Зінченко М. Д., Маначін І. О., Бурчак А. А.

МІКРОПРОЦЕСОРНА ТЕХНІКА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

М. Д. Зінченко, І. О. Маначин, А. А. Бурчак

Мікропроцесорна техніка

НАВЧАЛЬНИЙ ПОСІБНИК

ДНІПРО
2025

УДК 004.272.4(075.8)
З 63

Авторський колектив:
Зінченко М. Д., Маначин І. О., Бурчак А. А.

Рекомендовано Радою якості освітньої діяльності УДУНТ
Протокол № 4 від 24 грудня 2024 р.

З 63 **Зінченко, М. Д.** Мікропроцесорна техніка : навч. посіб. / М. Д. Зінченко, І. О. Маначин, А. А. Бурчак ; за ред. канд. техн. наук М. О. Рибальченко ; Укр. держ. ун-т науки і технологій. – Електрон. вид. – Дніпро : УДУНТ, 2025. – 209 с.

ISBN 978-617-8314-26-2 (PDF)

У навчальному посібнику розглянуто системи числення, наведено загальні принципи побудови мікропроцесорів та мікроконтролерів. Розглянуто особливості архітектури і можливості мікропроцесорів і однокристальних мікроконтролерів з RISC-архітектурою. Наведені приклади складання програм.

Призначений для опанування освітнього компонента «Мікропроцесорна техніка» освітньо-професійної програми «Комп'ютеризовані системи управління та робототехніка» підготовки бакалаврів за спеціальністю 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка».

Іл. 104, табл. 65, бібліогр. 11 назв.

УДК 004.272.4(075.8)

ISBN 978-617-8314-26-2 (PDF)
DOI 10.15802/978-617-8314-26-2

ЗМІСТ

ВСТУП

Невід'ємною складовою підготовки фахівців з автоматизації та комп'ютерно-інтегрованих технологій є опанування здобувачами вищої освіти основ побудови та функціонування мікропроцесорної техніки та її застосування в системах автоматизації, набуття навичок проектування та налаштування побудованих на базі мікроконтролерів технічних засобів автоматизації, а також розробки їх програмного забезпечення на мові Асемблер.

Сучасні управляючі обчислювальні комплекси використовують різні мікропроцесорні набори, які мають спільні принципи побудови та часові діаграми роботи. Серед цих комплексів значне місце посідають мікропроцесорні пристрої на базі 8-розрядних процесорів.

Незважаючи на те, що обчислювальні потужності сучасних процесорів значно перевершують 8-розрядні процесори, останні продовжують широко застосовуватись для реалізації мікропроцесорних приладів, регуляторів та простих і дешевих контролерів, що управляють нескладними об'єктами управління, які не вимагають високої швидкодії і мають невелику кількість вхідних та вихідних змінних. До того ж розгляд 8-розрядних процесорів дозволяє в наочній і доступній формі розглянути побудову мікропроцесора та з'ясувати технологію обробки даних в ньому.

Даний навчальний посібник призначений для студентів, які здобувають ступінь бакалавра в Українському державному університеті науки і технологій за спеціальністю 174 – автоматизація, комп'ютерно-інтегровані технології та робототехніка на освітній програмі «Комп'ютеризовані системи управляння та робототехніка». Матеріал посібника охоплює лекційний матеріал обов'язкової для вивчення професійної навчальної дисципліни «Мікропроцесорна техніка», забезпечуючи, зокрема, формування таких визначених стандартом вищої освіти програмних компетентностей, як:

- здатність застосовувати знання фізики, електротехніки, електроніки і мікропроцесорної техніки, в обсязі, необхідному для розуміння процесів в системах автоматизації та комп'ютерно-інтегрованих технологіях.

- здатність обґрунтовувати вибір технічних засобів автоматизації на основі розуміння принципів їх роботи, аналізу їх властивостей, призначення і технічних характеристик з урахуванням вимог до системи автоматизації і експлуатаційних умов; налагоджувати технічні засоби автоматизації та системи керування.

- здатність обґрунтовувати вибір технічної структури та вміти розробляти прикладне програмне забезпечення для мікропроцесорних систем керування на базі локальних засобів автоматизації, промислових логічних контролерів та програмованих логічних матриць і сигнальних процесорів.

- здатність розробляти системи керування роботизованими комплексами на основі сенсорів технологічних параметрів, систем технічного зору, мікропроцесорних керуючих засобів із застосуванням сучасних технологій програмування.

Відповідно до робочої програми дисципліни «Мікропроцесорна техніка» засвоєння навчального матеріалу посібника забезпечує досягнення здобувачами вищої освіти таких очікуваних програмних результатів навчання:

1) розуміння будови мікропроцесорів та способу їх функціонування, уміння застосовувати різні системи числення;

2) уміння користуватись засобами розробки програмного забезпечення та системою команд мікроконтролерів, базуючись на розумінні їх будови та усвідомленні їх відмінності від мікропроцесорів;

3) уміння розробляти програми арифметичних операцій та перетворення кодів для мікроконтролерів на мові Асемблер;

4) уміння застосовувати таймери та порти для введення/виведення сигналів мікроконтролерів та розробляти відповідне програмне забезпечення на мові Асемблер;

5) уміння конфігурувати та налаштовувати мікроконтролери, розробляти програми опитування клавіатури і виведення даних на цифрові індикатори;

6) уміння розробляти програмне забезпечення для налаштування та опитування аналого-цифрових перетворювачів;

7) уміння застосовувати послідовні інтерфейси мікроЕОМ, аналізувати їх роботу та виявляти помилки послідовної передачі даних

Навчальний матеріал посібника міститься у дев’ятьох розділах.

У першому розділі розглядаються системи числення, які застосовуються в обчислювальній техніці, правила арифметичних операцій з двійковими числами.

