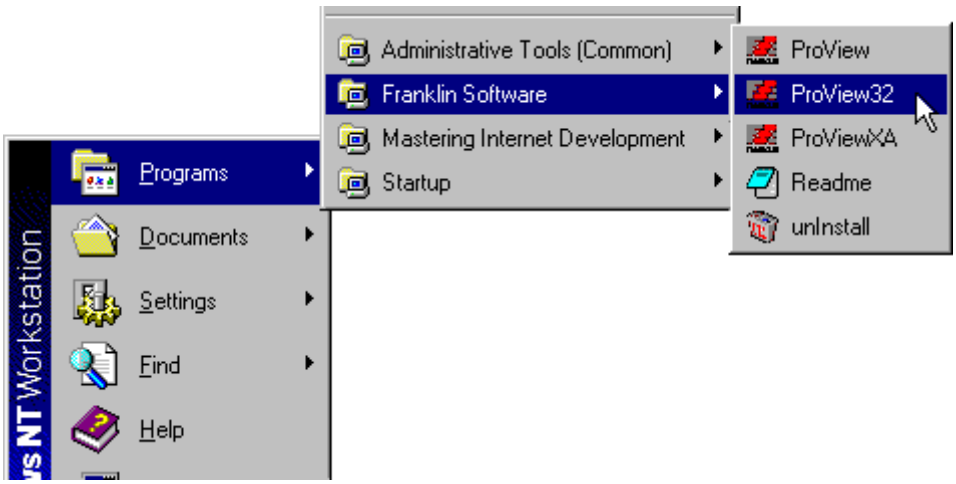


Рис. 3. Запуск програми

Будь-яка нова робота в ProView, як і у всіх сучасних компіляторах, починається зі створення нового файлу проекту. Файл проекту містить імена усіх вихідних файлів, зв'язаних із проектом, а також установки компіляції, трансляції і зв'язування файлів, щоб генерувати виконувану програму.

Для того щоб створити новий файл проекту, виберіть New з меню Project. Відкриється діалогове вікно New Project (рис. 4). Використовуйте кнопку Browse, щоб ввійти у свою папку. Знайдіть папку \Hello і натисніть кнопку [OK]. Потім виберіть «8051» як тип проекту.

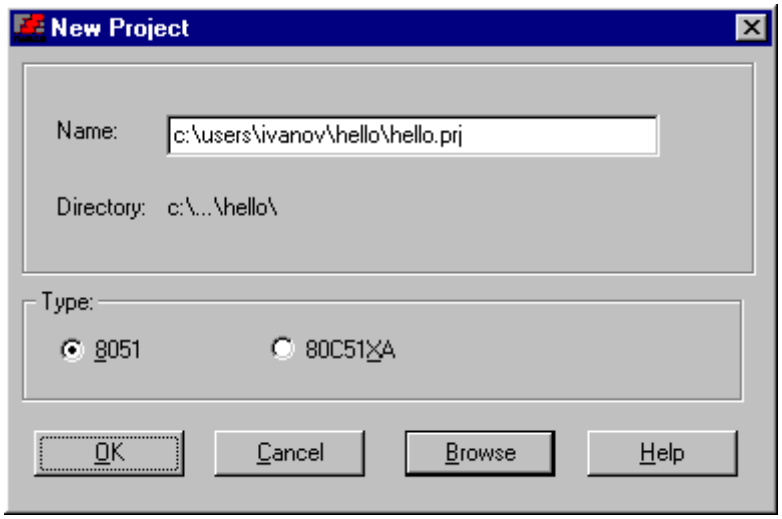


Рис. 4. Діалогове вікно New Project

Коли менеджер проекту відкриває файл проекту, вікно проекту показує включені вихідні файли. У даному випадку поки немає ніяких вихідних файлів. Мається тільки один вихідний файл, якому необхідно підключити - hello.c.

Добавка файлу з вихідним текстом і його редагування

Тепер можна додати hello.c до проекту. Виберіть Add file зменю Project. Відкриєтьсядіалоговевікно Add File (рис. 5). Виберіть hello.c зісписку.

Наш проект має тільки один вихідний файл. Надалі Ваші проекти, можливо, будуть складатися з безлічі вихідних файлів. Діалог Add File дозволить Вам вибрати і додати кілька файлів відразу. Для цього використовують комбінацію клавіші [CTRL] і покажчика миші. Коли Ви натиснете [Open], вихідні файли будуть додані до проекту в обраному порядку.

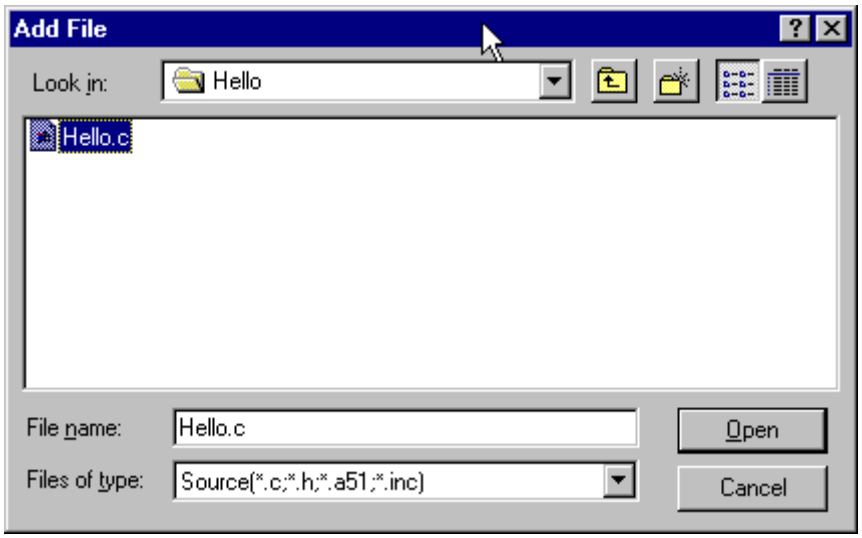


Рис. 5. Діалогове вікно Add File

Тепер можна редагувати текст із файлу hello.c. Виберіть hello.c з вікна Project (рис. 6). Натисніть його правою кнопкою миші і виберіть View source file, чи просто двічі клацніть мишею для того, щоб переглядати файл у вікні редагування.

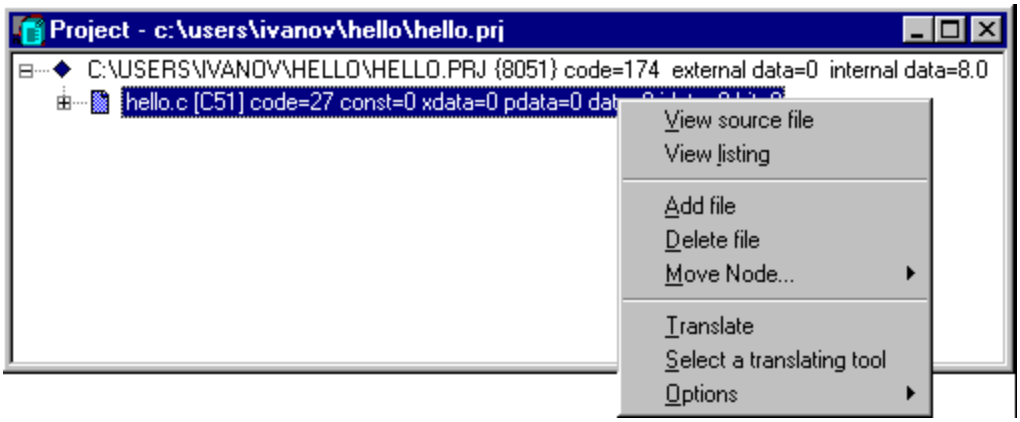


Рис. 6. Діалогове вікно Project

ProView завантажує і показує зміст hello.c у вікні, де можна редагувати файл. Вікно редагування (рис. 7) - полнофункціональний редактор вихідного тексту, що пропонує такі можливості, як высвічування синтаксичних елементів і контекстний пошук. Якщо вибрати «printf» і натиснути клавішу [F1], ProView відкриє систему довідки і перейде до розділу довідки про «printf».

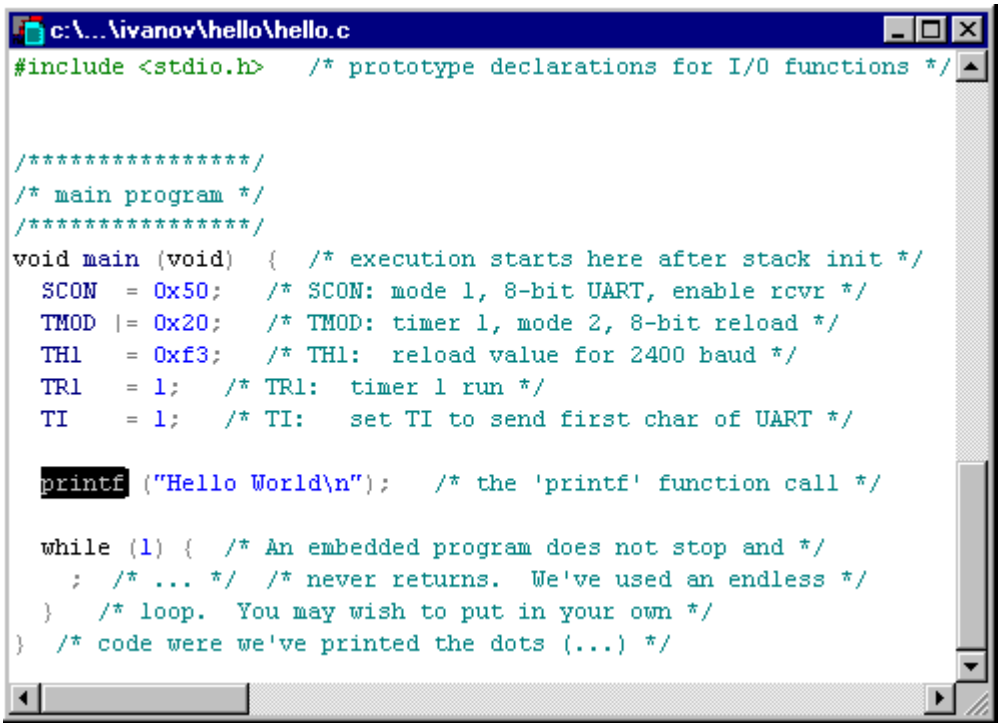


Рис. 7. Вікно редагування

Компіляція і компонування

Цей процес компілює, зв'язує hello.c з бібліотеками і створює абсолютний об'єктний модуль, що ми зможемо перевірити в відладчику WinSim.

Виберіть Make зменю Project. ProView відображає вікно, показуючи поточне стан процесу. Коли процес компіляції закінчиться, у вікні Message (рис. 8) відображається повідомлення завершення.

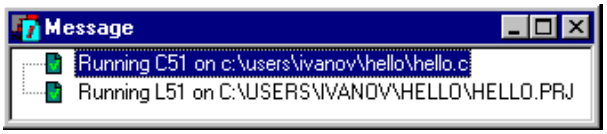


Рис. 1. Вікно повідомлень Message

Тестування і налагодження

Виконаємо налагодження програми. Якщо проект новий, відкриється діалогове вікно Debug Options (рис. 9), де Ви можете змінювати встановлення відладчика. Надалі можна установити опції відладчика, вибравши Debug з меню Options. Наш проект використовує значення за замовчуванням.

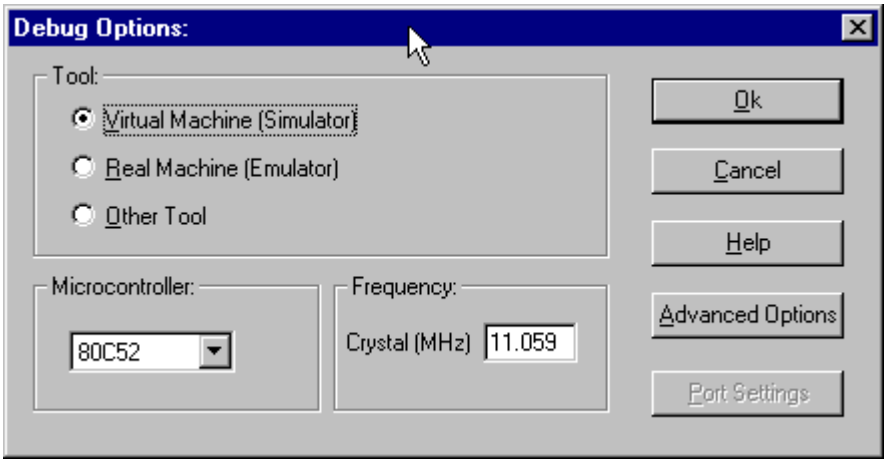


Рис. 9. Вікно діалогу опцій відладчика

Виберіть Start з меню Debug.

Виберіть Hardware (апаратні засоби) з меню View. Виберіть UART, відкриється вікно послідовного порту (рис. 10). Надалі при роботі програми тут можна буде побачити усе, що виводить мікроконтролер у послідовний порт.

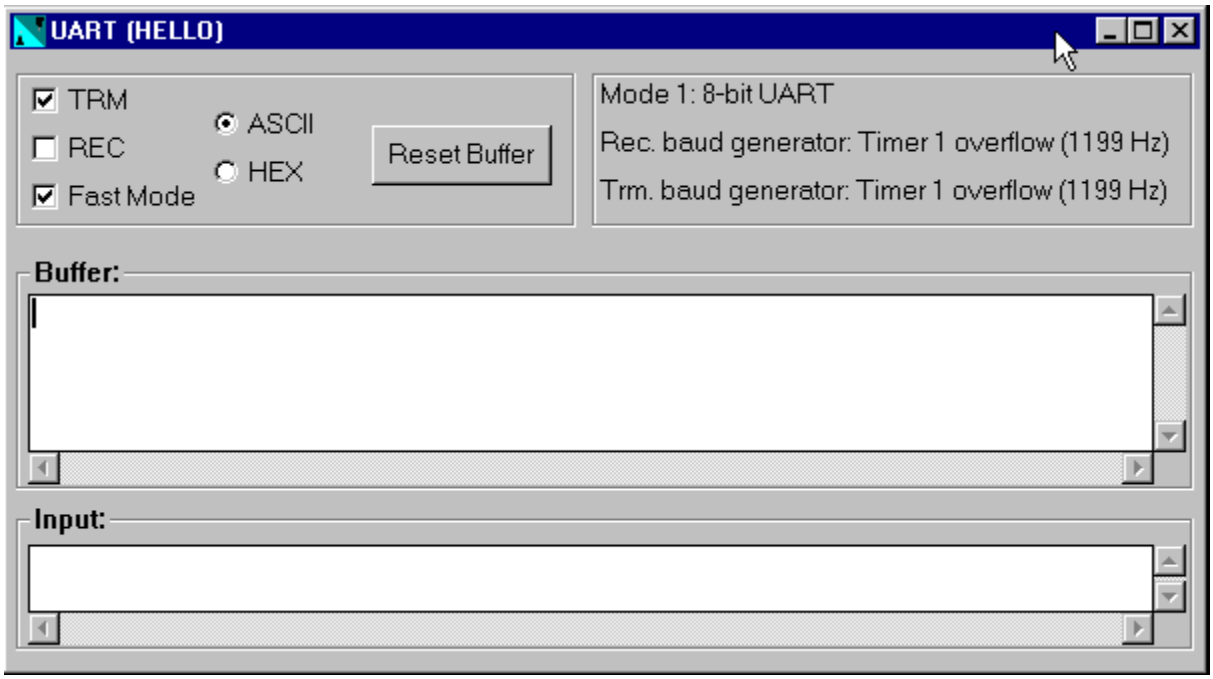


Рис. 2. Вікно послідовного порту

Виберіть Run з меню Debug чи натисніть кнопку 

.

Рис. 11 показує, як виглядає екран отладчика WinSim при виконанні програми. Зверніть увагу, що у вікно UART виведений текст «Hello World».

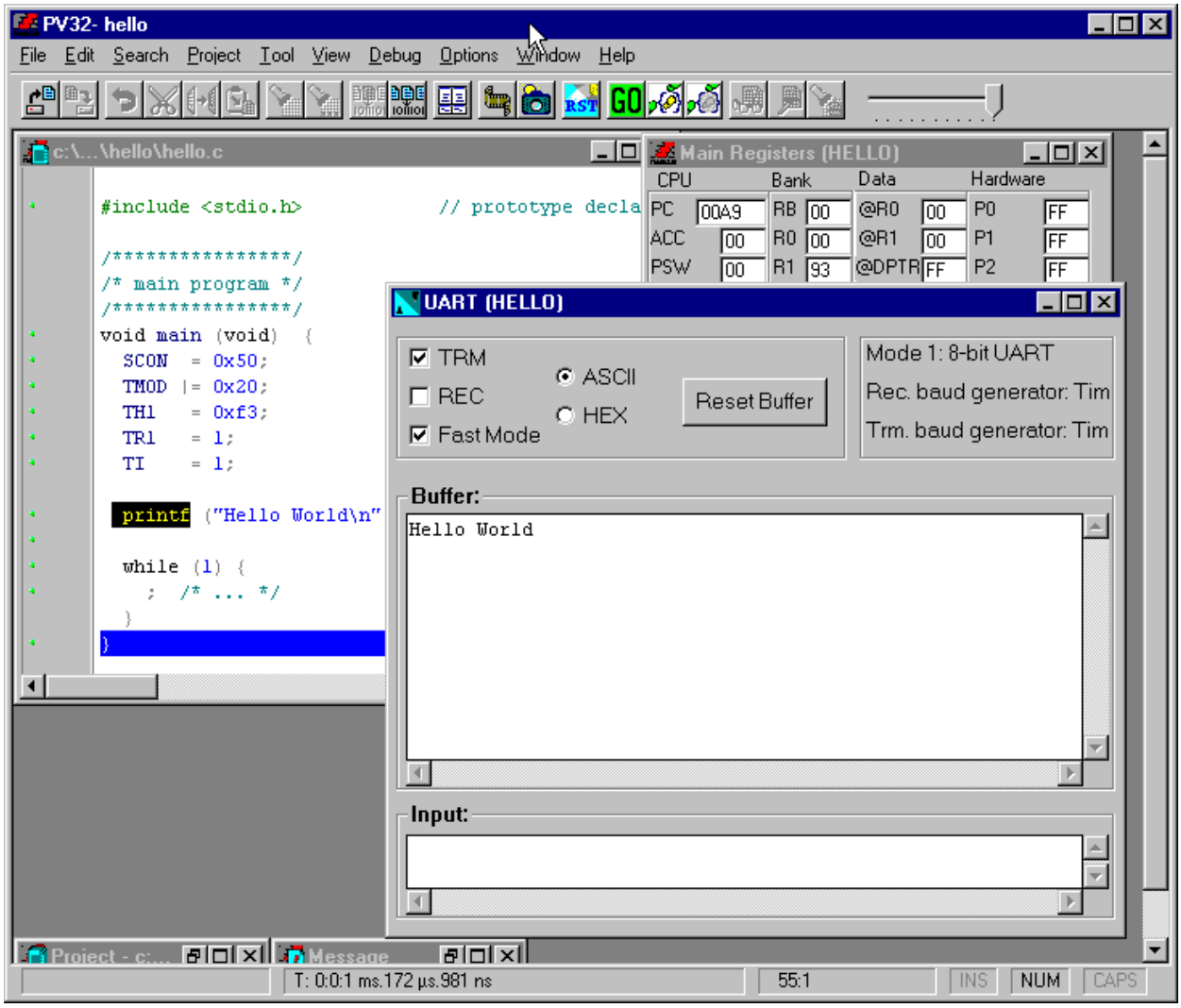


Рис. 3. Екран відладчика при виконанні програми

При виводі символів у порт починається виконання нескінченного циклу. Ви можете зупинити виконання програми, вибравши Stop з меню Debug. За допомогою регулятора  при натиснутій кнопці  на панелі інструментів можна змінювати швидкість роботи відладчика. Рядок стану показує поточний реальний час.

Покроковий режим і вихід з відладчика

Ви можете використовувати відладчик, щоб переміщатися по програмі. Виберіть Reset з меню Debug (ця команда скине моделюємий процесор) і виберіть Step Into і Step Over з меню Debug.

Команди Step дозволяють «крокувати» по кожнім рядку вихідного тексту. Поточна команда высвічується на кожнім кроці. Step Into дозволяє ввійти у викликувану функцію, Step Over – переступити через неї, не входячи в усередину.

Проробіть ці операції.

Для завершення роботи з відладчиком у будь-який момент часу Ви можете вибрати Terminate з меню Debug і повернутися в режим редагування.

Зверніть увагу, що в режимі налагодження на екрані видні ще два вікна. Перше – вікно коду (рис. 12), де в покроковому режимі паралельно з вихідним текстом мовою С йде трасування тексту на асемблері.

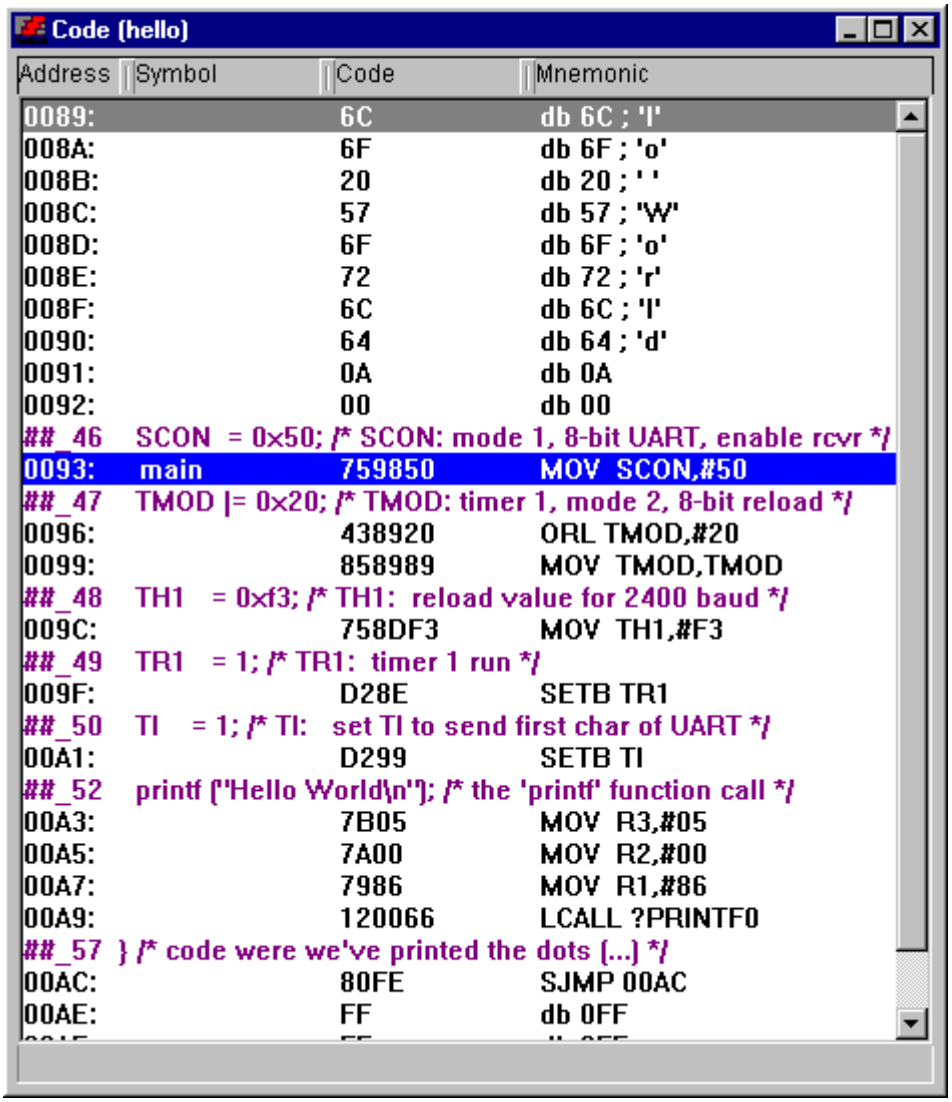


Рис. 4. Вікно коду

Прокрутить вікно коду і вивчить асемблерний аналог вихідного тексту. Із символів «##» починаються рядки, за допомогою яких легко зіставити асемблерний текст і текст мовою С. Зверніть увагу на те, скільки коду довелося б написати, якщо проектувати програму на асемблері.

Асемблерний аналог тексту зберігається уфайлі hello.lst, якщо в опціях проекту (Project з меню Options) відзначене Generate Listing (мал. 13). Тут же можна вказати, яку інформацію включати в листинг.

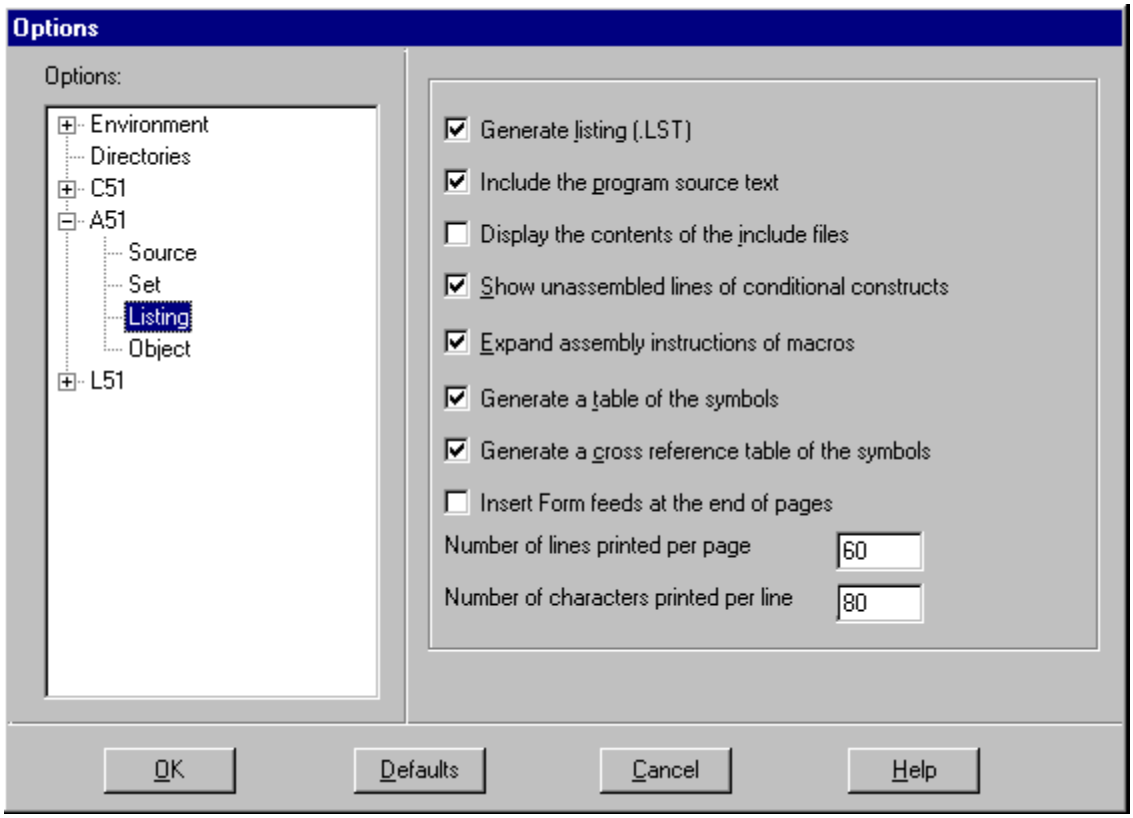


Рис. 5. Діалог опцій проекту

Вивчите зміст інших опцій проекту в розділах Environment, C51, A51, L51. Відкрийте файл листинга (рис. 14) за допомогою View listing з меню View.

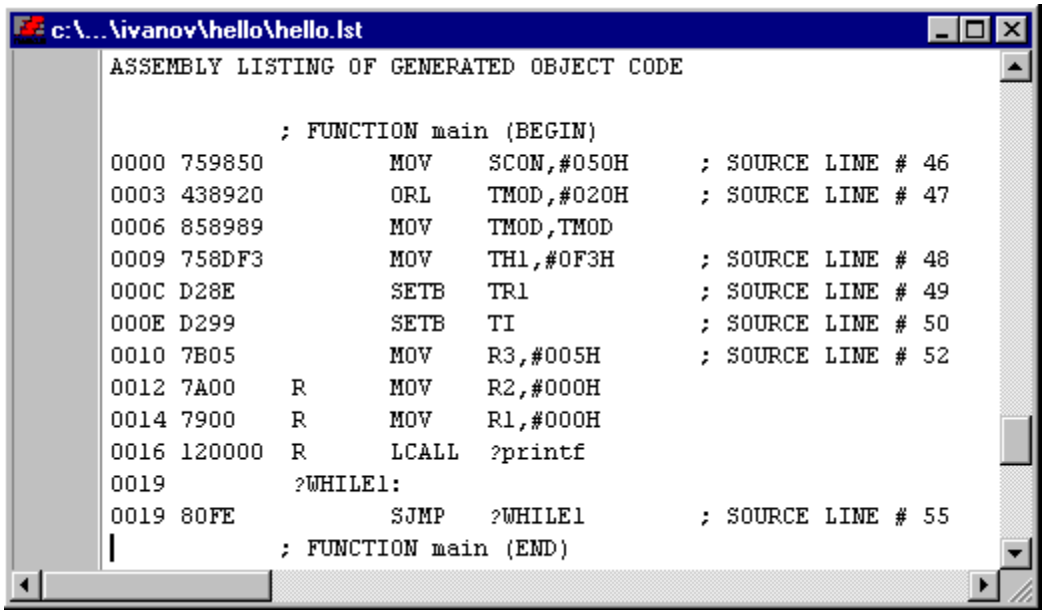


Рис. 6. Вікно файлу листинга

Вивчить і потрібно зрозуміти зміст розділів файлу листинга.

Друге вікно, що є присутнім на екрані під час налагодження, – вікно регістрів Main Registers (рис. 15).

У цьому вікні постійно відображається поточний стан усіх програмно-доступних регістрів мікроконтролера. Більш того, вміст регістрів можна змінювати під час налагодження.

За допомогою пункту Data dump з меню View можна подивитися вміст пам'яті різного типу в режимі налагодження. Спробуйте це зробити.

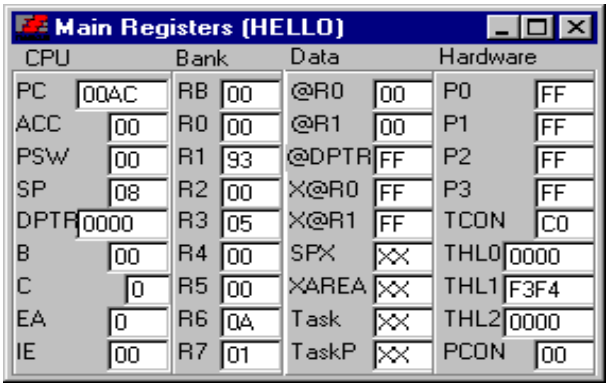


Рис. 75. Вікно регістрів Main Registers

ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Для заглибленого вивчення можливостей ProView і її компонентів самостійно вивчить зміст усіх пунктів меню, кнопок інструментальної панелі, вікон і настроювань. Для цього скористайтеся убудованою довідковою системою, що викликається через меню Help. Ці знання будуть потрібні при виконанні наступних лабораторних робіт.

ЗМІСТ ЗВІТУ

Звіт про лабораторну роботу повинний містити:

- титульний лист;
- мета і задачі роботи;
- структурну схему мікроконтролера;
- текст програми на асемблері з коментарями кожної команди;
- висновки по роботі.

Практична робота №2
СИСТЕМА КОМАНД МІКРОКОНТРОЛЕРА INTEL8051

МЕТА РОБОТИ

Метою роботи є вивчення системи команд мікроконтролерів популярного сімейства Intel8051, а також продовження початого в лабораторній роботі №1 вивчення інтегрованого середовища ProView фірми Franklin Software Inc., що призначена для розробки програмного забезпечення цього сімейства. Робота розрахована на 4 години

домашньої підготовки і 4 години занять у лабораторії.

При підготовці до роботи вивчається система команд одного з клонів сімейства - мікроконтролера Intel 8051, групи команд і особливості їхнього виконання. Виконується домашнє завдання, що складається у формуванні наборів команд, що виконують визначені в завданні операції з заданими типами адресації операндів. Розробляється найпростіша програма поразрядной обробки даних.

Перед початком лабораторної роботи проводиться колоквіум. Студенти, що успішно відповіли на поставлені питання, допускаються до лабораторної частини роботи.

У лабораторії виконуються кілька завдань, що ілюструють особливості системи команд. Правильність вибору команд у домашнім завданні перевіряється шляхом контролю вмісту регістрів і пам'яті мікроконтролера при виконанні кожної команди. Виконується налагодження програми порозрядної обробки.

Потім оформляється звіт із зазначеним нижче змістом.

ЗАВДАННЯ ДЛЯ ДОМАШНЬОЇ ПІДГОТОВКИ

Вивчить систему команд мікроконтролерів сімейства Intel8051

Загальні відомості про систему. Формати команд, типи операндів. Способи адресації: реєстрова, пряма, безпосередня, непряма і неявна. Прапори результату. Символічні імена регістрів спеціальних функцій і портів.

Група команд передачі даних. Типи операндів і структура інформаційних зв'язків. Звертання до акумулятора і зовнішньої пам'яті даних.

Арифметичні команди. Операції додавання, віднімання, множення і ділення, десяткової корекції, інкремента/декремента.

Команди логічних операцій.

Команди операцій з бітами.

Група команд передачі керування. Підпрограми. Робота зі стеком.

Складіть набір з команд

Для виконання лабораторного завдання складіть набір з команд, що виконують задані в табл. 1 операції. Для деяких з цих команд заданий також спосіб адресації хоча б одного з операндів.

Для кожної команди з набору необхідно визначити джерела операндів і приймач результату. Далі варто задати початкові чисельні значення (ненульові) для регістрів і комірок пам'яті, що беруть участь у цій операції. Для деяких команд необхідно визначити і прапори.

За змістом команд визначити чисельні значення результатів операцій.

Операції і способи адресації

№ команди	Операція	Спосіб адресації
1	Пересилання даних	Реєстрова
2	Пересилання даних	Пряма
3	Пересилання даних	Непряма
4	Пересилання даннх	Безпосередня
5	Завантаження в стек	
б	Витяг зі стека	
7	Обмін байтами	Акумулятор <--->регістр
8	Обмін байтами	Акумулятор <---> пам'ять
9	Обмін молодшими тетрадами	Акумулятор <---> резидентна пам'ять даних
10	Додавання	Реєстрова
11	Додавання з переносом	Непряма
12	Віднімання	Пряма
13	Віднімання	Безпосередня
14	Множення	
15	Розподіл	
16	Логічне І	Реєстрова
17	Логічне АБО	Безпосередня
18	Виключаючє АБО	Пряма для першого операнда (приймача)
19	Циклічне зрушення вліво	
20	Зрушення вправо через біт переносС	
21	Безумовний перехід у межах сторінки	

22	Умовний перехід по прапору C	
23	Умовний перехід по довільному біту (прапор користувача)	
24	Порівняння і перехід, якщо “не дорівнює”	Безпосередня для другого операнда
25	Декремент і перехід, якщо “не дорівнює” 0	Реєстрова
26	Передача керування підпрограмі	

Складіть програму порозрядної обробки

Програма повинна установити нульовий розряд числа в регістрі R5 у 1, скинути четвертий розряд у 0 і інвертувати шостий розряд. Використовуйте команди логічних операцій числа з масками. Складіть контрольний приклад для налагодження програми.

Контрольні витання

По яких функціональних групах можна класифікувати команди мікроконтролера?

Який формат може мати команда?

Як тривалість машинного циклу мікроконтролера співвідноситься з його тактовою частотою?

Як визначити час виконання команди?

З якими типами даних може оперувати мікроконтролер?

Для чого використовуються чотирибітні операнди?

Які команди працюють з чотирибітними операнди?

Для чого використовуються двобайтні операнди?

Як побічно адресуються байти пам'яті?

Вкажіть призначення прапорів слова стану програми PSW.

Сформулюйте умови установки прапора OV.

Яке призначення регістрів покажчиків?

Чи може порт одночасно бути джерелом операнда і приймачем результату операції?

Які способи адресаії використовуються в мікроконтролері?

Чи можна адресувати порти і регістри спеціальних функцій побічно?

Приведіть приклади команд передачі даних з різними способами адресації

Розшифруйте команду MOVCA, @A+DPTR.

Приведіть приклади команд доступу до 256 байтів резидентної пам'яті даних до зовнішньої пам'яті даних.

Приведіть приклади логічних і арифметичних команд.

Як виконати вирахування многобайтних операндів?

Перелічить команди операцій з бітами.

Як інвертувати окремі біти портів?

Чи можна адресувати біти побічно?

Які переходи можливі в командах керування?

Для чого використовуються непрямі переходи в програмах?

Поясніть відмінності довгого, абсолютного і відносного переходів у програмах. Як організувати процедуру чекання за допомогою однієї команди?

Які команди використовуються при організації підпрограм?

Які команди модифікують прапори результату?

Укажіть, які з регістрів спеціальних функцій допускають бітову адресацію.

Які прапори використовуються командами умовних переходів? Чим відрізняються команди RET і RETI?

СИСТЕМА КОМАНД МІКРОКОНТРОЛЕРІВ INTEL8051

Загальні відомості

Визначення й асемблерна мнемоніка команд, їхній тип відповідно до рис. 16. Система містить 111 базових команд, що по функціональній ознаці можуть бути розділені на п'ять груп:

- команди передачі даних,
- арифметичні операції,
- логічні операції,
- операції з бітами,
- команди передачі керування.

Більшість команд мають формат в один чи два байти і виконуються за один чи два машинних цикли. Притактовій частоті 12 Мгц тривалість машинного циклу складає 1 мкс. На рис. 16 показані 13 типів команд.

Перший байт команди будь-яких типу і формату завжди містить код операції. Другий і третій байти містять або адреси операндів, або безпосередні операнди

Типиоперандів

Склад операндів містить у собі операнди чотирьох типів: біти, 4-бітні цифри, байти і 16-бітні слова.

Мікроконтролер має 128 програмно-керованих прапорів користувача. Мається також можливість адресації окремих бітів блоку регістрів спеціальних функцій і портів. Для адресації бітів використовується пряма 8-бітна адреса (bit). Непряма адресація бітів неможлива. Карти адрес окремих бітів представлені на мал. 2 і 3.

Чотирибітні операнди використовуються тільки при операціях обміну SWAPі XCHD.

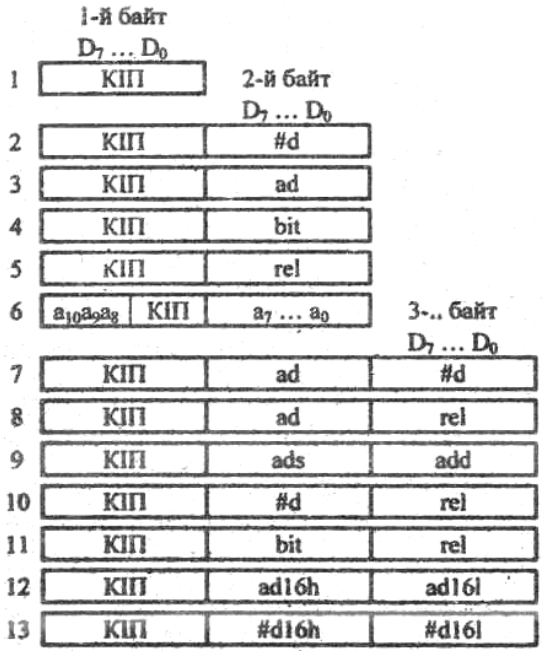


Рис. 16. Типи команд

Адреси	(D_7)							(D_0)
7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08

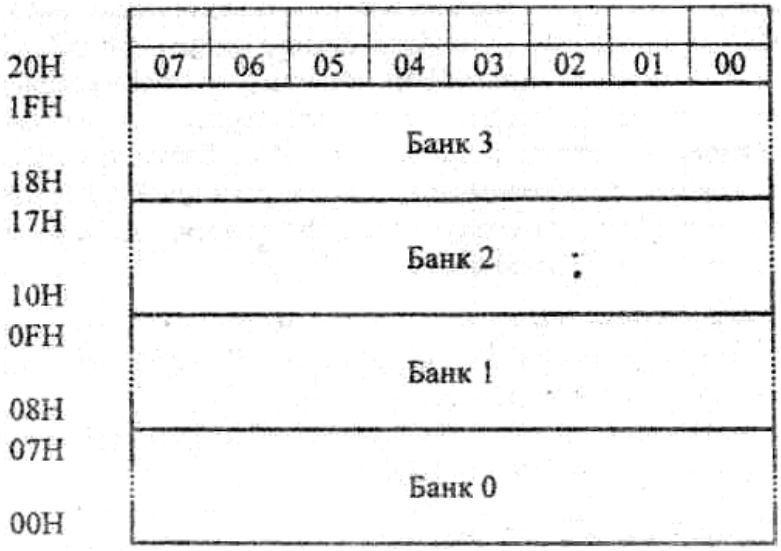


Рис.17. Карта адресуємих бітів в резидентній пам'яті даних

Пряма адреса біта	(D_7)							(D_0)	Ім'я регістра
0FFH									
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	A
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	-	-	-	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	-	-	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0

Рис. 18. Карта адресуємих бітів у блоці регістрів спеціальних функцій

Восьмибітним операндом може бути комірка пам'яті програм (ПП) чи даних (резидентної (РПД) чи зовнішньої (ЗПД)), константа (безпосередній операнд), ре гістри спеціальних функцій, а також порти вводу/виводу. Порти і регістри спеціа льних функцій адресуються тільки прямим способом. Байти пам’яті можуть адре суватися також і непрямим образом через адресні регістри R0, R1, DPTRi PC

Двобайтні операнди - це константи і прямі адреси, для представлення яких використовуються другий і третій байти команди.

Способи адресації даних

У мікроконтролері використовуються наступні способи адресації даних: пряма, безпосередня, непряма і неявна.

При непрямому способі адресації резидентної пам'яті даних використовуються усі вісім бітів адресних регістрів R0 і R1.

Прапори результату

Слово стану програми PSWвключає в себе чотири прапори: З - перенос, АС - допоміжний перенос (напівперенос), OV- переповнення і Р - паритет.

Прапор паритету прямо залежить від поточного значення акумулятора. Якщо число одиничних бітів акумулятора непарне, то прапор Р установлюється, а якщо парне - скидається. Усі спроби змінити прапор Р, привласнюючи йому нове значення, марні, якщо вміст акумулятора при цьому залишиться незмінним.

Прапор АС установлюється, якщо при виконанні операції додавання чи віднімання між тетрадами байта (напівбайтами) виник перенос чи позика.

Прапор С установлюється, якщо в старшому біті результату виникає перенос чи позика. При виконанні операцій множення і ділення прапор Зі скидається.

Прапор OVустановлюється, якщо результат операції додавання віднімання не укладається в сімох бітах і старший (восьмий) біт результату не може інтерпретуватися як знаковий. При виконанні операції розподілу прапор OVскидається, а у випадку ділення на нуль установлюється. При множенні прапор OVуста- новлюється, якщо результат більше 255.

У табл. 2 перелічуються команди, При виконанні яких модифікуються прапори результату. У таблиці відсутній прапор паритету, тому що його значення змінюється всіма командами, що змінюють вміст акумулятора. Крім команд, приведених у таблиці, прапори модифікуються командами, у яких місцем призначення результату визначені PSWчи його окремі біти, а також командами операцій над бітами.

Символічна адресація

При використанні асемблера для одержання об’єктних кодів програм допус- кається застосування в програмах символічних імен регістрів спеціальних функцій, портів і їх окремих битов (мал. 3).

Для адресації окремих бітів і портів (така можливість мається не у всіх регістрів спеціальних функцій) можна використовувати символічне ім'я біта насту пної структури: <ім'я чи регістра порту>.<номер біта>.