Другий розділ має допоміжний характер. Присвячений схемотехніці ЕОМ, певним чином повторюючи відомості дисципліни «Електроніка», він

сприяє кращому розумінню матеріалу, що викладається у наступних розділах посібника.

У третьому розділі розглядаються базові поняття мікропроцесорної техніки: організація пам’яті, системи команд та архітектури мікропроцесора, що сприяє формуванню у здобувачів освіти розуміння будови мікропроцесорів та способу їх функціонування.

У четвертому розділі розглянуто будову та функціонування мікроконтролерів, що дозволяє усвідомити їх відмінності від мікропроцесорів і користуватись засобами розробки програмного забезпечення та відповідною системою команд.

Матеріал п’ятого розділу містить відомості про особливості розробки програм арифметичних операцій та перетворення кодів для мікроконтролерів на мові Асемблер.

Особливості застосування таймерів та портів для введення/виведення сигналів мікроконтролерів, а також розробка відповідного програмного забезпечення на мові Асемблер розглянуті у шостому розділі.

Сьомий розділ посібника містить матеріал щодо конфігурування та налаштовування мікроконтролерів, розробки програм опитування клавіатури і виведення даних на цифрові індикатори.

Особливості розробки програмного забезпечення для налаштування та опитування аналого-цифрових перетворювачів розглянуті у восьмому розділі посібника.

У завершальному дев’ятому розділі розглянуті зовнішні послідовні інтерфейси мікроЕОМ. Матеріал цього розділу формує у здобувачів уміння аналізувати роботу цих інтерфейсів та виявляти помилки послідовної передачі даних.

Викладення матеріалу супроводжується значною кількістю прикладів, що полегшує його сприйняття та засвоєння. Подано перелік запитань для самоконтролю.

1. ІСТОРІЯ РОЗВИТКУ ЕОМ ТА СИСТЕМИ ЧИСЛЕННЯ

1.1. Історія розвитку ЕОМ

Перші механічні обчислювальні пристрої, рахівниці, з'явилися ще в античному світі. За 1000 років до нашої ери з'явилися рахівниці в Китаї і Японії, рахівниці мали форму намист, укріплених на спеціальній рамі. Намиста називалися «калькулями».

Наступним етапом розвитку обчислювальних пристроїв є створення Блезом Паскалем в 1647 р. арифметичної машини, яка використовувала принцип коліс від 0 до 9. У 1666 р. С. Морланд створив пристрій, що підсумовував і віднімав, а в 1673 р. був виготовлений калькулятор Морланда з множенням чисел.

У тому ж таки XVII ст. Готфрид Лейбниць реалізував обчислювальний пристрій для множення і ділення чисел.

За отця обчислювальної техніки вважається Ч. Беббідж. Він у 1821 р. створив спеціалізовані машини для обчислення різниць між числами, а на початку 30-х років XIX ст. приступив до створення універсальної обчислювальної машини, оскільки розумів обмеженість різницевих машин. Він першим дійшов висновку, що комп'ютер повинен мати 5 основних блоків:

- пристрій введення числової інформації (спочатку це були перфокарти Жакара для ткацьких верстатів);
- пристрій пам'ять для зберігання чисел та програм (перфокарта);
- арифметичний пристрій;
- пристрій управління для контролю над ходом виконання програми;
- пристрій виведення результатів.

У 1890 р. американський вчений Г. Холеріт винайшов пристрій для аналізу результатів перепису населення США, який базувався на використанні перфокарт з отворами, через які замикався контакт. На основі реалізації даного винаходу надалі утворилася відома фірма IBM (International Business Machines).

Подальший розвиток обчислювальної техніки стимулювався потребами науки і техніки, які вимагали усе більш значного об'єму розрахункових операцій, які до того ж необхідно було виконувати швидко. Ці завдання були

пов'язані з вирішенням управління, а саме управління вогнем зенітної артилерії, розшифровкою радіограм тощо. У цей же час почала зароджуватись наука про управління живими й неживими об'єктами – кібернетика.

Засновником кібернетики вважається американський вчений Норберт Вінер. Він разом з мексиканським професором медицини проаналізував патології поведінки людини при різних захворюваннях нервової системи і сформулював поняття зворотного зв'язку, яке стало основоположним в теорії управління.

На початку 40-х років минулого століття, коли стало можливим використовувати електронні лампи як елементи схем цифрових машин, подальший розвиток обчислювальної техніки пішов у напрямі використання підсилювальних елементів, що працюють у ключовому режимі з двома станами «так» та «ні».

Засоби обчислювальної техніки розвивалися одночасно в декількох країнах: у США, Великій Британії, Німеччині тощо.

У Німеччині в 1935 р. К. Цузе, чия компанія пізніше влилася в компанію SIEMENS, створює ЕОМ Z1 і патентує її. У 1941 р. він створює ЕОМ Z3 – двійковий програмований комп'ютер, у 1945 р. – першу мову програмування високого рівня.

У США в 1943 р. в Гарварді була створена ЕОМ MARK-1, у Великій Британії – Colossus-1 – для розшифровування повідомлень німецьких передавачів.

У 1946 р. Дж. Маклі та Дж. Еккерт (Пенсильванський університет) створили ЕОМ ENIAC (Electronic Numerical Integrator and Calculator) – для швидкого обчислення балістичних таблиць для знарядь і ракет, яка складалась з 18000 ламп, мала об'єм пам'яті – 20 десятирозрядних чисел та споживала 150 кВт електроенергії.

У 1945 р. Дж. Фон Нейман приступив до проектування машини EDVAC (Electronic Discrete Variable Automatic Computer). У проєкті цієї ЕОМ була реалізована концепція, за якою програма, що управляє, зберігається в пам'яті машини, а також ідея, за якою команди можна обробляти так само, як іншу інформацією, тобто програму можна модифікувати в ході її виконання.

У 1947 р. в Кембріджі була створена ЕОМ EDSAC (Electronic Delay Storage Automatic Computer), для якої була вперше створена операційна система.

У 1958 р. з'явилися промислові зразки транзисторів, які замінили лампи, що дозволило значно зменшити габарити обчислювальних машин, скоротити енергоспоживання, підвищити їх швидкодію. Це призвело до створення машин другого покоління.

У 1965 р. були випущені інтегральні схеми, які замінили окремі компоненти ЕОМ, що призвело до подальшого скорочення габаритів та енергоспоживання. Підвищилась надійність роботи електронних машин, які були вже машинами третього покоління. Ці ЕОМ почали застосовуватись для вирішення управління технологічними процесами.

У 1959 р. розробники фірми Datapoint (США) розробили перший мікропроцесор і запропонували фірмі-виробнику Intel реалізувати його. Але характеристики цього процесора за швидкодією виявились гіршими, ніж очікувалося, і фірма Datapoint відмовилася від продовження подальших досліджень. Проте, фірма Intel продовжила дослідження, оскільки вже понесла витрати на розробку, й домоглася необхідних результатів. На світовому ринку з'явився перший мікропроцесор Intel 8008.

Перші вітчизняні ЕОМ були створені під керівництвом академіків С. А. Лебедева, Ю. А. Базільовського та І. С. Брука. Це малі машини «Наїрі», «Мир». а також унікальна за швидкодією машина – БЕСМ-6.

1.2. Системи числення

Розвиток обчислювальної техніки привів до розширення використання різних систем числення. Зазвичай використовується десяткова система числення. Проте її застосування для реалізації обчислювальних процесів за допомогою електронних пристроїв пов'язане із труднощами розпізнавання десяти різних рівнів сигналів, які мають відповідати різним цифрам. Тому в обчислювальній техніці для реалізації обчислень і представлення чисел та кодів команд використовують двійкову систему числення, яка має лише дві цифри «0» та «1», для реалізації яких в технічних пристроях необхідні лише два рівні сигналів: «високий» та «низький», тобто необхідно розрізняти лише дві події: чи сигнал наявний, чи він відсутній.

Система числення визначає правило представлення числа з комбінації цифр даної системи числення. У десятковій системі числення, якою ми користуємося повсякденно, використовується десять цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), комбінація яких визначає число. При представленні числа з цифр має

значення, місце, на якому знаходиться цифра. Місце визначає вагу даної цифри. Одна й та цифра може визначати кількість одиниць, десятків, сотень тощо. Усе залежить від її місця розташування в числі.

Такі системи числення називаються позиційними системами числення за основою [1,2]. У цих системах використовується кінцевий набір символів, кожен символ називається цифрою і позначає деяку кількість. Число різних цифр в наборі називається основою системи числення. Щоб отримати яке-небудь число, необхідно цифри записати поряд. Кожній позиції цифри в числі ставиться у відповідність ваговий множник (коефіцієнт).

Представлення числа в десятковій системі числення можна уявити у вигляді полінома. Наприклад, десяткове число «678» матиме вигляд: $6\times10^2+7\times10^1+8\times10^0$.

Тут цифра «6» входить з вагою 100, цифра «7» –з вагою 10, цифра «8» – з вагою 1.

У загальному вигляді даний вираз можна представити таким чином

$$N=a_{n-1}a_{n-2}...a_1a_0=a_{n-1}b^{n-1}+a_{n-2}b^{n-2}+...+a_1b^1+a_0b^0 \qquad (1.1)$$

де $a_{n-1}...a_0$ - цифри числа; $b^{n-1}...b^0$ – вагові коефіцієнти.

У таблиці 1.1 наведені найбільш вживані системи числення та їх характеристики.

Таблиця 1.1

Системи числення

Основа	Система числення	Позначення	Цифрові символи
2	Двійкова	b	0,1
3	Трійкова		0,1,2
4	Четверткова		0,1,2,3
5	П'ятіркова		0,1,2,3,4
8	Вісімкова	o	0,1,2,3,4,5,6,7
10	Десяткова	d	0,1,2,3,4,5,6,7,8,9
12	Дванадцяткова		0,1,2,3,4,5,6,7,8,9,A,B
16	Шістнадцяткова	h	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Найбільше застосування в обчислювальній техніці, разом з десятковою системою числення, отримали двійкова, вісімкова та шістнадцяткова системи. Для систем числення застосовують позначення:

- $100_{(10)}$, $100_{(d)}$ – число представлене у десятковій системі числення;
- $100_{(2)}$, $100_{(b)}$ – число представлене у двійковій системі числення;
- $100_{(8)}$, $100_{(o)}$ – число представлене у вісімковій системі числення;
- $100_{(16)}$, $100_{(h)}$ – число представлене в шістнадцятковій системі числення.

1.2.1. Двійкова система числення

Двійкова система числення використовує дві цифри «0» та «1», що дозволяє використовувати цю систему числення у технічних пристроях для виконання арифметичних операцій [2]. Дані технічні пристрої мають два стійкі стани: «увімкнено – вимкнено», або «низький – високий» рівень.

У двійковій системі числення кожній позиції відповідає певна вага, яка визначається як ступінь числа 2, оскільки основа двійкової системи числення дорівнює 2. Через те, що дана система числення має дві цифри, розрядність двійкових чисел є значно більшою розрядності десяткових чисел.

Представлення двійкових чисел та їх десяткових чисел здійснюється за виразом (1.1) у такий, наприклад, спосіб

$$101101_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 =$$

$$= 1\times32 + 0\times16 + 1\times8 + 1\times4 + 0\times2 + 1\times1 = 45_{10}$$

У запису двійкового числа кожна позиція зайнята двійковою цифрою, яка називається бітом. Слово «біт» штучне, воно з'явилось як скорочення від двох слів: «*binary digit*» (двійкова одиниця) – *bit*.