Наприклад, символічне ім’я п'ятого біта акумулятора буде наступним: ACC5. Символічні імена є зарезервованими словами, і їхній не треба визначати за допомогою директив асемблера.

Команди, що модифікують прапори результату

Команди	Прапори	Команди	Прапори
ADD	C, OV, AC	CLRC	C = 0
ADDC	C, OV, AC	CPLC	C = NOT(C
SUBB	C, OV, AC	ANL C, b	C
MUL	C = 0, OV	ANLC, /b	C
DIV	C = 0, OV	ORLC, b	C
DA	C	ORL C,/b	C
RRC	C	MOV C, b	C
RLC	C	CJNE	C
SETBC	C = 1		

1.31. Команди передачі даних

Велику частину команд даної групи складають команди передачі й обміну байтів. Команди пересилання битов представлені в групі команд бітових операцій. Усі команди даної групи не модифікують прапори результату, за винятком команд завантаження PSWi акумулятора (прапор паритету).

Структура інформаційних зв 'язків

У залежності від способу адресації і місця розташування операнда можна виділити дев'ять типів операндів, між якими можливий інформаційний обмін. Граф можливих операцій передачі даних показаний на рис. 19. Передачі даних мо жуть виконуватися без участі акумулятора.

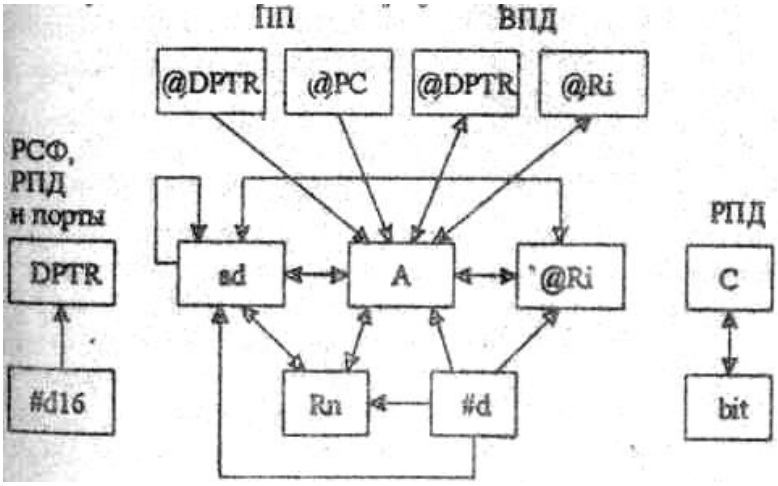


Рис. 19. Граф можливих операцій передачі даних.

Звертання до акумулятора

Звертання до акумулятора може бути виконане з використанням неявної і прямої адресації. У залежності від способу адресації акумулятора застосовується одне із символічних імен: A чи ACC (прямої адреса). При прямої адресації звертання до акумулятора виробляється як до одному з регістрів спеціальних функцій, і його адреса указується в другому байті команди. Використання неявної адреса; акумулятора переважніше, але не завжди можливо, наприклад, при звертанні до окремих биток акумулятора.

Звертання до зовнішньої пам ’яті даних

При використанні команд MOVX @Rізабезпечується доступ до 256 байтів зовнішньої пам'яті даних. Існує також режим звертання до розширеної зовнішньої пам'яті даних, коли для доступу використовується 16-бітна адреса, що зберігається в регістрі-покажчику даних DPTR. Команди MOVX @DPTR забезпечують доступ до 65536 байтів зовнішньої пам'яті даних.

Арифметичні операції

Дану групу утворять 24 команди, що виконують операції додавання, десяткової корекції, інкремента/декремента байтів. Маються команди віднімання, множення і ділення байтів.

Команди ADDi ADDCдопускають додавання акумулятора з великим числом операндів. Аналогічно командам ADDCіснують чотири команди SUBB, що дозволяє досить просто робити віднімання байтів і многобайтних двійкових чисел.

У мікроконтролері реалізується розширений список команд інкремента/декремента байтів, команда інкремента 16-бітного регістра-покажчика даних.

Логічні операції

Дану групу утворять 25 команд, що реалізують функціонально повну систему логічні операції над байтами. У мікроконтролері розширене число типів операндів, що беруть участь в операціях.

Мається можливість робити операцію “ виключаєче АБО” із вмістом портів. Команда XRLможе бути ефективно використана для інверсії окремих бітів портів.

Команди передачі керування

До даної групи команд (табл. П.1.4) відносяться команди, умовного і безумовного розгалуження, виклику підпрограм і повернення з них, а також команда порожньої операції NOP. У більшості команд використовується пряма адресація, тобто адреса переходу цілком (чи його частина) міститься в самій команді передачі керування. Можна виділити три різновиди команд розгалуження по розряд- ності адреси переходу, що вказується.

Довгий перехід

Перехід по всьому адресному просторі пам'яті програм. У команді міститься повна 16-бітна адреса переходу (ad16). Трибайтні команди довгого переходу містять у мнемокоді букву L (Long). Усього існує дві такі команди: LJMP- довгий перехід і LCALL- довгий виклик підпрограми. На практиці рідко виникає необхідність переходу в межах всього адресного простору, і частіше використовуються укорочені команди переходу, що займають менше місця в пам'яті.

Абсолютний перехід

Перехід у межах однієї сторінки пам'яті програм розміром 2048 байтів. Такі команди містять тільки 11 молодших бітів адреси переходу (ad11). Команди абсолютного переходу мають формат 2 байти. Початкова буква мнемокоду - A (Absolute). При виконанні команди в обчисленій адресі наступної один по одному команди ((PC) = (PC) + 2) 11 молодших бітів заміняються на adl 1 з тіла команди абсолютного переходу.

Відносний перехід

Короткий відносний перехід дозволяє передати керування в межах від - 128 до +127 байт щодо адреси наступної команди (команди, що випливає один по одному за командою відносного переходу). Існує одна команда короткого безумовного переходу SJMP (Short). Усі команди умовного переходу використовують даний метод адресації. Відносна адреса переходу (rel) міститься в другому байті команди.

Непрямий перехід

Команда JMP@A + DPTRдозволяє передавати керування по непрямій адресі. Ця команда зручна тим, що надає можливість організації переходу за адресою, що обчислюється самою програмою і невідомому при написанні вихідного тексту програми.

Умовні переходи

Система умовних переходів надає можливість здійснювати розгалуження по наступним умовах: акумулятор містить нуль (JZ), вміст акумулятора не дорівнює нулю (JNZ), перенос дорівнює одиниці (JC), перенос дорівнює нулю (JNC), адресуємий біт дорівнює одиниці (JB), адресуємий біт дорівнює нулю (JNB).

Для організації програмних циклів зручно користатися командою DJNZ. Як лічильник циклів може використовуватися не тільки регістр, але і прямоадресує- мий байт (наприклад, комірка резидентної пам'яті даних).

Команда CJNE використовується в процедурах чекання якої- небудь події. Наприклад, команда

WAIT: CJNEA, P0, WAIT

буде виконуватися доти, поки на лініях порту 0 не установиться інформація, що збігається з вмістом акумулятора.

Усі команди даної групи, за винятком CJNE і JBC, не роблять впливу на прапори.

Команда CJNE встановлює прапор C, якщо перший операнд виявляється менше другого.

Команда JBCскидає прапор C у випадку переходу.

Підпрограми

Для звертання до підпрограм необхідно використовувати команди виклику підпрограм LCALL і ACALL. Ці команди на відміну від команд переходу LJMP іAJMPзберігають у стеку адреси повернення в основну програму. Для повернення з підпрограми необхідно виконати команду RET. Команда RETI відрізняється від команди RET тим, що дозволяє преривання обслугованого рівня.

Операції з бітами

Відмінною рисою даної групи команд є те, що вони оперують з однобітними операндами. У якості таких операндів можуть виступати окремі біти деяких регістрів спеціальних функцій і портів, а також 128 програмних прапорів користувача.

Існують команди скидання (CLR), установки (SETB) і інверсії (CPL) бітів, а також кон'юнкції і диз'юнкції біта і прапора переносу. Для адресації бітів використовується пряма восьмирозрядна адреса (bit). Непряма адресація бітів неможлива.

ЗАВДАННЯ

Виконання прикладів програм

У покроковому режимі роботи ProView виконайте дослідження приведених нижче прикладів програм. Перший приклад містить докладний опис послідовності дій. Інші приклади досліджуються самостійно.

Приклад використання команд передачі даних

Приклад 1. Запис даних. Записати в резиденту пам'ять даних по адресах 41 і 42 число 1C3FH:

```
LOAD: MOVR0,#41H;завантаження в R0 покажчика даних
      MOV @RO,#1CH;запис у пам'ять числа 1CH
      INCR0    ;інкремент покажчика
      MOV @R0,#3FH;запис у пам'ять числа 3FH
```

Доповнимо програму директивами і командами асемблера, що забезпечують налагодження і тестування:

```
START:JMPLOAD;перехід до програми
      ORG 30H       ;директива розміщення програми з адреси 30
LOAD: MOVRO,#41H   ;завантаження в R0 покажчика даних
            MOV  @RO,#1CH;запис  у  пам'ять  числа
1CH
INCR0         ;інкремент покажчика
MOV @R0,#3FH;запис у пам'ять числа 3FH
JMPSTART ;зациклення програми
END ;директива закінчення трансляції
```

Рис.20. Закриття проекту

Переходимо до створення проекту і тестуванню програми в середовищі ProView.

Крок 1. Завантажити програму ProView. Якщо після завантаження програми відкривається попередній проект, з яким вироблялася робота, то необхідно закрити його через меню (рис. 20).

Крок 2. Створити новий проект (рис.21).

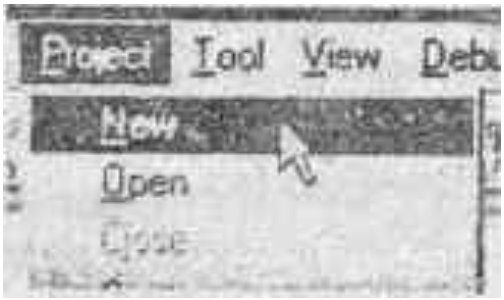


Рис.21. Створення нового проекту

Вкажіть ім'я файлу проекту у вашій особистій папці (рис. 22).

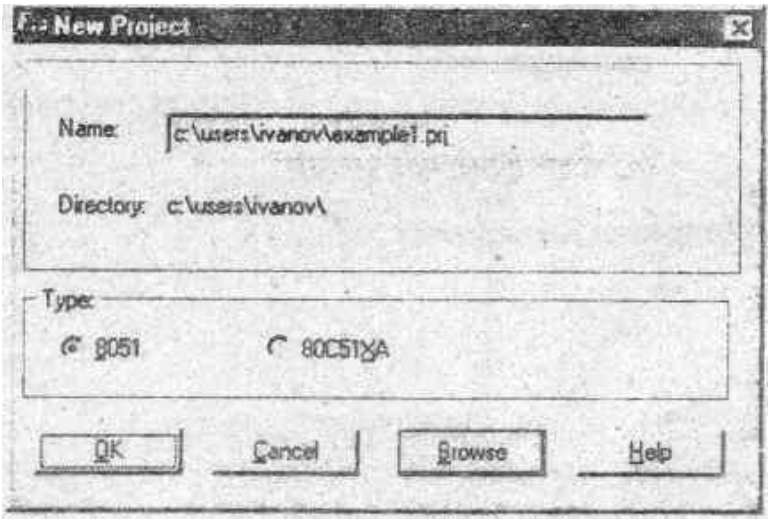


Рис. 22. Ім'я і шлях файлу проекту

Крок 3. Створити новий файл для наступного введення тексту програми (рис. 23) і вказати тип файлу (рис. 24).

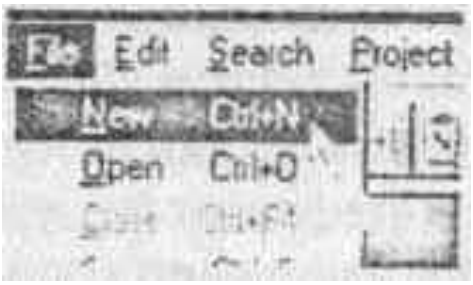


Рис. 23. Новий файл

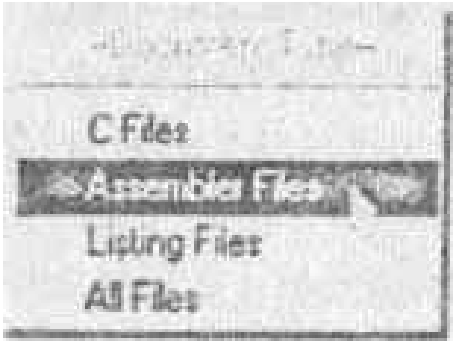


Рис. 24. Тип файлу

Відкриється порожнє вікно для введення асемблерного тексту програми. Записати цей файл (рис. 25).

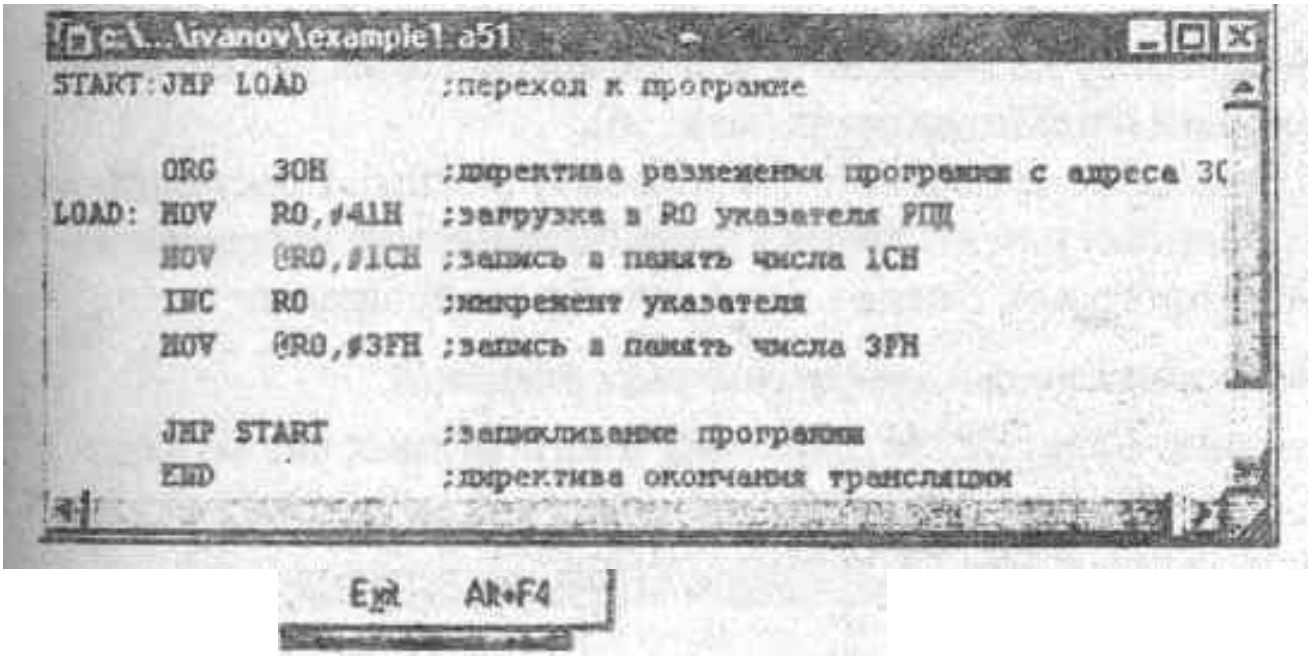


Рис. 25. Запис файлу

Вкажіть ім'я і розширення імені файлу (рис. 26).

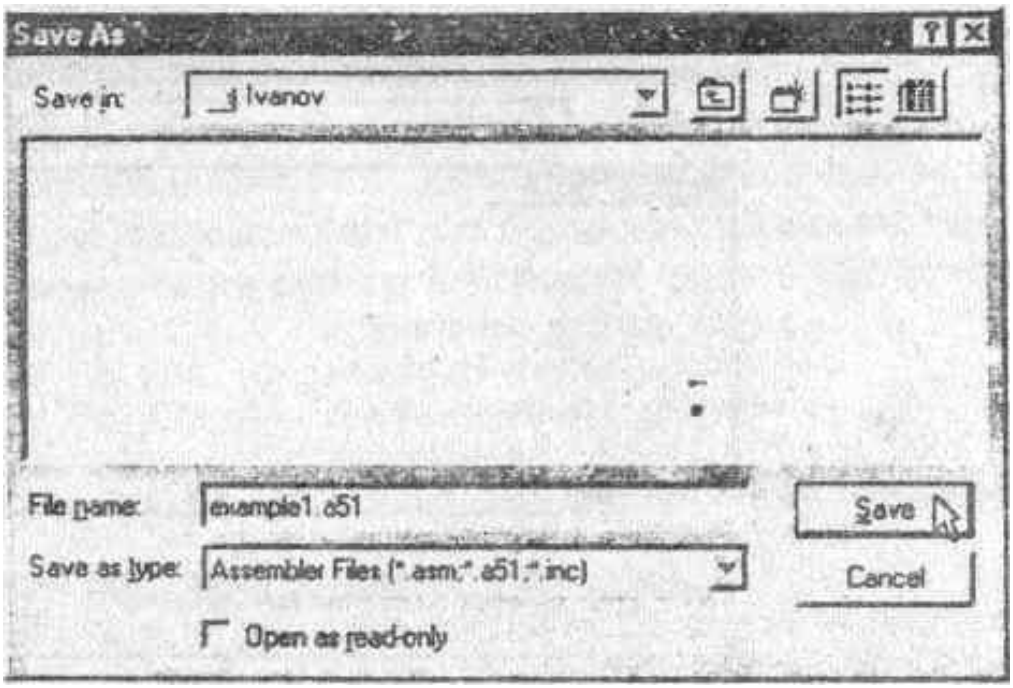


Рис. 26. Ім'я файлу

Крок 4. Ввести текст програми.

Крок 5. Додати цей файл до проекту, попередньо активізувавши вікно проекту (рис. 27).

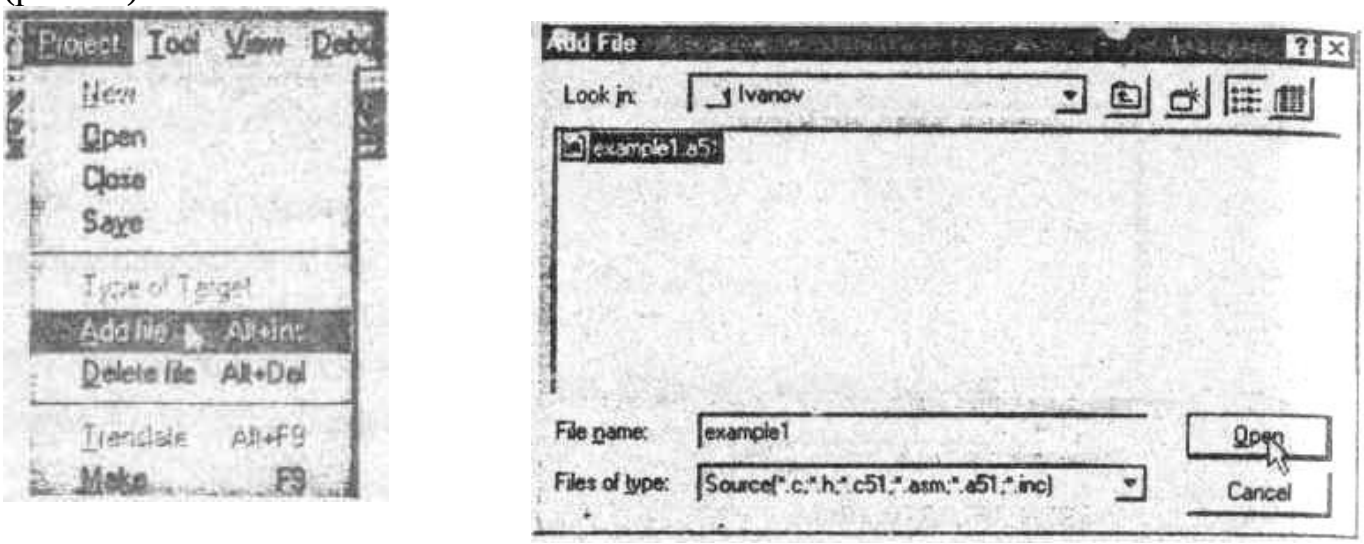


Рис. 27. Додавання файлу до проекту.

Крок 6.Виконати компіляцію проекту.

Крок 7.Запустити програму на виконання за допомогою команди Start з меню Debug, вказавши необхідні опції відладчика.

Подальші кроки. Виконайте програму в покроковому режимі. Спостерігаючи зміну умісту вікна резидентної пам'яті даних і вікна MainRegisters, переконайтесь в правильності роботи програми. Визначте час виконання програми.

Приклади використання команд арифметичних операцій

Приклад 2. Додавання. Скласти два двійкових многобайтних числа. Доданки розташовуються в резидентній пам'яті даних, починаючи з молодшого байта Початкові адреси доданків задані в R0 і R1, формат доданків у байтах - у R2:

```
        CLRC          ;скидання переносу
LOOP:  MOVA, SR0    ;завантаження в А поточного байта першого
                       доданка
        ADDCA, SR1 ;додавання  байтів з врахуванням переносу
        MOV @R0, A  ;розміщення байта результату
        INCR0       ;просування покажчиків INCR1
        DJNZR2, LOOP;цикл, якщо не всі байти просумували
```

При додаванні чисел без знаку на переповнення вкаже прапор С, а у випадку додавання чисел зі знаком - прапор OV.

Доповніть програму додавання командами, що забезпечують її тестування. складіть контрольний приклад і виконайте налагодження в ProView. Визначте час обчислення в залежності від формату вихідних чисел.