Під час розгляду двійкових чисел користуються поняттями найменший значущий біт (наймолодший двійковий розряд) та найбільший значущий біт (найстарший двійковий розряд). Звичайне двійкове число записується так, що найбільший значущий біт єкрайнім зліва.

чисел з десяткової системи числення до двійкової здійснюється шляхом багатократного ділення десяткового числа на «2». Розглянемо, наприклад, десяткового числа «35» у двійкове число:

$35 : 2 = 17$ + залишок $1 = a_0$

$17 : 2 = 8$ + залишок $1 = a_1$

$8 : 2 = 4$ + залишок $0 = a_2$

$4 : 2 = 2$ + залишок $0 = a_3$

$2 : 2 = 1$ + залишок $0 = a_4$

$1 = a_5$

Таким чином, десяткове число «35» у двійній формі запису матиме вигляд $a_5a_4a_3a_2a_1a_0$ – 100011.

Переконаймося у цьому перевіркою:

$$100011_2 = 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 =$$
$$= 1\times32 + 0\times16 + 0\times8 + 0\times4 + 1\times2 + 1\times1 = 35_{10}.$$

1.2.2. Шістнадцяткова система числення

У шістнадцятковій системі числення використовуються 16 цифр:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Шістнадцяткова система числення використовується для скороченого запису 4-х розрядного двійкового числа. У таблиці 1.2 наведені шістнадцяткові числа та їх двійкові і десяткові еквіваленти.

Таблиця 1.2

Системи числення

Шістнадцяткове число	Двійкове число	Десяткове число	Шістнадцяткове число	Двійкове число	Десяткове число
0	0000	0	10	10000	16
1	0001	1	11	10001	17
2	0010	2	12	10010	18
3	0011	3	13	10011	19
4	0100	4	14	10100	20
5	0101	5	15	10101	21
6	0110	6	16	10110	22
7	0111	7	17	10111	23
8	1000	8	18	11000	24
9	1001	9	19	11001	25
A	1010	10	1A	11010	26
B	1011	11	1B	11011	27
C	1100	12	1C	11100	28
D	1101	13	1D	11101	29
E	1110	14	1E	11110	30
F	1111	15	1F	11111	31

Перетворення двійкового числа на шістнадцякове полягає в тому, що біти, починаючи з молодшого значущого біта, об'єднуються в групи по чотири. Кожній групі привласнюється відповідний шістнадцятковий символ. Наприклад, щоб представити двійкове число 101010111111101_2 як

шістнадцяткове, необхідно зліва додати два незначущі нулі з метою формування чотирьох повних тетрад (груп з чотирьох бітів):

0010 1010 1111 1101.

Замінивши кожну тетраду відповідним шістнадцятковим символом, отримаємо число $2AFD_{16}$.

Дана форма запису набагато простіше і сприймається легше, ніж двійкова.

Слід пам'ятати, що шістнадцяткові числа – це спосіб представлення двійкових чисел, якими оперує мікропроцесор.

Представлення шістнадцяткового числа як двійкового також здійснюється за виразом (1.1):

$$2AFD_{16} = 2\times16^3 + A\times16^2 + F\times16^1 + D\times16^0 =$$
$$= 2\times4096 + 10\times256 + 15^*\times16 + 13\times1 =$$
$$8192 + 2560 + 240 + 13 = 11005_{10}$$

$$0010\ 1010\ 1111\ 1101_2 = 1\times2^{13} + 0\times2^{12} + 1\times2^{11} + 0\times2^{10} + 1\times2^9 + 0\times2^8 +$$
$$+ 1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 =$$
$$= 8192 + 0 + 2048 + 0 + 512 + 0 + 128 + 64 + 32 + 16 +$$
$$+ 8 + 4 + 0 + 1 = 11005_{10}$$

1.3. Арифметичні операції в двійковій системі числення

Чотири основні арифметичні операції, а саме додавання, віднімання, множення та ділення можна виконувати в позиційній системі числення з будь-якою основою.

1.3.1. Додавання двійкових чисел

a	b	a + b
0	0	0
1	0	1
0	1	1
1	1	10
1+1+1		11

Правила додавання двійкових та десяткових чисел аналогічні, але в результаті швидшого заповнення розрядів у двійковій системі числення під

час додавання двійкових чисел швидше відбувається й перенесення до старшого розряду.

При додаванні в результаті переповнювання сусіднього, молодшого розряду відбувається збільшення на одиницю в старшому розряду, що називають перенесенням одиниці до старшого розряду при переповненні молодшого розряду.

Додавання двійкових чисел виконується за тими ж правилами, що й додавання десяткових чисел. Порядок додавання двійкових чисел наведений на рис.1.1 на прикладі додавання 1101010 до 1101100.

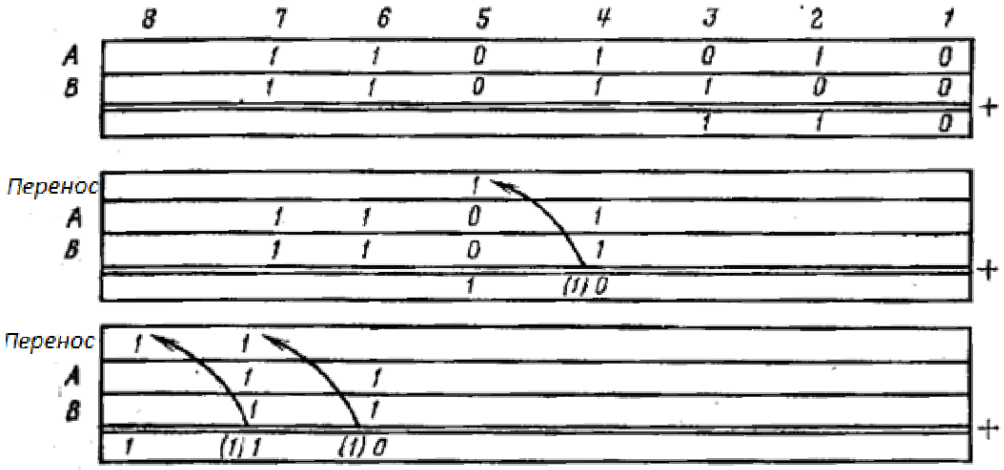


Рис.1.1. Схема додавання двійкових чисел 1101010 та 1101100 [2]

У першому молодшому розряді доданками наявні «0» та «0». Результат їх додавання – «0».

У другому розряді до «1» додається «0». Результат додавання – «1».

У третьому розряді до «0» додається «1». Результат додавання – «1».

У четвертому розряді результатом додавання «1» до «1» є 10_2.

Одиницю перенесення записуємо над п'ятим розрядом, в якому підсумовування «1», «0» та «0» дає в результаті «1».

У шостому розряді знову підсумовуються «1» та «1». Результат додавання – 10_2.

Аналогічним чином одиницю перенесення записуємо над сьомим розрядом, в якому тепер необхідно скласти три одиниці. Результатом є 11_2.

Одиницю перенесення розташовуємо над восьмим розрядом, який порожній для обох доданків, тому в результаті додавання у восьмому розряді з'явиться «1».

1.3.2. Віднімання двійкових чисел

Зрозуміти механізм віднімання двійкових чисел легко на прикладі віднімання десяткових чисел. Наприклад, віднімемо 17283 (від'ємник) від числа 909009 (зменшуваного) (див. рис. 1.2). Віднімання починають з наймолодшого розряду. Віднімаючи 3 від 9, отримуємо 6.

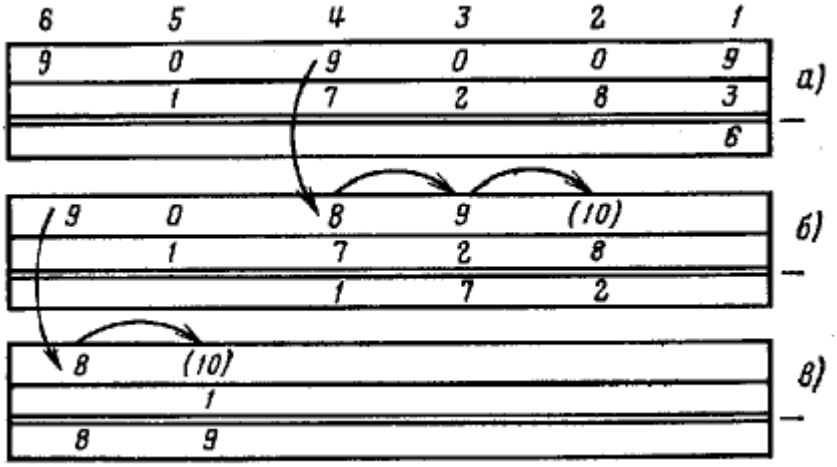


Рис.1.2. Схема віднімання двох десяткових чисел [2]

У наступному розряді необхідно відняти 8 від 0, що безпосередньо зробити неможливо через те, що 8 > 0. Щоб здійснити подальші обчислення необхідно звернутися до розрядів, що розташовані зліва, для знаходження числа, яке не дорівнює 0. В даному випадку таким числом буде 9. П 1 з 9, внаслідок чого в четвертому розряді замість 9 з'являється 8, у третьому розряді замість 0 з'являється 9, а в другому – 10.

Тепер в другому розряді від 10 можна відняти 8, отримавши 2. У третьому розряді віднімаючи 2 від 9, отримуємо 7. У четвертому розряді віднімаючи 7 від 8, отримуємо 1. У п'ятому розряді необхідно відняти 1 від 0. Для цього необхідно знову п 1, рухаючись вліво до тих пір, доки не дійдемо до першого ненульового розряду. У ньому 9 заміниться на 8, а замість 0 в п'ятому розряді отримуємо 10. Тепер обчислення можна продовжити.

Для двійкових чисел процес обчислення різниці наведений на рис.1.3.

Результат обчислення в молодшому розряді «1»–«1»=«0». У другому розряді відняти «1» від «0» просто так не вдасться, й тому проглядаємо розряди справа наліво до тих пір, доки не знайдемо «1». Додаємо «1» до «0» в другому розряді, що дає 10_2 , якому відповідає десяткове число 2_{10}. Нулі, які стоять між двома розрядами, перетворюються на одиниці. У другому

розряді віднімаємо від $10_2 - 1_2$ й отримуємо 1_2. У третьому та четвертому розрядах результат віднімання буде дорівнювати «0». У п'ятому розряді знову від «0» потрібно відняти «1», для чого знову виконується позичання, як в описаному вище випадку.

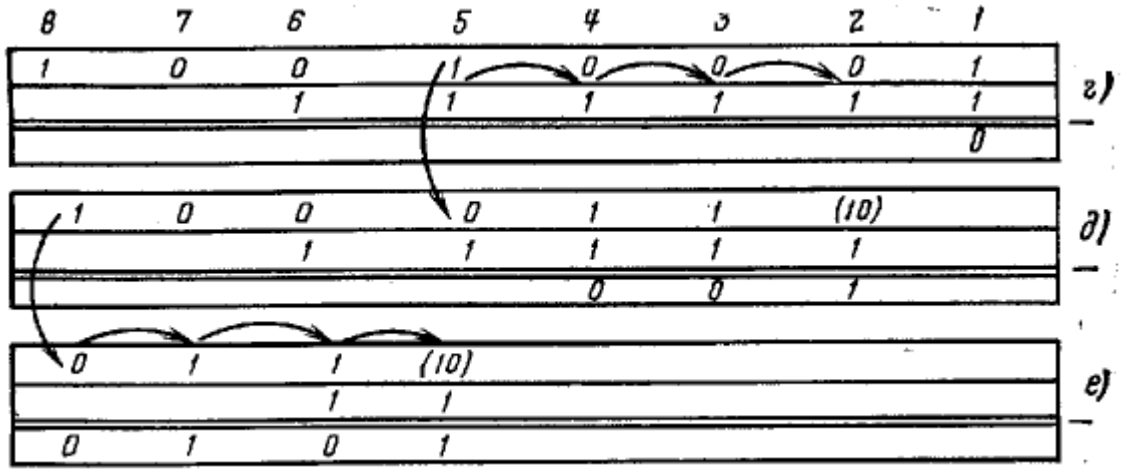


Рис.1.3. Схема віднімання двох двійкових чисел [2]