Приклад 3. Множення. Команда MULобчислює добуток двох цілих безнакових чисел, що зберігаються в регістрах А і В. Молодша частина добутку розміщається в А, а старша - у регістрі-розширнику В. Якщо вміст В виявляється рів ним нулю, то прапор OV скидається, інакше - установлюється. Прапор переносу завжди скидається. Наприклад, якщо акумулятор містив число 200 (0C8H), а розширник 160 (0A0H), те в результаті виконання команди MULAB вийде добуток 32000 (7D00H). Акумулятор

буде містити нуль, а розширник - 7DH, прапор OV буде встановлений, а прапор C - скинутий.

Нехай потрібно помножити ціле двійкове число довільного формату на константу 73. Вихідне число розміщується в резидентній пам'яті даних, адреса молодшого байта знаходиться в регістрі R0. Формат числа в байтах зберігається в R1:

```
        MOVA,#0             ;скидання акумулятора
LOOP:   ADDA, @R0       ;завантаження множеного
        MOV B,#73           ;завантаження множника
        MULAB               ;множення
        MOV @R0,A          ;запис молодшого байта   часткового
                            ;добутку
        INC R0              ;збільшення адреси
        MOVA,B             ;пересилання старшого байта    часткового
                            ;добутку в акумулятор
        XCH A,@R0           ;попереднє формування
                            ;чергового байта добутку
        DJNZR1,LOOP;цикл, якщо не всі байти вихідного
                            ;числа помножені на константу
```

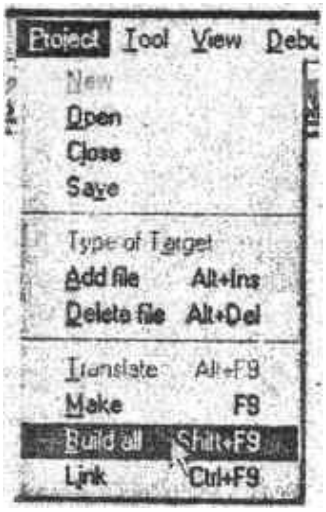


Рис. 28. Компіляція

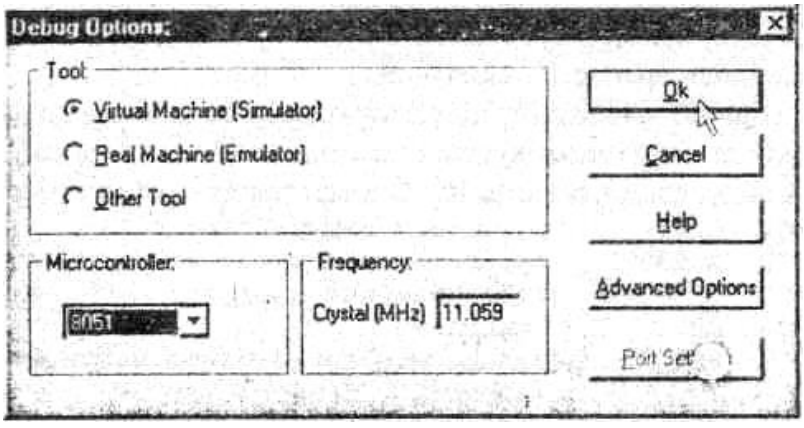
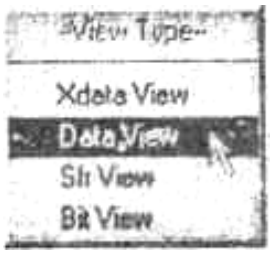


Рис. 29. Вікно опцій відладчика



Крок 8. За допомогою команди Datadump з меню View відкрити вікно резидентної пам'яті даних (рис. 30).

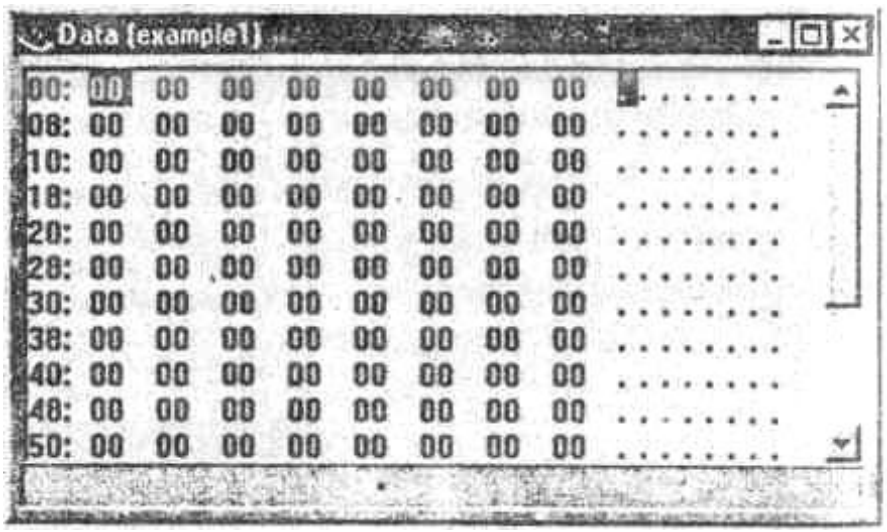


Рис. 30. Вікно резидентної пам'яті даних

Отриманий добуток розміщується на місці вихідного числа і займає в пам'яті на один байт більше.

Розберіться в алгоритмі множення. Доповніть програму командами, що забезпечують її тестування, складіть контрольний приклад і виконайте налагодження в ProView. Визначте час обчислення в залежності від формату вихідного числа.

Приклад 4. Розподіл. Команда DIV робить розподіл вмісту акумулятора на вміст регістра-розширника. Після розподілу акумулятор містить цілу частину частки, а розширник - залишок. Прапори C і OV скидаються. При розподілі на нуль установлюється прапор переповнення, а частка залишається невизначеним. Команда розподілу може бути використана для швидкого перетворення двійкових чисел у десяткові двійково-кодовані (BCD-числа).

Як приклад розглянемо програму, що переводить двійкове число, що міститься в акумуляторі, у BCD-код. При такому перетворенні може вийти трьохрозрядне BCD - число. Старша цифра (число сотень) буде розміщена в регістрі R0, а дві молодші в акумуляторі:

```
MOVB,#100     ;завантаження 100 для обчислення кількості сотень
DIVAB         ;акумулятор містить число сотень (старшу цифру)
MOVR0,A  ;пересилання в R0 старшої цифри
XCH  A,B      ;пересилання залишку вихідного числа в акумулятор
MOVB,#10 ;завантаження 10 для обчислення кількості десятків
DIVAB         ;A містить число десятків, У - число одиниць
SWAPA         ;розміщення числа десятків у старшій тетраді A
ADDA,B        ;підсумування залишку (числа одиниць),
               ;тепер акумулятор містить дві молодші цифри
```

Доповніть програму командами, що забезпечують її тестування, складіть контрольний приклад і виконайте налагодження в ProView. Визначте час обчислення.

Приклад використання команд логічних операцій

Приклад 5. Логічні операції. Виконайте налагодження програми порозряд ної обробки, що була підготовлена при виконанні домашнього завдання. Переконайтеся в її працездатності на контрольних прикладах. Визначте час обчислення.

Приклад використання команд передачі керування і роботи зі стеком

Приклад 6. Операції зі стеком. Перед завантаженням у стек вміст регістра-покажчика стека SP інкрементується, а після витягу зі стека декрементується.

По сигналі системного скидання в SPзаноситься початкове значення 07H. Для перевизначення SPможна скористатися командою MOVSP, #d.

Таким чином, стек може розташовуватися в будь-якім місці резидентної пам'яті даних. Стік використовується для організації звертань до підпрограм і при обробці преривань, може бути використаний для передачі параметрів підпрограмам і для тимчасового збереження вмісту регістрів спеціальних функцій.

Підпрограма повинна зберегти в стеці вміст тих регістрів, що вона сама буде використовувати, а перед поверненням у перервану програму повинна відновити їхнього значення.

Підпрограма з доповненнями для її тестування може, наприклад, мати наступну структуру:

```
START:   MOV R1,#02H ;завантаження регістрів
         MOVA,#30H
         MOVR2,#00H
         LCALLSUB ;перехід на підпрограму
         SUMPSTART
SUB:     PUSHPSW  ;збереження в стеці PSW
         PUSHACC  ;збереження акумулятора
         PUSHB        ;збереження розширника акумулятора B
         ADDA,R1  ;власне обробка даних
         MOVR2,A
         POPB;відновлення B
         POPACC   ;відновлення акумулятора
         POPPSW   ;відновлення PSW
              RET     ;повернення
         END
```

Якщо припустити, що SP перед виникненням преривання містив значення 1FH, то розміщення регістрів у стеці після входу в підпрограмму обробки буде таким, як на рис. 19.

Дослідіть процес виконання команд виклику і повернення з підпрограми, а також команд роботи зі стеком. Для цього запустіть програму в покроковому режимі.

Замініть команду POP PSW на NOP і простежте, як буде виконуватися програма. Поясніть зміни, що відбулись, сформулюйте висновки для звіту.

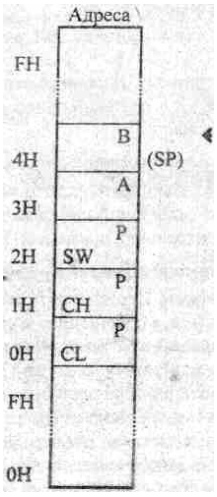


Рис. 31. Вміст стека після виконання команди CALLі серії команд PUSH

Запустіть на виконання команди з набору, сформованого при виконанні домашнього завдання. Зробити це можна різними способами, наприклад:

```
    JMPM1
    ORG 30H
M1: <досліджувана команда>
 END
```

Команда з набору записується в другому рядку програми за міткою M1 і буде розміщена за адресою 30H у резидентній пам’яті програм. Далі здійснюється асемблерування і завантаження програми. Перед пуском програми необхідно за допомогою миші і клавіатури задати початкові значення (операнди) для тих регістрів і комірок пам'яті, що беруть участь у виконанні відповідної команди. Потім здійснюється пуск програми, після чого необхідно зафіксувати результати операції.

Далі описаний процес повторюється для наступної команди і т.д. Для команд, що модифікують прапори слова стану програми PSW, необхідно також зафіксувати зміни стану бітів C, OV, AC.

Результати виконання операцій заносяться до таблиці.

Таблиця

Результати виконання команд операцій

№	Команда	Код	Виконувана операція	Зміст регістрів і пам'яті до і після виконання		Пояснення
				До	Після	
1	MOV A,R0	E8	Пересилання байта даних із регістра R0 в акумулятор A	A/00 R0/F2	A/F2 R0/F2	
2	...	...	...	...	...	...
...	...	...	...	...	...	...
26	...	...	...	...	...	...

Проаналізувати відповідність результатів визначенню виконуваних операцій. Виділити команди, модифікуємі флаги слова стану програми PSW, і пояснити стан флагів після виконання цих команд.

Система команд Intel 8051

Мнемокод	КОП	Мнемокод	КОП	Мнемокод	КОП
ACALL OxxH	11	ANL A, R4	5C	DJNZ R1, rel	D9
ACALL 1xxH	31	ANL A, R5	5D	DJNZ R2, rel	DA
ACALL 2xxH	51	ANL A, R6	5E	DJNZ R3, rel	DB
ACALL 3xxH	71	ANL A, R7	5F	DJNZ R4, rel	DC
ACALL 4xxH	91	ANL A, @R0	56	DJNZ R5, rel	DD
ACALL 5xxH	B1	ANL A, @R1	57	DJNZ R6, rel	DE
ACALL бxxH	D1	ANL A, #d	54	DJNZ R7, rel	DF
ACALL 7xxH	F1	ANL ad, A	52	INC A	04
ADD A, ad	25	ANL ad, #d	53	INC ad	05
ADD A, R0	28	ANL C, bit	82	INC DPTR	A3
ADD A, R1	29	ANL C,/bit	B0	INC R0	08
ADD A, R2	2A	CJNE A, ad, rel	B5	INC R1	09
ADD A, R3	2B	CJNE A, #d, rel	B4	INC R2	0A
ADD A, R4	2C	CJNE R0, #d, rel	B8	INC R3	0B
ADD A, R5	2D	CJNE R1, #d, rel	B9	INC R4	0C
ADD A. R6	2E	CJNE R2, #d, rel	BA	INC R5	0D
ADD A, R7	2F	CJNE R3, #d, rel	BB	INC R6	0E
ADD A, @R0	26	CJNE R4, #d, rel	BC	INC R7	0F
ADD A, @R1	27	CJNE R5, #d, rel	BD	INC @R0	06
ADD A, #d	24	CJNE R6, #d, rel	BE	INC @R1	07
ADDC A, ad	35	CJNE R7, #d, rel	BF	JB bit, rel	20
ADDC A, RO	38	CJNE @R0, #d, rel	B6	JBC bit, rel	10
ADDC A, R1	39	CJNE @R1,#d, rel	B7	JC rel	40
ADDC A, R2	3A	CLRA	E4	JMP @A + DPTR	73
ADDC A, R3	3B	CLRbit	C2	JNB bit, rel	30
ADDC A, R4	3C	CLRC	C3	JNC rel	50
ADDC A, R5	3D	CPLA	F4	JNZ rel	70
ADDC A, R6	3E	CPLbit	B2	JZ rel	60
ADDC A, R7	3F	CPL C	B3	LCALL adl6	12
ADDC A, @R0	36	DA A	D4	LJMP ad 16	02
ADDC A, @R1	37	DEC A	14	MOV A, ad	E5
ADDC A, #d	34	DEC ad	15	MOV A, R0	E8
AJMP 0XXH	01	DEC R0	18	MOV A, R1	E9
AJMP 1XXH	21	DEC R1	19	MOV A, R2	EA
AJMP2XXH	41	DEC R2	1A	MOV A, R3	EB
AJMP3XXH	61	DEC R3	1B	MOV A, R4	EC
AJMP4XXH	81	DEC R4	1C	MOV A, R5	ED
AJMP5XXH	A1	DEC R5	1D	MOV A, R6	EE
AJMP 6XXH	C1	DEC R6	1E	MOV A, R7	EF
AJMP7XXH	E1	DEC R7	1F	MOV A, @R0	E6
ANL A, ad	55	DEC @R0	16	MOV A, @R1	E7
ANL A, R0	58	DEC @R1	17	MOV A, #d	74
ANLA,R1	59	DIV AB	84	MOV ad, A	F5
ANLA.R2	5A	DJNZ ad, rel	D5	MOV ad, R0	88
ANLA.R3	5B	DJNZ R0, rel	D8	MOV ad, R1	89

Система команд Intel 8051

Мнемокод	КОП	Мнемокод	КОП	Мнемокод	КОП
ACALL OxxH	11	ANL A, R4	5C	DJNZ R1, rel	D9
ACALL 1xxH	31	ANL A, R5	5D	DJNZ R2, rel	DA
ACALL 2xxH	51	ANL A, R6	5E	DJNZ R3, rel	DB
ACALL 3xxH	71	ANL A, R7	5F	DJNZ R4, rel	DC
ACALL 4xxH	91	ANL A, @R0	56	DJNZ R5, rel	DD
ACALL 5xxH	B1	ANL A, @R1	57	DJNZ R6, rel	DE
ACALL 6xxH	D1	ANL A, #d	54	DJNZ R7, rel	DF
ACALL 7xxH	F1	ANL ad, A	52	INC A	04
ADD A, ad	25	ANL ad, #d	53	INC ad	05
ADD A, R0	28	ANL C, bit	82	INC DPTR	A3
ADD A, R1	29	ANL C,/bit	B0	INC R0	08
ADD A, R2	2A	CJNE A, ad, rel	B5	INC R1	09
ADD A, R3	2B	CJNE A, #d, rel	B4	INC R2	0A
ADD A, R4	2C	CJNE R0, #d, rel	B8	INC R3	0B
ADD A, R5	2D	CJNE R1, #d, rel	B9	INC R4	0C
ADD A. R6	2E	CJNE R2, #d, rel	BA	INC R5	0D
ADD A, R7	2F	CJNE R3, #d, rel	BB	INC R6	0E
ADD A, @R0	26	CJNE R4, #d, rel	BC	INC R7	0F
ADD A, @R1	27	CJNE R5, #d, rel	BD	INC @R0	06
ADD A, #d	24	CJNE R6, #d, rel	BE	INC @R1	07
ADDC A, ad	35	CJNE R7, #d, rel	BF	JB bit, rel	20
ADDC A, RO	38	CJNE @R0, #d, rel	B6	JBC bit, rel	10
ADDC A, R1	39	CJNE @R1,#d, rel	B7	JC rel	40
ADDC A, R2	3A	CLRA	E4	JMP @A + DPTR	73
ADDC A, R3	3B	CLRbit	C2	JNB bit, rel	30
ADDC A, R4	3C	CLRC	C3	JNC rel	50
ADDC A, R5	3D	CPLA	F4	JNZ rel	70
ADDC A, R6	3E	CPLbit	B2	JZ rel	60
ADDC A, R7	3F	CPL C	B3	LCALL adl6	12
ADDC A, @R0	36	DA A	D4	LJMP ad 16	02
ADDC A, @R1	37	DEC A	14	MOV A, ad	E5
ADDC A, #d	34	DEC ad	15	MOV A, R0	E8
AJMP 0XXH	01	DEC R0	18	MOV A, R1	E9
AJMP 1XXH	21	DEC R1	19	MOV A, R2	EA
AJMP2XXH	41	DEC R2	1A	MOV A, R3	EB
AJMP3XXH	61	DEC R3	1B	MOV A, R4	EC
AJMP4XXH	81	DEC R4	1C	MOV A, R5	ED
AJMP5XXH	A1	DEC R5	1D	MOV A, R6	EE
AJMP 6XXH	C1	DEC R6	1E	MOV A, R7	EF
AJMP7XXH	E1	DEC R7	1F	MOV A, @R0	E6
ANL A, ad	55	DEC @R0	16	MOV A, @R1	E7
ANL A, R0	58	DEC @R1	17	MOV A, #d	74
ANLA,R1	59	DIV AB	84	MOV ad, A	F5
ANLA.R2	5A	DJNZ ad, rel	D5	MOV ad, R0	88
ANLA.R3	5B	DJNZ R0, rel	D8	MOV ad, R1	89

Шістнадцяткові коди команд Intel 8051

Старша цифра	Молодша							
	0	1	2	3	4	5	6	7
0	NOP	AJMP 0XXH	LJMP ad16	RR A	INC A	INC ad	INC @R0	INC @R1
1	JBC bit, rel	ACALL 0XXH	LCALL ad16	RRC A	DEC A	DEC ad	DEC @R0	DEC @R1
2	JB bit, rel	AJMP 1XXH	RET	RL A	ADD A, #d	ADD A, ad	ADD A, @R0	ADD A, @R1
3	JNB bit, rel	ACALL 1XXH	RETI	RLC A	ADDC A, #d	ADDC A, ad	ADDC A, @R0	ADDC A, @R1
4	JC rel	AJMP 2XXH	ORL ad, A	ORL ad, #d	ORL A, #d	ORL A,ad	ORL A, @R0	ORL A, @R1
5	JNC rel	ACALL 2XXH	ANL ad, A	ANL ad, #d	ANL A, #d	ANL A, ad	ANL A, @R0	ANL A, @R1
6	JZ rel	AJMP 3XXH	XRL ad, A	XRL ad, #d	XRL A, #d	XRL A, ad	XRL A, @R0	XRL A, @R1
7	JNZ rel	ACALL 3XXH	ORL C, bit	JMP @A+DPTR	MOV A, #d	MOV ad, #d	MOV @R0, #d	MOV @R1, #d
8	SJMP rel	AJMP 4XXH	ANL C, bit	MOVC A, @A+PC	DIV AB	MOV add, ads	MOV ad, @R0	MOV ad, @R1
9	MOV DPTR, #d16	ACALL 4XXH	MOV bit, C	MOVC A,@A+DPTR	SUBB A, #d	SUBB A, ad	SUBB A, @R0	SUBB A, @R1
A	ORL C, /bit	AJMP 5XXH	MOV C, bit	INC DPTR	MUL AB	[illegible]	MOV @R0, ad	MOV @R1, ad
B	ANL C, /bit	ACALL 5XXH	CPL bit	CPL C	CJNE A, #d, rel	CJNE A, ad, rel	CJNE @R0,#d, rel	CJNE @R1,#d, rel
C	PUSH ad	AJMP 6XXH	CLR bit	CLR C	SWAP A	XCH A, ad	XCH A, @R0	XCH A, @R1
D	POP ad	ACALL 6XXH	SETB bit	SETB C	DA A	DJNZ ad, rel	XCHD A, @R0	XCHD A, @R1
E	MOVX A, @DPTR	AJMP 7XXH	MOVX A, @R0	MOVX A, @R1	CLR A	MOV A, ad	MOV A, @R0	MOV A, @R1
F	MOVX @DPTR, A	ACALL 7XXH	MOVX @R0, A	MOVX @R1, A	CPL A	MOV ad, A	MOV @R0, A	MOV @R1, A

X – шістнадцатирічна цифра.

цифра								Старша цифра
8	9	A	B	C	D	E	F	
INC R0	INC R1	INC R2	INC R3	INC R4	INC R5	INC R6	INC R7	0
DEC R0	DEC R1	DEC R2	DEC R3	DEC R4	DEC R5	DEC R6	DEC R7	1
ADD A, R0	ADD A, R1	ADD A, R2	ADD A, R3	ADD A, R4	ADD A, R5	ADD A, R6	ADD A, R7	2
ADDC A, R0	ADDC A, R1	ADDC A, R2	ADDC A, R3	ADDC A, R4	ADDC A, R5	ADDC A, R6	ADDC A, R7	3
ORL A, R0	ORL A, R1	ORL A, R2	ORL A, R3	ORL A, R4	ORL A, R5	ORL A, R6	ORL A, R7	4
ANL A, R0	ANL A, R1	ANL A, R2	ANL A, R3	ANL A, R4	ANL A, R5	ANL A, R6	ANL A, R7	5
XRL A, R0	XRL A, R1	XRL A, R2	XRL A, R3	XRL A, R4	XRL A, R5	XRL A, R6	XRL A, R7	6
MOV R0, #d	MOV R1, #d	MOV R2, #d	MOV R3, #d	MOV R4, #d	MOV R5, #d	MOV R6, #d	MOV R7, #d	7
MOV ad, R0	MOV ad, R1	MOV ad, R2	MOV ad, R3	MOV ad, R4	MOV ad, R5	MOV ad, R6	MOV ad, R7	8
SUBB A, R0	SUBB A, R1	SUBB A, R2	SUBB A, R3	SUBB A, R4	SUBB A, R5	SUBB A, R6	SUBB A, R7	9
MOV R0, ad	MOV R1, ad	MOV R2, ad	MOV R3, ad	MOV R4, ad	MOV R5, ad	MOV R6, ad	MOV R7, ad	A
CJNE R0, #d, rel	CJNE R1, #d, rel	CJNE R2, #d, rel	CJNE R3, #d, rel	CJNE R4, #d, rel	CJNE R5, #d, rel	CJNE R6, #d, rel	CJNE R7, #d, rel	B
XCH A, R0	XCH A, R1	XCH A, R2	XCH A, R3	XCH A, R4	XCH A, R5	XCH A, R6	XCH A, R7	C
DJNZ R0, rel	DJNZ R1, rel	DJNZ R2, rel	DJNZ R3, rel	DJNZ R4, rel	DJNZ R5, rel	DJNZ R6, rel	DJNZ R7, rel	D
MOV A, R0	MOV A, R1	MOV A, R2	MOV A, R3	MOV A, R4	MOV A, R5	MOV A, R6	MOV A, R7	E
MOV R0, A	MOV R1, A	MOV R2, A	MOV R3, A	MOV R4, A	MOV R5, A	MOV R6, A	MOV R7, A	F

Практична робота №3
ПРОГРАМУВАННЯ МОВОЮ АСЕМБЛЕРА МІКРОКОНТРОЛЕРА INTEL 8051

МЕТА РОБОТИ

Метою роботи є вивчення основ мови асемблера мікроконтролерів сімейства Intel 8051, а також продовження початого в лабораторних роботах №1 і №2 вивчення інтегрованого середовища ProView фірми Franklin Software Inc., що призначене для розробки програмного забезпечення цього сімейства.

При домашній підготовці до роботи вивчаються основні правила програмування мовою асемблера і найбільш вживані директиви. Далі відповідно до номера варіанта студенти складають програми мовою асемблера.

Перед початком лабораторної роботи проводиться колоквіум. Студенти, що успішно відповіли на поставлені питання, допускаються до лабораторної частини роботи.

При виконанні завдання здійснюється введення вихідних текстів програм, асемблірування і налагодження програм.

Після виконання роботи оформляється звіт із зазначеним нижче змістом.

ЗАВДАННЯ ДЛЯ ПІДГОТОВКИ

1. Вивчить методику розробки прикладного програмного забезпечення мікроконтролерних систем

Формалізований підхід до розробки прикладних програм. Етапи формалізації в розробці алгоритмів. Модульний принцип побудови прикладних програм, процедури і підпрограми. Виклик підпрограм, збереження параметрів основної програми, передача параметрів у підпрограми. Методика налагодження прикладних програм.

Правила запису програм мовою асемблера. Поля мітки, операції, операндів і коментарю. Обробка виражень в процесі трансляції. Директиви асемблера: BIT, DATA, DB, DS, DW, END, EQU, ORG, RSEG, SEGMENT, SET, XDATA.

2. Складіть программу. У тексті підпрограми необхідно забезпечити збереження вмісту регістрів мікроконтролера, тому що регістри можуть використовуватися основною програмою. Розгляньте різні способи передачі параметрів у підпрограму. У максимально можливому ступені використовуйте директиви асемблера.

Складіть програму, що забезпечує тестування підпрограми.

Вихідний текст програми повинний містити не тільки оператори машинних команд, що реалізують алгоритм запропонованої задачі, але і директиви завдання початкових значень пам'яті і регістрів. У максимально можливому ступені використовуйте директиви асемблера.

Результати роботи програми повинні фіксуватися в пам'яті.

Контрольні питання

1. Перелічить й охарактеризуйте етапи розробки прикладної програми.
2. Сформулюйте поняття функціональної специфікації прикладної програми.
3. Сформулюйте поняття модульної декомпозиції задачі прикладної програми.
4. Вкажіть достоїнства і недоліки мови асемблера.
5. Яка структура рядка програми, написаної мовою асемблера, які поля рядка є обов'язковими?
6. Вкажіть правила вибору імені мітки.
7. Яким образом виділяється коментар?
8. Перелічить основні директиви мови асемблера.
9. Які директиви використовуються для вказівки початку і кінця програмного модуля?
10. Перелічить директиви, які використовуються для резервування пам'яті.
11. Які директиви використовуються для визначення програмних сегментів?
12. Вкажіть функції наступних директив: ORG, EQU, SET, BIT. У чому полягає відмінність директив EQU і SET?

13. Вкажіть призначення директив DATA, XDATA.
14. Вкажіть призначення директив DB, DS, DW, END.
15. Вкажіть призначення директив RSEG, SEGMENT.
16. Яким образом визначається макрокоманда?
17. Вкажіть достоїнства і недоліки підпрограм у порівнянні з макрокомандами. У яких випадках доцільно використовувати підпрограми, а в яких - макрокоманди?
18. Які команди можна використовувати для створення циклічної програми?
19. Якими обов'язковими властивостями повинна володіти підпрограма?
20. Яким образом використовується стек при виконанні підпрограм?
21. Від чого залежить глибина вкладеності підпрограм?
22. Для чого може бути використаний прийом переключення реєстрових банків?
23. Поясніть механізм передачі параметрів підпрограми через пам'ять.
24. Поясніть механізм передачі параметрів підпрограми через регістри загального призначення.
25. Поясніть механізм передачі параметрів підпрограми через регістр ознак.
26. Поясніть механізм передачі параметрів підпрограми через стек.
27. Вкажіть основні етапи роботи з асемблерною програмою.
28. Як розрахувати час виконання асемблерної програми?
29. Вкажіть основні прийоми, які використовуються при налагодженні програми.

МЕТОДИКА РОЗРОБКИ ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ

1. Загальні відомості

Якщо задача на розробку прикладної програми для мікроконтролера поставлена, то для одержання тексту вихідної програми необхідно виконати ряд послідовних дій.

30. Докладно описати задачу.
31. Проаналізувати задачу.
32. Виконати інженерну інтерпретацію задачі, бажано з залученням того чи іншого апарата формалізації (граф автомата, мережі Петри, матриці станів і зв’язковості і т.п.).
33. Розробити загальну схему алгоритму роботи контролера.
34. Розробити деталізовані схеми окремих процедур, виділених на основі модульного принципу складання програм.
35. Детально проробити інтерфейс контролера і внести виправлення в загальну і деталізовану схеми алгоритмів.
36. Розподілити робочі регістри і пам'ять.
37. Сформувати текст вихідної програми.

У результаті роботи з трьох перших пунктів даного переліку одержують так звану функціональну специфікацію прикладної програми,

у якій основна увага приділяється деталізації способів формування вхідної і вихідної інформації.

Мовою схем алгоритмів розроблювач описує метод, обраний їм для рішення поставленої задачі. Досить часто буває, що та сама задача може бути вирішена різними методами. Спосіб рішення задачі, обраний на етапі її інженерної інтерпретації, на основі якого формується схема, визначає не тільки якість розроблювальної прикладної програми, але і якісні показники кінцевого виробу.

Алгоритм є точно визначена процедура, що наказує контролеру однозначно визначені дії по перетворенню вихідних даних в оброблені вихідні дані. Тому розробка схеми вимагає граничної точності й однозначності використовуваної атрибутики: символічних імен змінних, констант, підпрограм (модулів), символічних адрес таблиць, портів вводу-виводу і т.п. Основну увагу при розробці варто приділити тому розділу функціональної специфікації прикладної програми, у якому приводиться опис апаратури сполучення з об'єктом керування. Цей опис повинний бути деталізований аж до електричних і тимчасових характеристик кожного вхідного і вихідного чи сигналу пристрою.

Успіх розробки прикладної програми полягає у використанні методу декомпозиції, при якому вся задача послідовно розділяється на менші функціональні модулі. Кожний з модулів можна аналізувати, розробляти і налагоджувати окремо від інших. При виконанні прикладної програми в мікроконтролері керування без усяких двозначностей передається від одного функціонального модуля до іншого. Схема зв’язковості цих функціональних модулів, кожний з який реалізує деяку процедуру, утворює загальну схему алгоритму прикладної програми. Мову графічних образів схеми алгоритму можна використовувати на будь-якому рівні деталізації опису модулів аж до того, що кожному оператору схеми буде відповідати єдина команда мікроконтролера.

Структурне програмування є процес побудови прикладної програми з набору програмних модулів, кожний з яких реалізує визначену процедуру обробки даних. Програмні модулі повинні мати тільки одну точку входу й одну точку виходу. Тільки в цьому випадку окремі модулі можна розробляти і налагоджувати незалежно, а потім поєднувати в закінчену прикладну програму з мінімальними проблемами їхнього взаємозв'язку.

Джерелом переважної більшості помилок програмування є використання модулів, що мають один вхід і кілька виходів. При необхідності організації множинних розгалужень у програмі декомпозицію задачі виконують таким чином, щоб кожен функціональний модуль мав тільки один вхід і один вихід. Для цього умовні оператори (що мають два виходи) включають усередину модуля, поєднуючи їх з операторами обробки, чи виносять у систему міжмодульних зв'язків, формуючи тим самим схему алгоритму більш високого рангу.

У міжнародному стандарті на програмний продукт HIPO (Hierarchy-Input-Process-Output) («хай-по») декларується аналогічний підхід до розробки прикладних програм.

Розробка схеми алгоритму функціонального модуля програми має яскраво виражений ітеративний характер, тобто вимагає багаторазових проб, перш ніж виникає впевненість, що алгоритм реалізації процедури правильний і завершений. Поза залежністю від функціонального призначення процедури при розробці її схеми необхідно дотримуватись наступної черговості роботи.

38. Визначити, що повинен робити модуль.
39. Визначити способи одержання модулем вихідних даних (від датчиків через порти вводу, з таблиць у пам'яті чи через робочі регістри).
40. Визначити необхідність якої-небудь попередньої обробки введених вихідних даних (маскування, зрушення, масштабування, перекодування).
41. Визначити метод перетворення вхідних даних у необхідні вихідні. Використовуючи оператори процедур і умовних операторів ухвалення рішення, відобразити мовою схеми алгоритму обраний метод.
42. Визначити способи видачі з модуля оброблених даних (передати в пам'ять, у програму, що викликала, чи в порти виводу).
43. Визначити необхідність будь-якої вторинної обробки виведених даних (зміна формату, перекодування, масштабування, маскування).
44. Повернутися до пункту 1 дійсного переліку і проаналізувати отриманий результат. Виконати ітеративне коректування схеми алгоритму з метою зробити її простою, логічною, стрункою й маючою чіткий графічний образ.
45. Перевірити працездатність алгоритму на папері шляхом підстановки в нього дійсних даних. Переконатися в його збіжності і результативності.
46. Розглянути граничні випадки і спробувати визначити граничні значення інформаційних об'єктів, з якими оперує алгоритм, за межами яких він втрачає властивості кінцевості, збіжності чи результативності. Особливу увагу при цьому варто приділити аналізу можливих ситуацій переповнення розрядної сітки, зміни знака результату операції, розподілу на змінну, котра може прийняти нульове значення.
47. Провести уявний експеримент по визначенню працездатності алгоритму в реальному масштабі часу, коли стохастичні події, що відбуваються в об'єкті керування, можуть уплинути на роботу алгоритму. При цьому самому ретельному аналізу варто піддати реакцію алгоритму на можливі переривання з метою визначення критичних операторів, які необхідно захистити від переривань. Крім того, у ході цього уявного експерименту варто проаналізувати логіку алгоритму з метою визначення таких послідовностей операторів, при виконанні яких мікроконтролер може «не помітити» короткочасних

подій в об'єкті керування. При виявленні таких ситуацій у логіку варто внести корективи.

Практика розробки програмного забезпечення показала, що послідовне використання описаної поетапної процедури, що складає основу методу структурного програмування, дозволяє впевнено одержувати працездатні прикладні програми.

Перетворення розробленої схеми алгоритму у вихідний текст програми справа нескладна. Але перш ніж приступити до написання програми, необхідно специфікувати пам'ять і вибрати мову програмування.

Специфікація пам'яті і робочих регістрів полягає у визначенні адреси першої команди прикладної програми, дійсних початкових адрес стека, таблиць даних, буферних зон передачі параметрів між процедурами, підпрограм обслуговування переривань і т.д. При цьому варто пам'ятати, що в мікроконтролерах пам'ять програм і пам'ять даних фізично і логічно розділені.

Діапазон мов написання вихідного тексту прикладної програми простирається від машинного коду до майже природної мови. У машинному коді чи мовою асемблера програмувати складніше, ніж алгоритмічною мовою високого рівня, але зате виходить більш короткий код програми, потрібна менша ємність пам'яті програми і виконується така програма швидше.

Об'єктні коди, отримані шляхом трансляції вихідних програм, написаних алгоритмічною мовою високого рівня, займають у пам'яті більше місця і вимагають більшого часу на виконання. Вибір мовних засобів складання вихідних програм у кожнім конкретному випадку залежить від характеристик прикладної задачі, досвіду програміста і припустимих витрат на розробку.

На думку [1], величезна більшість прикладних задач керування об'єктами внаслідок того, що вони повинні зважуватися в реальному часі, пред'являє настільки високі вимоги по швидкодії, що для їхнього рішення основним мовним засобом написання прикладних програм ще довгі роки буде залишатися мова асемблера. Це положення про переважне використання мови асемблера підкріплюється і тією обставиною, що однокристальні мікроконтролери мають обмежений обсяг резидентной пам'яті програм і, отже, критичні до довжини прикладних програм.

2. Процедури і підпрограми

При розробці мікроконтролерних систем можуть бути використані два способи організації прикладних програм: монолітний і модульний. При першому способі вся прикладна програма розробляється як єдине ціле. При другому вона будується з окремих програмних блоків, кожний з який реалізує деяку процедуру обробки даних чи керування. Взаємозв'язок блоків визначається розроблювачем при монтажі з цих блоків закінченої прикладної програми.

Окремі фрагменти прикладної програми можуть бути отримані у вигляді лінійної послідовності блоків, інші (багаторазово використовувані) звичайно оформляються у виді підпрограм, до яких прикладна програма, названа основною, має можливість звернутися по мірі необхідності. Підпрограма повинна мати наступні властивості: виконувати закінчену процедуру обробки даних, мати тільки один вхід і один вихід і не мати ефект післядії, при якому поточне виконання підпрограми робило би вплив на її наступні виконання.

4.2.1. Виклик підпрограми

Звертання до підпрограми здійснюється по команді виклику CALL MARK, де MARK - символічне ім'я процедури. Ім'я процедури використовується в якості мітки, що відзначає одну з команд (найчастіше першу) підпрограми. Для Intel 8051 мнемонічне значення CALL є узагальненим і транслюється в одну з команд ACALL чи LCALL у залежності від адресної відстані викликуваної підпрограми.

По команді CALL у стеці зберігається значення лічильника команд, і повернення з підпрограми здійснюється в те місце основної програми, звідки був здійснений виклик (до команди основної програми, що випливає за командою CALL). Для цього будь-яка підпрограма повинна закінчуватися командою повернення RET, що здійснює відновлення вмісту програмного лічильника зі стека.

Досить часто виникає необхідність такої організації обчислювального процесу, при якій підпрограма викликає іншу підпрограму, та у свою чергу викликає наступну і т.д. Цей процес називається вкладенням підпрограм. Число підпрограм, що можуть бути викликані в такий спосіб (глибина вкладеності підпрограм), обмежується тільки ємністю стека.

4.2.2. Збереження параметрів основної програми

Іноді при звертанні до підпрограми виникає необхідність зберегти не тільки адресу повернення в основну програму, але і вміст окремих робочих регістрів. Зручним способом для цього є переключення банку регістрів. Наприклад, якщо основна програма використовує банк регістрів 0, то підпрограма може використовувати банк регістрів 1. Однак переключення банку регістрів не забезпечує збереження вмісту акумулятора, що приводить до необхідності створювати в одному з робочих регістрів чи у пам'яті копію акумулятора.

4.2.3. Передача параметрів

Для успішної роботи будь-якої підпрограми необхідно однозначно визначити спосіб передачі в неї вихідних даних і спосіб виводу результату її роботи. Підпрограма, якій потрібна додаткова інформація у виді параметрів чи настроювання операндів, називається параметризуємою.

Одержали поширення чотири способи передачі параметрів: через пам'ять, через регістри загального призначення, через регістр ознак і через стек.

При передачі вхідних параметрів через пам'ять основна програма обов'язково містить команди завантаження деяких комірок пам'яті, а підпрограма - команди зчитування з цих комірок. При передачі вхідних параметрів підпрограма повинна завантажити деякі комірки пам'яті, а основна програма - зчитати.

Передача параметрів через регістри здійснюється аналогічним образом.

Третій спосіб передачі параметрів - через регістр ознак - зручно використовувати при передачі вихідних параметрів (наприклад, у підпрограмах порівняння чисел). У цьому випадку підпрограма повинна установити (чи скинути) відповідні ознаки, а основна програма - проаналізувати їхнє значення. Intel 8051 має великі можливості для передачі параметрів через ознаки. У ньому мається 128 прапорів користувача, доступних для модифікації й аналізу.

Спосіб передачі через стек дозволяє використовувати як параметр вміст лічильника команд.

Використання процедур, оформлених у виді підпрограм, при розробці програмного забезпечення має ряд достоїнств. Насамперед відносно прості модулі, виділені зі складної програми, можуть програмуватися декількома розроблювачами з метою скорочення часу проектування. Ще більш важливим є те, що будь-яка підпрограма допускає автономне налагодження. Це, як правило, багаторазово скорочує час налагодження всього прикладного програмного забезпечення. І, нарешті, механізм використання підпрограм зменшує довжину прикладної програми, що має своїм наслідком зменшення ємності пам'яті, що вимагається, програмами.

Істотною є і та обставина, що налагоджені процедури організуються розроблювачами в бібліотеки параметризуємих підпрограм і можуть бути багаторазово використані в проектній роботі. Бібліотека підпрограм повинна будуватися на основі угоди про єдиний спосіб обміну параметрами.

Правила запису програм мовою асемблера

Вихідний текст програми мовою асемблера має визначений формат. Кожна команда і директива являє собою рядок:

МІТКА ОПЕРАЦІЯ ОПЕРАНД(И) КОМЕНТАРІ

Поля можуть відокремлюватися друг від друга довільним числом пробілів і табуляцією.

4.3.1. Мітка

В полі мітки розміщається символічне ім'я комірки пам'яті, у якій зберігається відзначена команда чи операнд. Мітка являє собою буквено-цифрову комбінацію, що починається з букви. Використовуються тільки

букви латинського алфавіту. Асемблер А51 допускає використання в мітках символу підкреслення (_). Мітка завжди завершується двокрапкою (:).

Директиви асемблера не перетворяться в двійкові коди, а тому не можуть мати міток. Виключення складають директиви резервування пам'яті і визначення даних (DS, DB, DW). У директив, що визначають символічні імена, у поле мітки записується визначаєме символічне ім'я, після якого двокрапка не ставиться.

Як символічні імена і міток не можуть бути використані мнемокоди команд, директив і операторів асемблера, зарезервовані імена, а також мнемонічні позначення регістрів і інших внутрішніх блоків мікроконтролера.

4.3.2. Операція

В поле операції записується мнемонічне позначення команди чи директиви асемблера, що є скороченням (абревіатурою) повного англійського найменування виконуваної дії. Наприклад: MOV - move - перемістити, JMP - jump - перейти, DB - define byte - визначити байт.

Для мікроконтролера Intel 8051 використовується строго визначений і обмежений набір мнемонічних кодів. Будь-який інший набір символів, розміщений у поле операції, сприймається асемблером як помилковий.

4.3.3. Операнди

У цьому полі визначаються операнди (чи операнд), що беруть участь в операції. Команди асемблера можуть бути без-, одно- чи двохоперандними. Операнди розділяються комами (,).

Операнд може бути заданий безпосередньо чи у виді його адреси (прямої чи непрямої). Безпосередній операнд представляється числом (MOV A, #15) чи символічним ім'ям (ADDC A, #OPER2) з обов'язковим покажчиком префікса безпосереднього операнда (#). Пряма адреса операнда може бути задана мнемонічним позначенням (IN A, P1), числом (INC 40), символічним ім'ям (MOV A, MEMORY). Вказівкою на непряму адресацію служить префікс @. У командах передачі керування операндом може бути число (LCALL 0135H), мітка (JMP LABEL), непряма адреса (JMP @A) чи вираження (JMP $ - 2, де $ - поточний вміст лічильника команд).

Використовувані в якості операндів символічні імена і мітки повинні бути визначені, а числа представлені з вказівкою системи числення, для чого використовується суфікс (буква, що коштує після числа): B – для двійкової, Q – для восьмирічної, D – для десяткової і H – для шістнадцатирічної. Число без суфікса за замовчуванням вважається десятковим.

Асемблер А51 допускає використання виражень у полі операндів, значення яких обчислюються в процесі трансляції.

Вираження являє собою сукупність символічних імен і чисел, пов'язаних операторами асемблера. Оператори асемблера забезпечують

виконання арифметичних («+» - додавання, «-» - віднімання, «*» - множення, «/» - ціле ділення, MOD – ділення по модулі) і логічних (OR – АБО, AND - І, XOR - виключаєче АБО, NOT - заперечення) операцій у форматі 2-байтних слів. Наприклад, запис ADD A, #((NOT 13)+1) еквівалентний запису ADD A, #0F3H і забезпечує додавання вмісту акумулятора з числом -13, представленим у додатковому коді.

Широко використовуються також оператори LOW і HIGH, що дозволяють обчислити молодший і старший байти 2-байтного операнда.

4.3.4. Коментар

Поле коментарю може бути використано програмістом для текстового чи символьного пояснення логічної організації прикладної програми. Поле коментарю цілком ігнорується асемблером, а тому в ньому припустимо використовувати будь-які символи. За правилами мови асемблера поле коментарю починається з крапки з комою (;).