Якщо зменшуване більше за від'ємник, то різниця виходить додатною, якщо навпаки, то результат буде від’ємним.

1.3.3. Множення двійкових чисел

Правила двійкового множення:

$$0\times0=0;$$
$$0\times1=0;$$
$$1\times0=0;$$
$$1\times1=1.$$

Множення двох двійкових чисел здійснюється так само, як і множення десяткових чисел. Розглянемо приклад множення чисел $9_{10}=1001_2$ та $3_{10}=11_2$:

$$\begin{array}{r} 1001 \\ \times\quad 11 \\ \hline +\ 1001 \\ \quad 1001 \\ \hline 11011 \end{array}$$

Перевірка дає

$$11011_2= 1\times16+1\times8+0\times4+1\times2+1\times1= 27_{10}\,.$$

1.4. Представлення від'ємних чисел у двійковій системі числення

Оскільки обчислювальна машина оперує з цифрами «0» та «1», знаки «+» та «–», що позначають додатні та від'ємні числа, у двійковій системі відсутні. Для представлення додатних та від'ємних чисел у двійковій системі числення та в обчислювальній техніці використовують старший значущий розряд двійкового числа. Якщо він дорівнює «1», число вважається від'ємним, якщо «0» – додатним [2].

Така форма представлення чисел виходить, якщо двійкове число представити таким чином:

$$b_7[-(2)^7] + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0$$

Таке представлення називають додатковим кодом двійкового числа. У цьому випадку старший значущий розряд має вагу не 2^7, а -2^7 . Якщо в цьому розряді стоїть «1», він представлятиме від'ємне десяткове число -128_{10} , оскільки найбільше число, що міститься в розрядах, не може перевищувати десяткове значення 127_{10}.

Таким чином, двійкове число 10110001 можна представити як

$$1\times(-128)+0\times64+1\times32+1\times16+0\times8+0\times4+0\times2+1\times1= -79_{10} .$$

Якщо в старшому значущому розряді стоїть «0», подібний запис є додатним числом, адже у решти розрядів вага є позитивною.

Наприклад,

$$01001000 = 0\times(-128)+1\times64+0\times32+0\times16+1\times8+0\times4+0\times2+0\times1= 72_{10}.$$

Для представлення від'ємного десяткового числа у двійковій формі необхідно спочатку це число записати як позитивне у двійковому вигляді; записати зворотний код цього двійкового числа, а потім додати до записаного зворотного коду одиницю, внаслідок чого отримаємо додатковий код двійкового числа, який дорівнюватиме від'ємному десятковому числу.

Наприклад, представимо у двійковому вигляді від'ємне десяткове число -77_{10}. Для цього отримаємо двійкове число додатного числа 77_{10}

$$77_{10} = 01001101_2.$$

Зворотний код цього числа отримаємо заміною усіх «0» на «1», та усіх «1» на «0»:

$$01001101_2 \rightarrow 10110010_2$$

Для отримання додаткового коду додамо до зворотного коду «1».

$$\begin{array}{rl} + & 10110010 \quad \text{- зворотний код} \\ & \underline{\qquad\quad 1} \\ & 10110011 \quad \text{- додатковий код числа } 77_{10} \end{array}$$

Перевірка:

$$101100112 = 1\times(-128)+0\times64+1\times32+1\times16+0\times8+0\times4+1\times2+1\times1= -7710$$

У такий спосіб можна представити числа в діапазоні від

$$10000000_2 = -128_{10} \text{ до } 01111111_2 = +127_{10}.$$

Якщо цей діапазон недостатній, необхідно використовувати 16-розрядні числа, які утворюються шляхом об'єднання двох 8-розрядних слів.

Використання додаткових кодів двійкових чисел значно спрощує операцію двійкового віднімання, адже віднімання можна замінити додаванням додатного та від'ємного чисел. Наприклад,

$$17_{10} - 22_{10} = 17_{10} + (-22_{10})$$

У двійковому вигляді це означає, що потрібно скласти $00010001_2 = 17_{10}$ та додатковий код числа $00010110_2 = 22_{10}$. Додатковий код числа 00010110_2 отримаємо, перетворивши його в зворотний код 11101001_2 і додавши до зворотного коду «1»: 11101010_2 .

Після цього додаємо

$$\begin{array}{rll} + & 00010001 & \text{- } 17_{10} \\ & \underline{11101010} & \text{- } -22_{10} \\ & 11111011 & \end{array}$$

У десятковому представленні

$$111110112 = 1\times*(-128)+1\times64+1\times32+1\times16+1\times8+0\times4+1\times2+1\times1= -5_{10}$$

1.5. Системи кодування

Щоб мати можливість оперувати не тільки з числами, але й символами, а отже представляти в ЕОМ текстову інформацію, використовують системи кодування, в яких кожному символу ставлять у відповідність певний двійковий код, тобто певну комбінацію нулів та одиниць.