Директиви асемблера

Асемблер транслює вихідну програму в об'єктні коди. Хоча він бере на себе більшість з рутинних задач програміста, такі як присвоєння дійсних адрес, перетворення чисел, присвоєння дійсних значень символьним змінним і т.п., програміст усе-таки повинний указати йому деякі параметри: початкова адреса прикладної програми, кінець асемблірuємої програми, формати даних і т.п. Усю цю інформацію програміст вставляє у вихідний текст прикладної програми у виді директив, що тільки керують процесом трансляції і не перетворюються в коди об'єктної програми.

Асемблер підтримує ряд директив, що дозволяють дати символічне визначення змінним, резервують і ініціалізують простір пам'яті, визначають розташування згенерованого об'єктного коду в пам'яті. За винятком DB і DW директиви не роблять об'єктний код. Директиви використовуються, щоб змінити стан асемблера, визначити об'єкти і додати інформацію до об'єктного файлу.

Директиви асемблера можуть бути розділені на ряд категорій:

- символічні визначення,
- резервування простору пам'яті,
- ініціалізація даних,
- керування станом асемблера,
- вибір сегментів,
- визначення макрокоманд.

Далі перелічуються всі директиви по категоріях і коротко описуються результати їхньої дії. Основні часто використовувані директиви перераховані за алфавітом в додатку. Інформацію з інших можна знайти в довідковій системі ProView.

4.4.1. Директиви символічних визначень

Директиви символічних визначень можуть бути використані для того, щоб резервувати простір пам'яті, поставити у відповідність символічним іменам визначені числові значення, регістри процесора і сегменти. Ці директиви вимагають, щоб ім'я символу було визначено поряд з адресою, числовим значенням, чи регістром типом сегмента.

Директива	Опис
BIT	Визначає символічне ім'я, що посилається на адресу біта.
CODE	Визначає символічне ім'я, що посилається на адресу коду.
DATA	Визначає символічне ім'я, що посилається на адресу резидентної пам'яті даних.
EQU	Призначає символічному імені числове значення чи ім'я регістра.
IDATA	Визначає символічне ім'я, що посилається на побічно адресуєму адресу резидентної пам'яті даних.
SEGMENT	Повідомляє ім'я переміщуваного сегмента, його тип і розташування.
SET	Призначає символічне ім'я числовому значенню чи регістру. Ім'я може бути згодом змінене за допомогою директиви SET.
XDATA	Визначає символічне ім'я, що посилається на адресу зовнішньої пам'яті даних.

4.4.2. Директиви резервування й ініціалізації пам'яті

Ці директиви використовуються для резервування й ініціалізації слів, байтів чи бітів. В абсолютному сегменті зарезервований простір починається з поточної адреси. У переміщуваному сегменті зарезервований простір починається з поточного зсуву. Покажчик розташування підтримується окремо для кожного сегмента, до нього можна звертатися, використовуючи символ ($).

Директива	Опис
DB	Заносить у пам'ять програм байтову константу.
DBIT	Резервує простір у бітовому сегменті.
DS	Резервує простір пам'яті в поточному сегменті.
DW	Ініціалізує пам'ять значенням слова.

4.4.3. Директиви компонування програми

Ви можете використовувати директиви компонування програми для того, щоб дати об'єктному модулю ім'я і визначити загальні і зовнішні символи. Ці директиви використовуються в L51 для об'єднання окремих об'єктних модулів у єдиний абсолютний об'єктний модуль.

Директива	Опис

EXTRN Визначає символічні імена, що оголошені в інших об'єктних модулях.

NAME Визначає ім'я об'єктного модуля.

PUBLIC Визначає символічні імена, що можуть використовуватися в інших об'єктних модулях.

4.4.4. Директиви керування станом асемблера

Ці директиви використовуються для того, щоб повідомити про кінець трансляції програми, вибрати початкова адреса чи зсув для сегмента, визначити використовуваний банк регістрів.

Директива Опис

END Повідомляє про кінець трансльованого модуля.

ORG Змінює значення асемблерного лічильника адреси поточного сегмента програми.

USING Вибирає номер банку регістрів загального призначення.

4.4.5. Директиви вибору сегмента

Наступні директиви визначають сегменти даних і коду.

Директива Опис

BSEG Вибирає абсолютний бітовий сегмент.

CSEG Вибирає сегмент програми в машинному коді.

DSEG Вибирає абсолютний сегмент резидентної пам'яті даних.

ISEG Вибирає абсолютний побічно адресуємий сегмент резидентної пам'яті даних.

RSEG Вибирає попередньо визначений переміщуваний сегмент.

XSEG Вибирає абсолютний сегмент зовнішньої пам'яті даних.

4.4.6. Директиви макровизначень

Наступні директиви використовуються для визначення макрокоманд.

Директива Опис

ENDM Закінчує макровизначення.

EXITM Змушує макророзширення негайно завершитися.

IRP Визначає список аргументів.

IRPC Визначає аргумент.

LOCAL Визначає до 16 локальних символів, використовуваних усередині макрокоманди.

MACRO Початок макровизначення, визначає ім'я макрокоманди і параметрів, що можуть бути передані макрокоманді.

REPT Визначає кількість повторень наступних рядків.

Налагодження прикладного програмного забезпечення мікроконтролерів

Після одержання об'єктного коду прикладної програми настає етап налагодження, тобто установлення факту її працездатності, а також виявлення, локалізації й усунення помилок. Без цього етапу ніяке програмне забезпечення взагалі не має права на існування. Налагодження програмного забезпечення являє собою окрему складну задачу, що майже не піддається формалізації і вимагає для свого виконання високого професіоналізму і глибоких знань розроблювача.

Звичайно налагодження прикладного програмного забезпечення здійснюється в кілька етапів. Прості синтаксичні помилки виявляються вже на етапі трансляції. Далі необхідно виконати:

- автономне налагодження кожної процедури в статичному режимі, що дозволяє перевірити правильність проведених обчислень, правильність послідовності переходів усередині процедури (відсутність «зациклення») і т.п.;
- комплексне налагодження програмного забезпечення в статичному режимі, що дозволяє перевірити правильність алгоритму керування (по послідовності формування керуючих впливів);
- комплексне налагодження в динамічному режимі без підключення об'єкта для визначення реального часу виконання програми і її окремих фрагментів.

Варто мати на увазі, що автономне налагодження окремих модулів настільки простіше й ефективніше налагодження всієї прикладної програми, що переходити до етапу комплексного налагодження доцільно тільки після вичерпання всіх засобів автономного налагодження.

Перераховані вище етапи здійснюються звичайно з використанням кросів-систем, до яких і відноситься досліджуваний пакет ProView.

До складу кросів-систем входять програми-відладчики, що моделюють виконання програм, написаних для мікроконтролерів. Такі програмні імітатори дозволяють ефективно налагоджувати обчислювальні процедури, а також сам алгоритм функціонування контролера.

Розроблювачу наданий доступ до будь-якого ресурсу мікроконтролера, мається можливість покомандного і пофрагментного виконання програм і останова за умовою, а також підрахунку числа тактів виконання тих чи інших фрагментів програми, ініціювання переривання, дизасемблірування вмісту пам'яті програм і т.п.

Кросс-відладчики дозволяють промоделіровати практично всі можливі варіанти роботи програми і тим самим переконатися в її працездатності. На цьому етапі можлива перевірка працездатності програми в позаштатних ситуаціях, в умовах надходження некоректних вхідних впливів.

Потужні імітатори повинні дозволяти моделювати об'єкти і датчики, що підключаються до мікроконтролера. При цьому з'являється можливість виконувати комплексне налагодження програмного

забезпечення, не побоюючись, що можливі помилки в програмі, чи алгоритмі некоректні дії оператора приведуть до виходу з ладу технічних засобів розроблювальної системи.

Головним недоліком кросів-систем є неможливість прогону програми в реальному масштабі часу.

Найбільш повне і комплексне налагодження прикладного програмного забезпечення разом з апаратними засобами контролера може бути зроблене на інструментальному комп'ютері з так називаним внутрісхемним емулятором.

ЗАВДАННЯ

1. Налагодження підпрограми обчислення скалярного добутку

За допомогою пакета ProView виконайте налагодження підпрограми обчислення скалярного добутку двійкових векторів, складену при виконанні домашнього завдання. Номер варіанта вихідних даних вибирається по останній цифрі номера студентського квитка (залікової книжки).

Таблиця

Варіанти вихідних даних

Варіант	X	Y
0	10110010	10001001
1	11010101	11000011
2	01101010	10101010
3	00111100	11100111
4	10101010	00111100
5	11001111	00010111
6	11100111	00110011
7	11110101	01010101
8	01000111	10011001
9	00011110	11011011

Для налагодження підпрограми:

- створіть файл вихідного тексту програми мовою асемблера, створіть файл проекту задач і додайте до проекту файл вихідного тексту;
- виконайте асемблірування вихідного тексту і виправте синтаксичні помилки;
- завантажте об'єктний код у процесі запуску відладчика;

- задайте початкові значення регістрів і пам'яті;
- здійсніть спробний пуск програми на контрольному прикладі;
- при наявності помилок перейдіть у покроковий режим, локалізуйте і виправте наявні помилки;
- після виправлення помилок повторіть пуск програми на контрольному прикладі і зафіксуйте отриманий результат;
- виконайте генерацію листинга програми, що включений в звіт про виконання роботи, вивчить усі компоненти стандартного листинга.

2. Налагодження варіанта програми

За допомогою пакета ProView виконайте налагодження вашого варіанта програми, що була підготовлена при виконанні домашнього завдання. Для цього:

- створіть файл вихідного тексту програми мовою асемблера, створіть файл проекту задач і додайте до проекту файл вихідного тексту;
- виконайте асемблірування вихідного тексту і виправте синтаксичні помилки;
- завантажте об'єктний код у процесі запуску відладчика;
- задайте початкові значення регістрів і пам'яті;
- здійсніть спробний пуск програми на контрольному прикладі;
- при наявності помилок перейдіть у покроковий режим, локалізуйте і виправте наявні помилки;
- після виправлення помилок повторіть пуск програми на контрольному прикладі і зафіксуйте отриманий результат;
- виконайте генерацію листинга програми, що включений в звіт про виконання роботи, вивчить усі компоненти стандартного листинга.

3. Завдання для заглибленого вивчення

За допомогою довідкової системи пакета ProView вивчить директиви асемблера, опис яких не включено в додаток. Вивчить розділ довідки «Macro Processor».

Модифікуйте Ваші програми з урахуванням нових знань і виконайте налагодження програм.

ЗМІСТ ЗВІТУ

Звіт про лабораторну роботу повинний містити:

- титульний лист;
- мета і задачі роботи;
- листинги з вихідними текстами й об'єктним кодом налагоджених програм з доповненнями, що забезпечують тестування і налагодження;
- перелік помилок, виявлених при налагодженні;
- результати рішення контрольних прикладів;
- висновки по роботі.

Додаток

Директиви асемблера за алфавітом

Нижче даний опис основних, часто використовуваних директив, доступних в асемблері A51 пакета ProView. Директиви можуть включати ряд факультативних полів чи аргументів:

Ім'я	**Опис**
Address	Припустимий код чи адреса даних.
Argument	Значення чи вираження, що заміняє формальне ім'я параметра.
Bit-address	Припустима адреса біта.
Expression	Припустиме вираження.
Label	Припустимий код програми чи мітка даних.
Number	Числова константа, складена тільки з цифр.
Parameter	Символічне ім'я формального параметра.
Register	Ім'я регістра: A, R0, R1, R2, R3, R4, R5, R6, чи R7.
String	Рядок символів і цифр.
Symbol	Припустиме символічне ім'я.

BIT

Опис Директива **BIT** призначає символічне ім'я адресі біта. Формат директиви:

symbol **BIT** *bit-address*

де ***symbol*** - символічне ім'я,
bit-address - адреса біта в резидентній пам'яті даних.

Символічні імена, визначені директивою **BIT,** не можуть бути змінені чи перевизначені.

Приклад

```
     RSEG      DATA_SEG       ;вибір сегмента
     CTRL:          DS   1          ;однобайтова змінна
(CTRL)
     ALARM          BIT  CTRL.0     ;біт у переміщуваному
байті
     SHUT      BIT  ALARM+1    ;наступний біт
     ENABLE_FLAG    BIT  60H        ;абсолютний біт
     DONE_FLAG BIT  24H.2      ;абсолютний біт
```

DATA

Опис Директива **DATA** призначає символічне ім'я адресі резидентної пам'яті даних. Формат директиви:

symbol **DATA** *address*

де ***symbol*** - символічне ім'я, що може використовуватися у всій програмі,
address - адреса резидентної пам'яті даних, повинна

знаходиться в діапазоні від 0 до 255.

Символічні імена, визначені цією директивою, не можуть бути змінені чи перевизначені.

Приклад

```
SERBUF    DATA SBUF
RESULT    DATA 40H
RESULT2   DATA RESULT + 2
PORT1     DATA 90H
```

DB

Опис Директива **DB** заносить у пам'ять програм 8-розрядне значення байта. Директива має наступний формат:

label: **DB** *expression , expression ...*

де ***label:*** - мітка, адреса ініціалізованої пам'яті,
expression - значення байта, що може бути символом, символьним рядком чи вираженням.

Директива DB може бути визначена тільки усередині сегмента коду. Якщо директива використовується в іншому сегменті, асемблер A51 генерує повідомлення про помилку.

Приклад

```
REQUEST:       DB   'PRESS ANY KEY TO CONTINUE', 0
TABLE:         DB   0, 1, 8, 'A', '0', Low(TABLE),
';'
ZERO:          DB   0, ''''
CASE_TAB: DB   Low(REQUEST), Low(TABLE), Low(ZERO)
```

DS

Опис Директива **DS** резервує визначене число байтів у резидентній пам'яті даних, зовнішньої пам'яті даних чи адресному просторі коду програми. Директива має наступний формат:

label: **DS** *expression*

де ***label:*** - мітка, привласнена адресі зарезервованої пам'яті,
expression - кількість зарезервованих байтів, не може містити форвардні посилання, переміщувані символи чи зовнішні символи.

Директива резервує простір у поточному сегменті по поточному адресі. Потім поточна адреса збільшується на значення вираження. Сума лічильника адреси і значення вираження не може перевищувати границю поточного адресного простору.

Примітка. A51 - асемблер із двома проходами по вихідному тексті програми. У першому проході обробляються символи і визначається довжина кожної команди. В другому проході обробляються форвардні посилання і генерується об'єктний код. З цих причин вираження, використовуване в директиві, не може містити форвардні посилання.

Приклад

```
GAP: DS   (($ + 16) AND 0FFF0H) - $
          DS   20
TIME:     DS   8
```

DW

Опис Директива **DW** ініціалізує пам'ять програм 16-розрядними значеннями слова. Директива має наступний формат:

label: **DW** *expression , expression ...*

де ***label:*** - мітка, привласнена адресі зарезервованої пам'яті,
expression - вираження - дані, що можуть містити символ, символьний рядок чи вираження.

Директива може бути визначена тільки усередині сегмента коду. Якщо директива використовується в іншому сегменті, асемблер A51 генерує повідомлення про помилку.

Приклад

```
TABLE:          DW   TABLE, TABLE + 10, ZERO
ZERO:           DW   0
CASE_TAB: DW    CASE0, CASE1, CASE2, CASE3, CASE4
                DW   $
```

END

Опис Директива **END** повідомляє про кінець асемблерного модуля. Будь-який текст в асемблерному файлі, що з'являється після цієї директиви, ігнорується. Директива потрібна в кожнім вихідному асемблерному файлі. Якщо директива відсутня, асемблер генерує повідомлення про фатальну помилку.

Приклад

```
END
```

EQU

Опис Директива **EQU** призначає числовому значенню чи регістру символічне ім'я. Формат директиви:

symbol **EQU** *expression*

symbol **EQU** *register*

де ***symbol*** - символічне ім'я, що заміняється на вираження чи регістр у всієї асемблерної програми,
expression - числове вираження, що не містить форвардних посилань,
register - одне з наступних імен регістра: A, R0-R7.

Символічні імена, визначені директивою, можуть використовуватися в операндах, вираженнях чи адресах. Символи, що визначені як ім'я регістра, можуть використовуватися у всіх командах, що працюють з регістрами. Символічні імена, визначені директивою, не можуть бути змінені чи перевизначені.

Приклад

```
LIMIT     EQU  1200
VALUE     EQU  LIMIT - 200 + 'A'
SERIAL    EQU  SBUF
ACCU EQU  A
COUNT     EQU  R5
```

ORG

Опис Директива **ORG** визначає адресу початку наступного коду чи програми даних. Формат директиви:

ORG *expression*

де ***expression*** - повинно бути абсолютним чи простим переміщуваним вираженням і не може мати форвардних посилань; символи, використовувані у вираженні, можуть посилатися тільки на поточний чи абсолютний сегмент.

Коли асемблер зіштовхується з цією директивою, він обчислює значення вираження і змінює лічильник адрес (внутрішня перемінна асемблера) для поточного сегмента. Якщо директива знаходиться в абсолютному сегменті, лічильнику призначається значення дійсної адреси. Якщо директива знаходиться в переміщуваному сегменті, лічильнику призначається зсув, обумовлений вираженням.

Директива змінює лічильник адрес, але не робить новий сегмент. Невикористаний діапазон адрес можна включити в поточний сегмент. Зверніть увагу, що при використанні абсолютних сегментів лічильник не може посилатися на адресу до початку зсуву.

Примітка. A51 - асемблер із двома проходами по вихідному тексті програми. У першому проході обробляються символи і визначається довжина кожної команди. В другому проході обробляються форвардні посилання і генерується об'єктний код. З цих причин

вираження, використовуване в директиві, не може містити форвардні посилання.

Приклад

```
ORG       100H
ORG       RESTART
ORG       EXTI1
ORG       ($ + 16) AND 0FFF0H
```

RSEG

Опис Директива **RSEG** вибирає переміщуваний сегмент, що був попередньо оголошений директивою SEGMENT. Формат директиви:

RSEG *segment*

де ***segment*** - ім'я сегмента, попередньо визначене директивою SEGMENT.

Для одержання додаткової інформації щодо використання сегментів звернетеся до розділу довідкової системи ProView «Assembly Programs».

Приклад

```
MYPROG    SEGMENT   CODE      ;оголошення сегмента
     RSEG      MYPROG    ;вибір сегмента
               MOV       A, #0
          MOV       P0, A
```

SEGMENT

Опис Директива **SEGMENT** використовується для того, щоб оголосити переміщуваний сегмент. Тип сегмента може бути визначений в оголошенні сегмента. Директива має наступний формат:

segment **SEGMENT** *segtype reloctype*

де ***segment*** - символічне ім'я, призначене сегменту,
segtype - тип сегмента, що визначає адресний простір сегмента; для одержання додаткової інформації див. таблицю нижче,
reloctype - тип переміщення для сегмента, що визначає параметри переміщення для компоновщика L51; звернетеся до таблиці нижче для одержання додаткової інформації.

Ім'я кожного сегмента усередині модуля повинне бути унікально. Однак L51 поєднує сегменти однакового типу. Це також застосовно до сегментів, оголошеним в інших вихідних модулях.

Перемінна segtype визначає адресний простір для сегмента і може бути кожний з наступних:

Segtype	**Опис**
BIT	Бітовий адресний простір (простір резидентної пам'яті даних з адреси 20H по 2FH).
CODE	Простір коду програми.
DATA	Простір резидентної пам'яті даних (адреси з 0 по 127).
IDATA	Побічно адресуємий простір резидентної пам'яті даних (з 0 по 127 чи з 0 по 255).
XDATA	Простір зовнішньої пам'яті даних.

Факультативний параметр *reloctype* - тип переміщення визначає дії компоновщика L51. Наступна таблиця містить список припустимих типів настроювання:

Reloctype	**Опис**
BITADDRESSABLE	Визначає сегмент, що буде переміщений L51 усередину бітової адресуємої області пам'яті (адреси з 20H по 2FH у резидентної пам'яті даних). Дозволений тільки для сегментів DATA, що по довжині не перевищують 16 байтів.
INBLOCK	Визначає сегмент, що повинний міститися в 2048-байтовому блоці. Цей тип допустимий тільки для сегментів CODE.
INPAGE	Визначає сегмент, що повинний міститися в 256-байтовом блоці. Цей тип настроювання допустимо тільки для сегментів CODE і XDATA.
OVERLAYABLE	Визначає, що сегмент може використовувати пам'ять разом з іншими сегментами цього ж типу. При використанні цього типу настроювання ім'я сегмента повинне бути оголошене відповідно до правил C51 чи PL/M-51.
PAGE	Визначає сегмент, чия початкова адреса повинна бути в 256-байтовій границі. Розміщення сегмента виконується компоновщиком L51. Цей тип настроювання допустимий тільки для сегментів CODE і XDATA.
UNIT	Цей тип розміщення заданий за замовчуванням як стандартний тип. Він

визначає сегмент, що починається в границі модуля. Модуль - байт для сегментів CODE, DATA, IDATA і XDATA і біт - для сегмента BIT.

Примітка. Сегментні символи, використовувані у вираженнях, являють собою базову адресу об'єднаного сегмента, що обчислюється компоновщиком L51.

Для одержання додаткової інформації щодо використання сегментів звернетеся до розділу довідкової системи ProView «Assembly Programs».

Приклад

```
STACK     SEGMENT   IDATA
          RSEG      STACK          ;вибір сегмента
               DS        10H            ;резервування 16
байтів
               MOV       SP, #STACK - 1 ;ініціалізація SP
```

SET

Опис Директива **SET** призначає символічне ім'я числовому значенню чи регістру. Формат директиви:

symbol **SET** *expression*

symbol **SET** *register*

де ***symbol*** - символічне ім'я, що заміняється на вираження чи регістр у всієї ассемблерної програмі,
expression - числове вираження, що не містить форвардних посилань,
register - одне з наступних імен регістра: A, R0-R7.