У мікроЕОМ найчастіше використовують два коди: американський стандартний код для обміну інформацією ASCII (American Standard Code for

Information Interchange) та розширений двійково-кодований EBCDIC (Extended Binary Coded Decimal Interchange).

Окрім цих кодів використовуються також коди КОІ-7, КОІ-8 тощо.

Контрольні запитання та завдання

1. Надайте визначення системі числення.
2. Чому розглянуті системи числення називаються позиційними?
3. Що називається основою системи числення?
4. Що означає біт у двійковій системі числення?
5. Що означає тетрада у двійковій системі числення?
6. Для якої системи числення застосовується ідентифікатор *b*?
7. Як і навіщо отримують обернений код двійкового числа?
8. Як утворюється і для чого використовується додатковий код двійкового числа?
9. Яким чином представляються від'ємні числа в двійковій системі числення?
10. Для чого застосовуються системи кодування?

2. СХЕМОТЕХНІКА ЕОМ

2.1. Логічні елементи

ЕОМ складається не з центрального процесора (мікропроцесора). Він є головним, але не єдиним з багатьох складових частин, що утворюють ЕОМ та входять до її складу.

До найбільш поширених складових (елементів) ЕОМ відносяться логічні елементи, що реалізують логічні функції: AND, OR, NOT, AND-NOT, OR-NOT, виключне OR; тригери; регістри; лічильники тощо.

Зазначені елементи, які можуть бути представлені окремими інтегральними мікросхемами, є цифровими елементами. На їх входах можуть з'являтися лише два характерні рівні напруги: низький (близько 0 В) та високий (наприклад, 5 В), причому низькому та високому рівням напруги відповідають двійкові значення «0» та «1» відповідно.

2.1.1 Логічний елемент AND

Умовне графічне зображення логічного елементу AND наведене на рис.2.1.

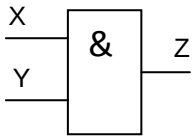


Рис. 2.1. Графічне зображення логічного елементу AND

Оскільки це цифровий елемент, на входи X та Y подається вхідна напруга, яка може набувати тільки фіксованих значень, прийнятих для даного типу мікросхеми, які визначаються розробником мікросхеми. Для мікросхем типу ТТЛ це рівні напруги 0 В або 5 В. Залежно від стану входів на виході Z елементу також з'являтиметься один з цих рівнів напруги. Стан виходу визначається таблицею 2.1 (таблицею істинності елементу).

Таблиця 2.1

Таблиця істинності логічного елементу AND

X	*Y*	*Z*
0	0	0
0	1	0
1	0	0
1	1	1

Таким чином, робота логічного елементу AND описується наступним правилом: вихідний сигнал дорівнює «1» тільки за умови, коли всі вхідні сигнали дорівнюють «1» (X=1 та Y=1). Це правило аналітично записується у вигляді

$$Z = X \cdot Y \qquad (2.1)$$

Даний запис запозичений з алгебри Буля, яку можна розглядати як алгебру систем, побудованихна логічних елементах.

Арифметико-логічний пристрій ЕОМ містить вісім двоходових логічних елементів AND, що дозволяє мікроЕОМ виконувати операцію над двома 8-бітовими словами A та B.

2.1.2. Логічний елемент OR

Умовне графічне зображення логічного елементу OR наведене на рис.2.2.

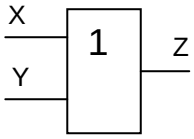


Рис. 2.2. Графічне зображення логічного елементу OR

Вихідний сигнал логічного елементу OR дорівнює «1», якщо принаймні один з вхідних сигналів дорівнює «1» (див. табл. 2.2). Аналітично це записується таким чином

$$Z = X + Y \tag{2.2}$$

Символ «+» позначає операцію OR. Таблиця істинності логічного

Таблиця 2.3

Таблиця істинності логічного елементу АБО

X	*Y*	*Z*
0	0	0
0	1	1
1	0	1
1	1	1

2.1.3. Логічний елемент NOT або інвертор

Вихідний сигнал інвертора протилежний за значенням вхідному сигналу. Інакше кажучи, сигнали на вході та на виході інвертора завжди не збігаються: якщо на вході «0», на виході «1»; якщо на вході «1», на виході - «0».

На рис. 2.3 наведене умовне графічне зображення інвертора. Кружечок на виході інвертора означає виконання операції NOT.

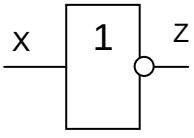


Рис. 2.3. Графічне зображення логічного елементу NOT

Операція інверсії аналітично записується у вигляді

$$Z = \overline{X} \qquad (2.3)$$

Горизонтальна смужка над змінною читається як «ні».

2.1.4. Логічний елемент AND-NOT

Логічний елемент AND-NOT відрізняється від логічного елементу AND додатковою операцією заперечення. Умовне графічне зображення елементу подібне до елементу AND з додаванням кружечка на виході (рис.2.4).

Згідно з алгеброю Буля операція AND-NOT аналітично записується таким чином

$$Z = \overline{X * Y} \qquad (2.4)$$

Над змінними X та Y спочатку здійснюється операція AND, результат якої інвертується згідно з таблицею істинності (табл. 2.4).

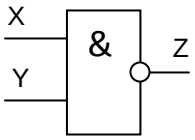


Рис. 2.4. Графічне зображення логічного елементу AND-NOT

Таблиця 2.4

Таблиця істинності логічного елементу AND-NOT

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

2.1.5. Логічний елемент OR-NOT

У логічному елементі OR-NOT вихідний сигнал дорівнює «1», якщо усі вхідні сигнали дорівнюють «0». Умовне графічне зображення наведене на рис. 2.5.

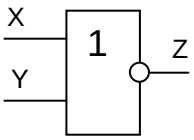


Рис. 2.5. Графічне зображення логічного елементу OR-NOT

У логічному елементі OR-NOT суміщене виконання двох операцій: OR та NOT, при цьому спочатку виконується операція OR, а потім здійснюється інвертування (див. табл. 2.5).