Символічні імена, визначені директивою, можуть використовуватися в операндах, вираженнях чи адресах. Символи, що визначені як ім'я регістра, можуть використовуватися у всіх командах, що працюють з регістрами. Імена, визначені директивою, можуть бути змінені наступними директивами **SET**.

Приклад

```
VALUE     SET  100
VALUE     SET  VALUE / 2
COUNTER   SET  R1
TEMP SET  COUNTER
TEMP SET  VALUE * VALUE
```

XDATA

Опис Директива **XDATA** призначає символічне ім'я адресі зовнішньої пам'яті даних. Формат директиви:

symbol **XDATA** *address*

де ***symbol*** - символічне ім'я, що може використовуватися у всій програмі,
address - адреса зовнішньої пам'яті даних, повинна знаходитися в діапазоні від 0 до 65535.

Символічні імена, визначені цією директивою, не можуть бути змінені чи перевизначені.

Приклад

```
RSEG XSEG1
ORG       100H
DTIM:     DS        6    ;резервує 6 байтів для DTIM
TIME XDATA     DTIM + 0
DATE XDATA     DTIM + 3
```

Практична робота №4
ВЗАЄМОДІЯ МІКРОКОНТРОЛЕРА INTEL 8051 З ОБ'ЄКТАМИ КЕРУВАННЯ

1. МЕТА І ЗМІСТ РОБОТИ

Метою роботи є вивчення основ організації взаємодії мікроконтролерів сімейства 8051 Intel з об'єктами керування. Ціль складається також у продовження початого в лабораторних роботах №1 - №3 вивчення інтегрованого середовища ProView фірми Franklin Software Inc., призначеної для розробки програмного забезпечення цього сімейства. Робота розрахована на 4 години домашньої підготовки і 4-8 годин занять у лабораторії в залежності від кількості виконуваних завдань.

При підготовці до роботи вивчаються основні прийоми програмування, спрямовані на організацію роботи з різними об'єктами керування. Далі студенти складають програми мовою асемблера.

Перед початком лабораторної роботи проводиться колоквіум. Студенти, що успішно відповіли на поставлені питання, допускаються до лабораторної частини роботи.

При виконанні завдання здійснюється введення вихідних текстів програм, ассемблювання і налагодження програм.

Після виконання роботи оформляється звіт із зазначеним нижче змістом.

2. ЗАВДАННЯ ДЛЯ ПІДГОТОВКИ

1. Вивчите апаратні засоби мікроконтролерів, призначені для взаємодії з об'єктами керування

Мікроконтролер Intel 8051.

Паралельні порти вводу-виводу P0-P3. Альтернативні функції порту P3.

Регістри таймерів TH0, TL0 і TH1, TL1. Робота в режимі таймера й у режимі лічильника. Регістр режиму таймера/лічильника TMOD, формат керуючого слова. Режими роботи таймера/лічильника. Регістр керування/статусу таймера TCON, функціональне призначення розрядів.

Буфер приймопередавача SBUF. Режими роботи послідовного порту. Регістр керування прийомопередавачем SCON, функціональне призначення розрядів. Робота UART у мультиконтролерних системах. Установка швидкості прийому/передачі. Регістр керування потужністю PCON, функціональне призначення розрядів. Система переривань мікроконтролера. Регістр пріоритетів переривань IP, регістр маски переривань IE.

2. Вивчите систему команд мікроконтролера з погляду підтримки взаємодії з об'єктами керування

Типи операндов і структура інформаційних зв'язків. Символічні імена регістрів спеціальних функцій і портів. Адресація бітів у регістрах спеціальних функцій, карта адресуємих бітів. Команди роботи з регістрами спеціальних функцій.

3. Вивчите розділ методичних указівок «Взаємодія мікроконтролера з об'єктами керування» і підготуйте до налагодження програми

Для подальшої роботи в лабораторії підготуйте до налагодження наступні програми:

підпрограма обробки зовнішнього переривання,

програма чекання імпульсного сигналу,

програма формування тимчасової затримки програмним способом,

програма формування тимчасової затримки за допомогою таймера,

програма підрахунку числа імпульсів між двома подіями,

програма підрахунку числа імпульсів за заданий проміжок часу на основі двох таймерів/лічильників,

програма опитування групи двійкових датчиків з передачею керування підпрограмам,

програма опитування групи імпульсних датчиків,

програма генерації імпульсного сигналу,

програма роботи з послідовним портом,

програма виміру тимчасових інтервалів програмним способом,

програма виміру тимчасових інтервалів на основі таймера.

Деякі приклади програм залишають можливість оптимізації з урахуванням особливостей системи команд мікроконтролера. Виконаєте оптимізацію. Доповните програми командами і директивами, що забезпечують тестування і подальше налагодження в лабораторії.

Контрольні питання

1. Перелічите характерні риси архітектури однокристальних мікроконтролерів, спрямовані на взаємодію з об'єктами керування.
2. Укажіть призначення регістрів спеціальних функцій.

3. Перелічите альтернативні функції рівнобіжних портів.
4. У якому стані знаходяться рівнобіжні порти після формування сигналу RST?
5. Чи може порт одночасно бути джерелом операнда і приймачем результату операції?
6. Як інвертувати окремі біти портів?
7. З якою частотою инкрементується вміст таймера/лічильника при роботі як таймер?
8. Чому дорівнює максимальна частота підрахунку вхідних сигналів при роботі таймера/лічильника в режимі лічильника?
9. Охарактеризуйте режими роботи таймера-лічильника.
10. Як за допомогою таймера можна виміряти тривалість імпульсного сигналу?
11. Охарактеризуйте режими роботи послідовного порту.
12. Для чого призначений регістр SCON?
13. Поясните принцип роботи UART у мультиконтролерних системах.
14. Як змінити швидкість передачі даних через послідовний порт?
15. Для чого використовується дев'ятий біт?
16. Намалюйте схему переривань. Перелічите й охарактеризуйте типи переривань.
17. Для чого потрібний регістр масок переривання? Як змінити пріоритети переривань?
18. Чим відрізняються команди RET і RETI?
19. Перелічите команди операцій з бітами.
20. Як організувати процедуру чекання події за допомогою однієї команди?
21. Укажіть, які з регістрів спеціальних функцій допускають бітову адресацію.
22. Перелічите засоби програми ProView, призначені для налагодження взаємодії мікроконтролера з об'єктами керування.
23. Які обмеження накладаються на тривалість імпульсного сигналу, що виявляється, при програмній реалізації циклу чекання?
24. Поясните принципи усунення дребезга контактів.
25. Поясните принцип організації процедур підрахунку числа імпульсів між двома подіями і за заданий проміжок часу.
26. У чому полягає табличний спосіб генерації мікроконтролером складних послідовностей керуючих сигналів?
27. Поясните принцип генерації періодичних і аперіодичних сигналів.
28. Як програмно формуються затримки різної тривалості?
29. Як за допомогою мікроконтролера виміряти часовий інтервал? Як оцінити точність виміру?

ВЗАЄМОДІЯ МІКРОКОНТРОЛЕРА З ОБ'ЄКТАМИ КЕРУВАННЯ

1. Переривання

Підпрограма обробки переривання повинна зберегти в стеці вміст тих регістрів, що вона сама будеть використовувати, а перед поверненням у перервану програму повинна відновити їхнього значення.

Підпрограма обробки зовнішнього переривання рівня 0 може, наприклад, мати наступну структуру:

```
          ORG  3              ;завдання адреси вектора
          переривання
          SJMP SUBINO         ;перехід на підпрограму обробки

          ORG  ЗОН
SUBINO:   PUSH PSW            ;збереження в стеці PSW
               PUSH ACC            ;збереження акумулятора
               PUSH У              ;збереження В
          PUSH DPL            ;збереження DPTR
          PUSH DPH
          MOV  PSW, #1000B    ;вибір банку регістрів 1
               MOV  A,#5           ;власне обробка переривань
               MOV  R1,A
          ADD  A,R1
               MOV  R2,A
          POP  DPH            ;відновлення DPTR
          POP  DPL
          POP  У              ;відновлення В
          POP  ACC            ;відновлення акумулятора
          POP  PSW            ;відновлення PSW і номера банку
               RETI                ;повернення
```

2. Введення інформації з датчиків

4.2.1. Опитування двійкового датчика. Чекання події

У пристроях і системах керування об'єктами події фіксуються з використанням різноманітних датчиків цифрового й аналогового типів. Найбільше поширення мають двійкові датчики типу так/ні.

Чекання статичного сигналу. Типова процедура чекання події (`WAIT`) складається з наступних дій: уведення сигналу від датчика, аналізу значення сигналу і передачі керування в залежності від стану датчика. Конкретна програмна реалізація процедури залежить від того, яким образом датчик підключений до мікроконтролера. Наприклад, при підключенні датчика до лінії біта 3 порти 1 програма чекання замикання контакту буде мати вид:

```
WAIT0:    JNB  P1.3, WAIT0 ;чекання розмикання контакту датчика
```

Іншою часткою є процедура чекання розмикання контакту, що може бути реалізована в такий спосіб:

```
WAITC:    JB   P1.3, WAITC ;чекання замикання контакту датчика
```

Для опитування особливо важливих датчиків з метою зменшення часу реакції на виняткову ситуацію в об'єкті доцільно використовувати режим переривання.

Чекання імпульсного сигналу. Особливість процедури чекання імпульсного сигналу полягає в тому, що мікроконтролер повинен знати не тільки факт появ, але і факт закінчення сигналу.

Для програмування цієї процедури зручно скористатися розглянутими вище прикладами, змонтувавши їх послідовно в лінійну програму. Оформляти процедури WAITC і WAIT0 у виді підпрограм недоцільно, тому що це подовжує програму, а довжина і, отже, час виконання програми визначають мінімальну тривалість імпульсу, що може бути виявлений програмою.

Послідовність склеювання процедур WAITC і WAIT0 залежить від форми імпульсу. Для «негативного» імпульсу (1→0→1) процедура WAITC передує процедурі WAIT0, для «позитивного» (0→1→0) випливає за нею.

Нижче приведений приклад програмної реалізації процедури чекання «негативного» імпульсного сигналу при підключенні датчика до біта 3 порти 1 за умови, що початковий стан входу – одиничне:

```
WAITC:    JB   P1.3, WAITC    ;чекання P1.3=0
WAIT0:    JNB  P1.3, WAIT0    ;чекання P1.3=1
```

Програмна реалізація циклу чекання накладає обмеження на тривалість імпульсу: імпульси тривалістю менше часу виконання циклу чекання можуть бути «не помічені» мікроконтролером. Для виявлення короткочасних імпульсів звичайно використовують спосіб фіксації імпульсу на зовнішньому тригері прапора. На вхід у цьому випадку надходить не короткочасний сигнал з датчика, а прапор, формований тригером. Тригер установлюється по фронті імпульсу, а скидається програмним шляхом – видачею спеціального керуючого впливу. Тривалість імпульсу при цьому буде обмежена знизу тільки швидкодією тригера.

4.2.2. Усунення дребезга контактів

При роботі з датчиками, що мають механічні чи електро-механічні контакти (кнопки, клавіші, реле і клавіатури), виникає явище, яке називається дребезгом. Він полягає в тім, що при замиканні контактів можлива поява відскоку (BOUNCE) контактів, що приводить до перехідного процесу. При цьому сигнал з контакту може бути прочитаний мікроконтролером як випадкова послідовність нулів і одиниць. Придушити це небажане явище можна схемотехнічними засобами, але частіше це робиться програмним шляхом.

Найбільше поширення одержали два програмних способи чекання сталого значення:

1) підрахунок заданого числа співпадаючих значень сигналу;
2) тимчасова затримка.

Суть першого способу складається в багаторазовому зчитуванні сигналу з контакту. Підрахунок вдалих опитувань, що знайшли, що контакт стійко замкнут, ведеться програмним лічильником. Якщо після серії вдалих опитувань зустрічається невдалий, то підрахунок починається спочатку. Контакт вважається стійко замкнутим, якщо пішло N вдалих опитувань. Число N підбирається експериментально для кожного типу використовуваних датчиків і лежить у межах від 5 до 50. Приклад програмного придушення дребезга контакту приводиться для випадку, коли датчик імпульсного сигналу підключений до входу T0, рахунок вдалих опитувань ведеться в регістрі R3, N=20:

```
DBNC:     MOV  R3, #20   ;ініціалізація лічильника
```

```
DBNC1:    JB   P3.4, DBNC;якщо контакт розімкнут, то
                          ;почати відлік опитувань спочатку
          DJNZ R3, DBNC1 ;повторювати, поки R3 не стане рівним 0
```

Усунення дребезга контакту шляхом уведення тимчасової затримки полягає в наступному. Програма, знайшовши замикання контакту, забороняє опитування його стану на час, свідомо більше тривалості перехідного процесу. Програма написана для випадку підключення датчика до входу T0 і програмної реалізації тимчасової затримки:

```
DBNCDL:   JTO  DBNCDL    ;чекання нуля на вході T0
          CALL DELAY     ;виклик підпрограми затримки
EXIT:     ...            ;вихід із процедури
```

Тимчасова затримка в межах 1-10 мс підбирається експериментально для кожного типу датчиків і реалізується підпрограмою DELAY.

4.2.3. Підрахунок числа імпульсів

Часто в керуючих програмах виникає необхідність чекання ланцюжка подій, що представляється послідовністю імпульсних сигналів від датчиків. Розглянемо дві типові процедури: підрахунок числа імпульсів між двома подіями і підрахунок числа імпульсів у заданий інтервал часу.

Підрахунок числа імпульсів між двома подіями. Один з можливих варіантів процедури підрахунку може бути реалізований, якщо використовувати вхід T1 як вхід лічильника подій. В акумуляторі фіксується число імпульсів, представлене в двійковому коді (максимальна кількість 255).

```
          MOV  TMOD, #0100000B;настроювання лічильника 1
               MOV  TH1, #0        ;скидання лічильника імпульсів
WAIT0:    JB   P3.4, WAIT0    ;чекання включення рахунка
               SETB TCON.6         ;пуск лічильника 1
WAIT1:    JNB  P3.4, WAIT1    ;чекання вимикання рахунка
          CLR  TCON.6         ;останов лічильника 1
               MOV  A, TH1         ;(акумулятор) ← число
імпульсів
EXIT:     ...                 ;вихід із процедури
```

Підрахунок числа імпульсів за заданий проміжок часу. При рішенні задачі перетворення числ-імпульсного коду в двоичный код, а також у ряді інших задач може виникнути необхідність підрахунку числа імпульсів за заданий інтервал часу. Ця процедура може бути реалізована різними способами:

програмною реалізацією тимчасового інтервалу і програмним підрахунком числа імпульсів на вході;

програмною реалізацією тимчасового інтервалу й апаратним підрахунком числа імпульсів (на внутрішньому таймері/лічильнику);

апаратною реалізацією тимчасового інтервалу і програмним підрахунком числа імпульсів;

апаратна реалізація тимчасового інтервалу з апаратним підрахунком числа імпульсів.

Четвертий спосіб підрахунку імпульсів вимагає використання двох лічильників. На Т/С1 можна виконувати підрахунок числа імпульсів, а на Т/С0 - відлік заданого інтервалу. Датчик імпульсів повинний бути підключений до входу Т1:

```
TIME EQU  NOT(10000)+1   ;визначення константи TIME для
                                  ;відліку інтервалу 10 мс
          MOV  TMOD, #01010001B   ;настроювання Т/С, 16 біт
                                       ;1 – лічильник, 0 –
таймер
               CLR  A                  ;скидання акумулятора
          MOV  TH1, A             ;скидання Т/З1
          MOV  TL1, A
  MOV     TH0, #HIGH(TIME)    ;завантаження в Т/З0
          MOV  TL0, #LOW(TIME)    ;константи TIME
               ORL  TCON, #50H         ;пуск Т/З1 і Т/З0
WAIT:     JBC  TCON.5, EXIT       ;перевірка переповнення Т/З0
               SJMP WAIT               ;цикл, якщо TF=0
EXIT:     MOV  B, TH1             ;(B)(A) ← число імпульсів
          MOV  A, TL1
               ...                     ;вихід із процедури
```

4.2.4. Опитування групи двійкових датчиків

Мікроконтролери найчастіше мають справу не з одним датчиком, як у розглянутих вище прикладах, а з групою автономних, логічно незалежних чи взаємозалежних, формуючих двійковий код датчиків. При цьому мікроконтролер може виконувати процедуру опитування датчиків і передачі керування окремим фрагментам прикладної програми в залежності від прийнятого коду.

Програмну реалізацію процедури чекання заданого коду (WTCODE) розглянемо для випадку підключення групи з восьми взаємозалежних статичних датчиків до входів порту 1:

```
WTCODE:   MOV  A, #10    ;завантаження в акумулятор еталонного
код
WAIT:     CJNE A, P1, WAIT    ;якщо кодова комбінація на входах
                                  ;порту 1 не збіглася з
еталонним
                               ;значенням, то чекати
EXIT:     ...                 ;вихід із процедури
```

При опитуванні двійкових датчиків передачу керування зручно здійснювати по таблиці переходів. Нижче приводиться текст програми, що здійснює передачу керування однієї з восьми прикладних програм PROG0 - PROG7, що розташовані в межах однієї сторінки пам'яті програм. Передача виробляється в залежності від кодової комбінації на входах P1.0 - P1.2:

```
GOCODE:   MOV  R0, #LOW BASE  ;завантаження в R0 початкової
адреси
                              ;таблиці переходів
          IN   A, P1          ;уведення байта
          ANL  A, #00000111B  ;виділення молодших біт
          ADD  A, R0          ;формування адреси рядка
                              ;у таблиці переходів
          JMPP  @A            ;передача керування
BASE:     DB   LOW PROG0      ;таблиця переходів
          ...
          DB   LOW PROG7
```

Програма опитує і виділяє сигнали від трьох датчиків шляхом маскирования старших бітів акумулятора.

Адреса рядка таблиці, у якій зберігаються адреси переходів, обчислюється як сума відносного (усередині поточної сторінки резидентной пам'яті програм) початкової адреси таблиці BASE і коду, прийнятого від датчиків. Команда JMPP @A, таблиця BASE і програми PROGO – PROG7 повинні розташовуватися в одній сторінці пам'яті програм.

При роботі з групою датчиків часто виникає необхідність здійснювати передачу керування не тільки в залежності від двійкового еквівалента прийнятого коду, як у розглянутому прикладі, але й у залежності від співвідношення прийнятого коду і деякої заздалегідь визначеної уставки. Нехай, наприклад, у порту 1 від групи двійкових датчиків формується восьмибитный двійковий код. Якщо код дорівнює десятковому еквіваленту 135, то необхідно передати керування програмі з міткою LABELA, у противному випадку – програмі з міткою LABELB:

```
          MOV  A, #135        ;завантаження уставки
          CJNE A, P1, LABELB  ;порівняння і передача керування
LABELA:   ...
```

Опитування групи імпульсних датчиків. Ця процедура складається з послідовності дій: чекання замикання одного з контактів, усунення дребезга, чекання розмикання замкнутого контакту.

Програмна реалізація процедури для випадку підключення чотирьох імпульсних датчиків до входів 0 - 3 порти 1 буде мати вид:

```
KBRD:     IN   A, P1          ;уведення коду
          CPL  A              ;інверсія коду
          ANL  A, #00001111B  ;є замкнутий контакт?
          JZ   KBRD           ;якщо жоден контакт не замкнут,
                              ;то чекати
          MOV  R2, A          ;передача прийнятого коду в R2
DBNC:     CALL DELAY          ;усунення дребезга
WAIT:     IN   A, P1          ;уведення коду
          CPL  A              ;інверсія коду
          ANL  A, #00001111B  ;є замкнутий контакт?
```

```
        JNZ  WAIT           ;якщо контакт замкнут, то чекати,
EXIT:   ...                 ;інакше вихід із процедури
```

Аналіз стану контактів здійснюється накладенням маски на прийнятий від датчиків код. Для датчиків, що формують «негативний» імпульс, прийнятий код попередньо інвертується.

Для групи імпульсних датчиків, що представляють собою клавішний регістр, процедура KBRD повинна бути доповнена процедурами ідентифікації натиснутої клавіші і захисту від одночасного натискання двох і більш клавіш.

Ідентифікація натиснутої клавіші може здійснюватися двома способами: по чи таблиці програмно. При табличному способі перекодування в пам'яті програм повинна знаходитися таблиця двійкових еквівалентів кодів клавіш. Програмне перетворення унітарного коду, прийнятого від клавіатури, у двійковий може бути виконано методом зрушень вихідного унітарного коду і підрахунком числа зрушень на лічильнику до появи першого переносу [1].

4.3. Керуючі сигнали

4.3.1. Формування статичних сигналів

Для керування виконавчим пристроєм, що працює за принципом включене/виключено, на відповідної вихідній лінії порту необхідно сформувати статичний сигнал 0 чи 1, що реалізується командами висновку безпосереднього операнда, що містить у необхідному біті значення 0 чи 1.

У випадку рівнобіжного керування групою автономних виконавчих пристроїв, підключених до вихідного порту, формується не двіковий керуючий вплив, а керуюче слово, кожному з розрядів якого ставиться у відповідність 1 чи 0 у залежності від того, які виконавчі пристрої повинні бути включені, а які виключені.

Керуючі слова зручно формувати командами логічних операцій над вмістом порту. Команда `ANL` використовується для скидання тих бітів, що у масці задані нулем. Команда `ORL` використовується для установки бітів керуючого слова. Командою `XRL` здійснюється інверсія біта.

Для формування складних послідовностей керуючих слів звичайно використовують табличний спосіб, при якому всі можливі слова упаковані в таблицю, а прикладна програма обчислює адресу необхідного слова, вибирає його з таблиці і передає в порт.

4.3.2. Формування імпульсних сигналів

Імпульс можна одержати послідовною видачею сигналів включити і відключити з проміжним викликом підпрограми тимчасової затримки:

```
PULS:                       ;видача імпульсу в лінію 3 порти 1
ON:     ANL  P1, #11110111B ;включення
     CALL DELAY             ;тимчасова затримка
OFF: ORL  P1, #00001000B ;відключення
     ...
```

Тривалість імпульсу визначається тимчасовою затримкою, реалізованою підпрограмою DELAY.

Генерація меандру. У цьому випадку можна скористатися процедурою видачі імпульсу PULS і підпрограмою затримки, рівній половині періоду сигналу DLYX:

```
MEANDR:
XCOR:     CPL       P1.3
               ACALL     DLYX
          SJMP      XCOR
```

Нескінченний періодичний сигнал формується в лінії 3 порти 1. На інших лініях сигнали залишаються незмінними.

Формування аперіодичних керуючих сигналів. Послідовність імпульсних сигналів з довільною тривалістю і шпаруватістю може бути отримана аналогічним образом, тобто шляхом чергування процедур видачі значення 0 чи 1 і виклику підпрограм тимчасових затримок заданих длительностей.

4.4. Робота з послідовним портом

Послідовний порт мікроконтролера може використовуватися у виді регістра зрушення для розширення чи виводу в якості UART з фіксованою чи перемінною швидкістю послідовного обміну і можливістю дуплексного включення. Тобто через порт можна передавати і приймати дані одночасно. Порт може приймати черговий байт навіть у тому випадку, якщо вже прийнятий до цей байт не був прочитаний з регістра приймача. Однак, якщо до закінчення прийому находящийся до регістрі приймача байт не буде прочитаний, прийнятий байт губиться. Програмний доступ до регістрів приймача і передавача здійснюється звертанням до регістру спеціальних функцій SBUF.

Нижче приведений приклад фрагмента програми, що приймає з послідовного порту байт і відправляє його назад у послідовний порт, налаштований на 8-бітний режим зі швидкістю передачі 1200 бод при тактовій частоті мікроконтролера 6Мгц.

```
   MOV SCON,#052H   ;установка режиму 8-бітного UART
   MOV TMOD,#020H ;установка режиму автозавантаження таймера 1
     MOV TCON,#069H
     MOV TH1,#0F3H  ;автовантажуємe значення для одержання
                          ;швидкості 1200 бод на частоті 6 Мгц
                         ;прийом символу з порту
CIN: JNB RI,CIN     ;чекання завершення прийому
     MOV A,SBUF     ;читання символу
               CLR RI          ;очищення прапора прийому
                               ;видача символу в послідовний порт
COUT:     JNB TI,COUT    ;чекання закінчення передачі
          CLR TI         ;очищення прапора передачі
               MOV SBUF,A     ;видача символу
     SJMP CIN

     END
```

4.5. Реалізація функцій часу

4.5.1. Програмне формування тимчасової затримки

Тимчасова затримка малої тривалості. Процедура реалізації тимчасової затримки використовує метод програмних циклів. При цьому в деякий робочий регістр завантажується число, що потім у кожнім проході циклу зменшується на 1. Так продовжується доти, поки вміст робочого регістра не стане рівним нулю, що інтерпретується програмою як момент виходу з циклу. Час затримки при цьому визначається числом, завантаженим у робочий регістр, і часом виконання команд, що утворять програмний цикл.

Припустимо, що в керуючій програмі необхідно реалізувати тимчасову затримку 99 мкс. Фрагмент програми, що реалізує тимчасову затримку, потрібно оформити у виді підпрограми, тому що передбачається, що основна керуюча програма буде робити до неї багаторазові звертання для формування вихідних імпульсних сигналів, тривалість яких кратна 99 мкс:

```
DELAY:    MOV  R2, #X    ;(R2)←X
COUNT:    DJNZ R2, COUNT ;декремент R2 і цикл, якщо не нуль
          RET            ;повернення
```

Для одержання необхідної тимчасової затримки необхідно визначити число X, що завантажується в робочий регістр. Визначення числа X виконується на основі розрахунку часу виконання команд, що утворять дану підпрограму. При цьому необхідно враховувати, що команди MOV і RET виконуються однократно, а число повторень команди DJNZ дорівнює числу X. Крім того, звертання до підпрограми тимчасової затримки здійснюється по команді CALL DELAY, час виконання якої також необхідно враховувати при підрахунку тимчасової затримки. В описі команд мікроконтролера вказується, за скількох машинних циклів (МЦ) виповнюється кожна команда. На підставі цих даних визначається сумарне число машинних циклів у підпрограмі: CALL - 2 МЦ, MOV - 1 МЦ, DJNZ - 2 МЦ, RET - 2 МЦ.

При тактовій частоті 12 Мгц кожен машинний цикл виконується за 1 мкс. Таким чином, підпрограма виконується за час $2 + 1 + 2X + 2 = 5 + 2X$ мкс. Для реалізації тимчасової затримки 99 мкс число $X = (99 - 5)/2 = 47$.

У даному випадку при завантаженні в регістр R2 числа 47 необхідна тимчасова затримка (99 мкс) реалізується точно. Якщо число X виходить дробовим, то тимчасову затримку можна реалізувати лише приблизно. Для більш точного підстроювання в підпрограму можуть бути включені команди NOP, час виконання кожної з який дорівнює 1 мкс.

Мінімальна тимчасова затримка, реалізована підпрограмою DELAY, складає 7 мкс ($X = 1$). Тимчасову затримку меншої тривалості програмним шляхом можна реалізувати, включаючи в програму ланцюжка команд NOP.

Максимальна тривалість затримки, реалізована підпрограмою DELAY, складає 515 мкс ($X = 255$).

Для реалізації затримки більшої тривалості можна рекомендувати збільшити тіло циклу включенням додаткових чи команд використовувати метод вкладених циклів. Так, наприклад, якщо в підпрограму DELAY перед командою DJNZ

уставити додатково двох команд NOP, те максимальна затримка складе 5 + X(2 + 1) = 5+3 * 255 = 770 мкс (тобто майже в півтора разу більше).

Тимчасова затримка великої тривалості. Затримка великої тривалості може бути реалізована методом вкладених циклів. Числа X и Y вибираються зі співвідношення T = 2 + 1 + X(1 + 2Y + 2) + 2, де T - реалізований часовий інтервал у мікросекундах. Максимальний часовий інтервал, реалізований таким способом, при X = Y = 255 складає 130.82 мс, тобто приблизно 0.13 с.

Як приклад розглянемо підпрограму, що реалізує тимчасову затримку 100 мс:

```
DELAY:    MOV  R1, #195       ;завантаження X
LOOPEX:   MOV  R2, #254       ;завантаження Y
LOOPIN:   DJNZ R2, LOOPIN     ;декремент R2 і внутрішній цикл,
                               ;якщо (R2)(0
          DJNZ R1, LOOPEX     ;декремент R1 і зовнішній цикл,
                               ;якщо (R1)(0
          MOV  R3, #174       ;точне підстроювання
LOOPAD:   DJNZ R3, LOOPAD     ;тимчасові затримки
          NOP                 ;затримки
          RET
```

Два вкладених цикли реалізують тимчасову затримку триваліcтю

5 + 195(3 + 2*254) = 99 650 мкс, а додатковий цикл LOOPAD і команда NOP реалізує затримку 350 мкс і тим самим забезпечує точне настроювання тимчасового інтервалу.

Тимчасова затримка тривалістю 1 с. З розглянутого приклада видно, що секунда є дуже великим інтервалом часу в порівнянні з тактовою частотою мікроконтролера. Такі затримки складно реалізувати методом вкладених циклів, тому їх звичайно набирають з точно підбудованих затримок меншої тривалості. Наприклад, затримку в 1 з можна реалізувати десятикратним викликом підпрограми, що реалізує затримку 100 мс:

```
ONESEC:   MOV  R3, #10 ;завантаження в R3 числа викликів
підпрограм
LOOP:     CALL DELAY     ;затримка 100 мс
          DJNZ R3, LOOP  ;декремент R3 і цикл, якщо (R3)≠0
```

Погрішність підпрограми складає 21 мкс. Для дуже багатьох застосувань це досить висока точність, хоча реалізовані на основі цієї програми годинник астрономічного часу за добу «утечуть» приблизно на 1.8 с.

4.5.2. Формування тимчасової затримки таймером

Затримка малої тривалості. Недоліком програмного способу реалізації тимчасової затримки є нераціональне використання ресурсів мікроконтролера: під час формування затримки він практично простоює, тому що не може вирішувати ніяких задач керування об'єктом. У той же час апаратні засоби дозволяють реалізувати тимчасові затримки на тлі основної програми роботи.

На вхід таймера/лічильника (Т/З) можуть надходити сигнали синхронізації з частотою 1 Мгц (Т/С в режимі таймера) чи сигнали від зовнішнього джерела (Т/С в режимі лічильника). Оба цих режима можуть бути використані для формування затримок. Якщо використовувати Т/С в режимі таймера повного формату (16 біт), то можна одержати затримки в діапазоні 1 - 65 536 мкс.

Як приклад розглянемо організацію тимчасової затримки тривалістю 50 мс. Передбачається, що біт IE.7 встановлений.

```
;організація переходу до мітки NEXT при переповненні Т/З0
          ORG  0BH       ;адреса вектора переривання від Т/З0
          CLR  TCON.4    ;останов Т/З0
          RETI           ;вихід з підпрограми обробки переривання

          ORG  100H      ;початкова адреса програми
          MOV  TMOD, #01H              ;настроювання Т/З0
               MOV  TL0, #LOW(NOT(50000-1))  ;завантаження
таймера
          MOV  TH0, #HIGH(NOT(50000-1))
               SETB TCON.4    ;старт Т/З0
          SETB IE.1      ;переведення у режим холостого ходу
               SETB PCON.0
NEXT:     ...
```

4.5.3. Вимір тимчасових інтервалів

У задачах керування часто виникає необхідність виміру проміжку часу між двома подіями. Звичайно події в об'єкті керування представляються сигналами від двійкових датчиків. Вважаючи подіями фронт і спад імпульсу, можна визначати тимчасові характеристики імпульсних сигналів: тривалість, період і парність.

Найпростішим способом виміру тривалості імпульсу є програмний. Для виявлення подій (фронт і спад імпульсного сигналу) у цьому випадку використовуються типові процедури WAIT, а відлік часу ведеться програмним способом. Для "позитивного" імпульсного сигналу, що надходить на вхід Т0, програма виміру його тривалості буде мати вид:

```
MSCONT:   MOV  R7, #0    ;скидання лічильника
WAIT0:    JNT0 WAIT0     ;чекання фронту сигналу
COUNT:    INC  R7        ;інкремент лічильника
          JT0  COUNT     ;чекання спаду сигналу
EXIT:     ...            ;вихід із процедури
```

Після виходу з процедури вміст лічильника R7 пропорційно тривалості імпульсу.

Для нормальної роботи цієї програми необхідно, щоб звертання до неї вироблялося в моменти, коли на вході Т0 є присутнім сигнал нульового рівня. Верхня межа вимірюваної тривалості «позитивного» імпульсу складе 255(1 + 2) мкс = 765 мкс. Ця межа може бути збільшена включенням у цикл COUNT додаткових команд NOP. Максимальна погрішність вимірів 3 мкс.

Для виміру тривалості сигналу може бути використаний таймер. Особливо ефективне використання для цієї мети таймера в 8051 Intel, що має вхід дозволу рахунка (альтернативна функція входу INT). Вимірюваний сигнал можна, наприклад, подавати на вхід INT0, а вимір тривалості при цьому буде виконуватися в T/30. Програма виміру тривалості «позитивного» імпульсу буде виглядати так:

```
            MOV  TMOD, #00001001B    ;настроювання T/C0
            MOV  TH0, #0             ;скидання таймера
                 MOV  TL0, #0
                 SETB TCON.4              ;старт T/C0
WAIT0:      JNB  P3.2, WAIT0         ;чекання 1
WAITC:      JB   P3.2, WAITC         ;чекання 0
                 CLR  TCON.4              ;стіп T/C0
EXIT:       ...                      ;вихід із процедури
```

Керування програмі повинне бути передане за умови, що на вході INT0 є присутнім низький рівень. Переривання від T/C0 і зовнішнє переривання по входу INT0 повинні бути заборонені. По завершенні програми в T/C0 буде знаходитися число, пропорційне тривалості «позитивного» імпульсу на вході INT0. Верхня межа виміру дорівнює 65 536 мкс, а максимальна похибка 1 мкс.

При необхідності виміру тимчасових інтервалів більшої тривалості можна програмним способом підраховувати число переповнень від таймера, тобто розширювати його розрядність за рахунок робочого чи регістра осередку резидентной пам'яті даних.

4.6. Засобу ProView для налагодження взаємодії з об'єктами керування

Інтегроване середовище ProView фірми Franklin Software Inc. має кілька вікон, призначених для налагодження взаємодії мікроконтролера з об'єктами керування. По-перше, це вже знайоме по попереднім лабораторних роботах вікно Main Registers (мал. 1), де доступні регістр масок переривання IE, порти P0 – P3, регістр керування/статусу таймера TCON, регістри таймерів THL0 і THL1, а також регістр керування потужністю PCON.

Крім того, через пункт Harware меню View (рис. 32) можна відкрити вікна контролера переривань, таймерів, паралельних портів P0-P3 і послідовного порту UART.

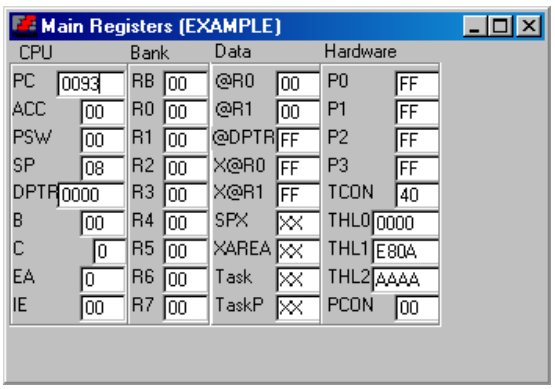


Рис. 32. Вікно регістрів.

У вікні контролера переривань вказується статус і пріоритет усіх джерел переривання.

У вікні таймеру доступний вміст 16-бітного таймера/лічильника THLx, регістра режиму роботи таймера/лічильника TMOD, регістра керування/статусу таймера TCON, прапор переповнення таймера TFx, біт керування таймера TRx. Тут же вказується стан і режим роботи таймера.

У вікнах паралельних портів доступний вміст регістра-засувки LATCH і окремих ліній порту. Стан ліній порту може бути змінене за допомогою миші.

У вікні послідовного порту UART відображається вміст буфера передавача Buffer. У буфер приймача Input може бути введена послідовність байтів. Інформація може бути представлена в символьному виді ASCII чи в шістнадцатирічній формі HEX. Буфер очищається за допомогою кнопки Reset Buffer.

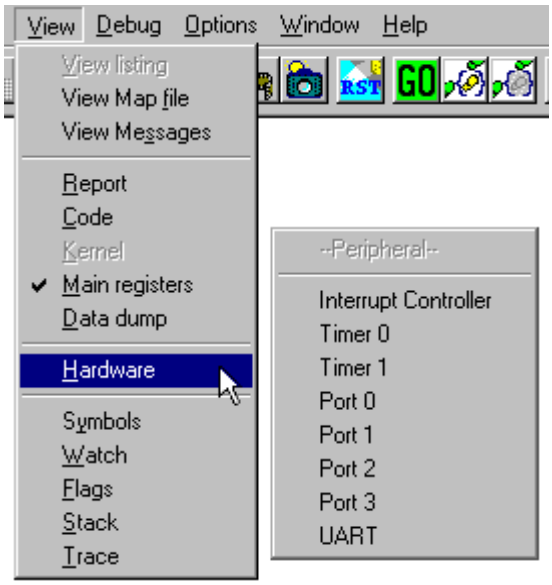


Рис. 33. Меню View

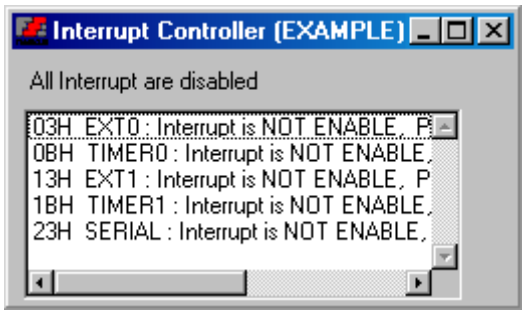


Рис. 35. Вікно контролера переривань

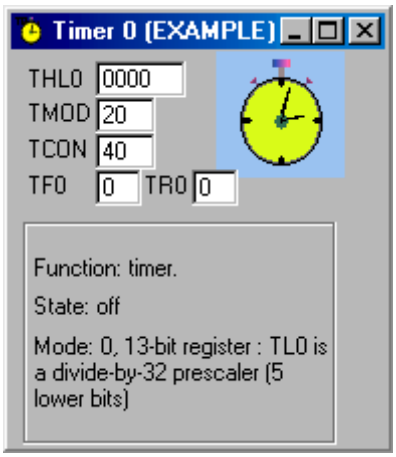


Рис. 34. Вікно таймера

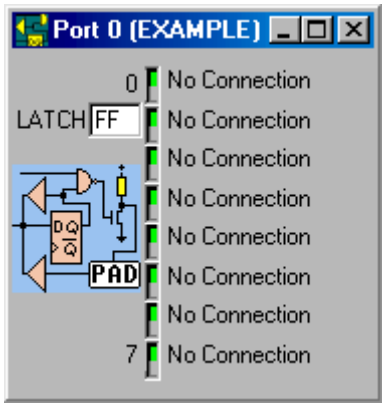


Рис. 36. Вікно паралельного порту

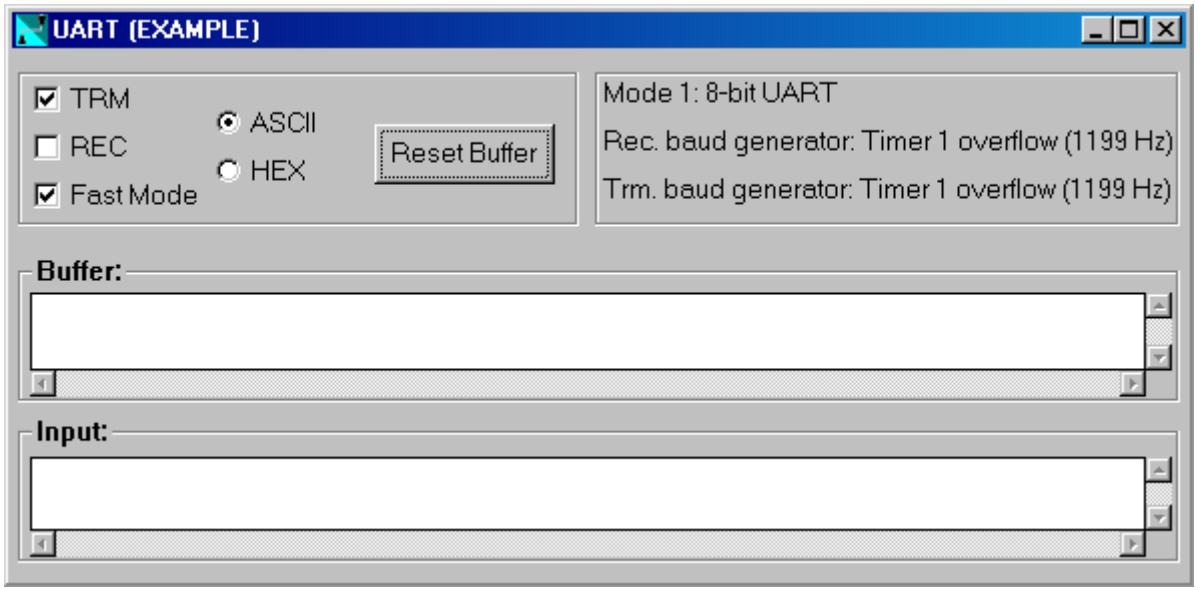


Рис. 37. Вікно послідовного порту

5. ЗАВДАННЯ

За допомогою пакета ProView виконаєте налагодження підготовлених при виконанні домашнього завдання програм:

підпрограма обробки зовнішнього переривання,
програма чекання імпульсного сигналу,
програма формування тимчасової затримки програмним способом,
програма формування тимчасової затримки за допомогою таймера,
програма підрахунку числа імпульсів між двома подіями,
програма підрахунку числа імпульсів за заданий проміжок часу на основі двох таймерів/лічильників,
програма опитування групи двійкових датчиків з передачею керування підпрограмам,
програма опитування групи імпульсних датчиків,
програма генерації імпульсного сигналу,
програма роботи з послідовним портом,
програма виміру тимчасових інтервалів програмним способом,
програма виміру тимчасових інтервалів на основі таймера.

Використовуйте оптимизированные варіанти програм і їх відладкові доповнення.

Для налагодження кожної програми:

- створіть файл вихідного тексту програми мовою асемблера, створіть файл проекту задач і додайте до проекту файл вихідного тексту,
- виконаєте асемблювання вихідного тексту і виправте синтаксичні помилки,
- завантажите об'єктний код у процесі запуску отладчика,
- задайте початкові значення регістрів і пам'яті,
- здійсните спробний пуск програми на контрольному прикладі,
- при наявності помилок перейдіть у покроковий режим, локалізуйте і виправте наявні помилки,
- після виправлення помилок повторите пуск програми на контрольному прикладі і зафіксуйте отриманий результат,
- виконаєте генерацію листинга програми, що включите в звіт про виконання роботи.

6. ЗМІСТ ЗВІТУ

Звіт про лабораторну роботу повинний містити:

- титульний лист;
- мета і задачі роботи;
- листинги з вихідними текстами й об'єктним кодом налагоджених програм з доповненнями, що забезпечують тестування і налагодження;
- перелік помилок, виявлених при налагодженні;
- результати рішення контрольних прикладів;
- висновки по роботі.

Навчально-методичне видання

ПРОГРАМНО-ТЕХНІЧНІ КОМПЛЕКСИ ТА ПРОМИСЛОВІ КОНТРОЛЕРИ.

Методичні вказівки до виконання практичних робіт для другого рівня вищої освіти зі спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» /Укл.: Трушаков Д.В., Федотова М.О. – Кропивницький: ЦНТУ, 2024 - 91 с.

Укладачі: Трушаков Д.В. – канд. техн. наук, доцент;
Федотова М.О. – канд. техн. наук, асистент.

Формат 210*297 1/16. Ум. друк. арк. 5,7.