Таблиця 2.5

Таблиця істинності логічного елементу OR-NOT

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Аналітично операція OR-NOT записується таким чином

$$Z = \overline{X + Y} \tag{2.5}$$

2.1.6. Логічний елемент ВИКЛЮЧНЕ OR

У логічному елементі ВИКЛЮЧНЕ OR вихідний сигнал дорівнює «1» тільки тоді, коли один з вхідних сигналів дорівнює «1», а інший – «0» (табл. 2.6).

Таблиця 2.6

Таблиця істинності логічного елементу ВИКЛЮЧНЕ OR

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Умовне графічне зображення приведене на рис.2.6.

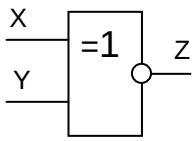


Рис. 2.6. Графічне зображення логічного елементу ВИКЛЮЧНЕ OR

2.2. Тригери та регістри

Тригер є логічною схемою з двома виходами, значення сигналів на яких протилежні один одному. Якщо на одному з виходів присутня «1», на

іншому виході – «0» і навпаки. Виходи тригера позначають Q та $\overline{Q}$. - За значенням $\overline{Q}$ є протилежним Q (читається як "не Q ").

Тригер може мати також додаткові входи (не менше двох): вхід встановлення, що позначається як S (*set* – встановлення), та вхід скидання, що позначається R (*reset* – скидання). Поява імпульсу на вході S переводить тригер у стан $Q=1$. Такий процес називають встановленням тригера у стан «1», а вхід S – входом встановлення тригера у стан «1». Залежно від схемних особливостей тригеравстановлення у стан «1» відбувається після подання на цей вхід сигналу «1», або сигналу «0».

Вхід R використовують для переведення тригера у стан $Q=0$. Цей процес називається встановленням тригера у стан «0», або скиданням. Відповідний вхід називають входом встановлення тригера у стан «0», або входом скидання. Такий тригер називається *RS*-тригером.

Умовне графічне зображення *RS*-тригера наведене на рис. 2.7.

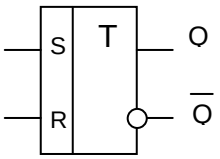


Рис. 2.7. Умовне графічне позначення RS-тригера|

Найважливішим з усіх можливих застосуваньтригера є використання його як пристрою, що запам’ятовує, який ще називається заскочкою. Наприклад, елементи статичної напівпровідникової пам'яті будуються на тригерах, причому для запам'ятовування в комірці пам'яті слова завдовжки 8 біт потрібні 8 тригерів.

Іншим прикладом використання тригера є його застосування як дільника частоти вхідних сигналів на два.

Окрім *RS*-тригерів в мікро-ЕОМ також використовуються *D*-тригери та двотактні *JK*-тригери.

2.2.1. *D*-тригер

D-тригер використовується як елемент пам’яті, двотактний *JK*-тригер може виконувати як функцію елементу пам’яті, так і ділення частоти на два. Обидва ці тригери мають вхід синхронізації, сигнал на який подається від генератора тактових сигналів. Момент передачі встановлення сигналів на виході тригера відносно моменту появи сигналу на вході синхронізації

визначається типом тригера: *D*-тригер реагує на появу фронту, а *JK*-тригер – на появу зрізу сигналу синхронізації. Фронт та зріз сигналу наведені на рис. 2.8.

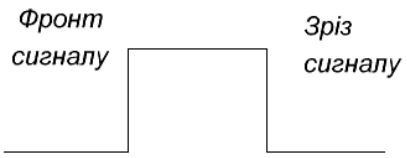


Рис. 2.8. До визначення фронту та зрізу сигналу синхронізації тригера

Умовне графічне зображення *D*-тригера наведене на рис. 2.9.

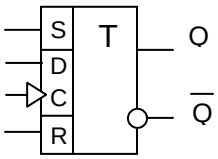


Рис.2.9. Умовне графічне зображення D-тригера

Для D-тригера обов'язковою є наявність двох входів: *C* (синхронізації) та *D* (*delay* – затримки). Зазвичай наявні також входи встановлення *S* та скидання *R*. На рис. 2.10 наведена часова діаграма роботи D-тригера.

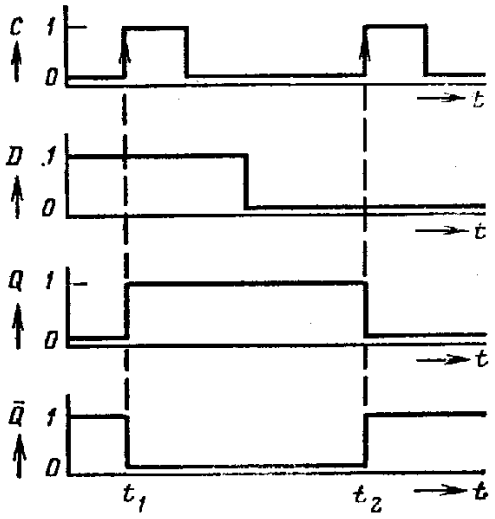


Рис. 2.10. Часова діаграма роботи D-тригера [2]

Сигнал, що з'являється на вході, передається на вихід при зміні сигналу *C* синхронізації з нуля в одиницю. Значення сигналу на виході зберігається

навіть при подальших змінах сигналів на вході, що й дозволяє використовувати тригер для запам'ятовування інформації. У момент часу, коли сигнал синхронізації *C* знову набуває значення «1», нове значення сигналу з входу передається на вихід та запам'ятовується.

2.2.2. Двотактний *JK*-тригер

Умовне графічне зображення двотактного *JK*-тригера наведене на рис. 2.11.

Тригер має три входи *J* та три входи *K*, хоч в загальному випадку їх може бути й інша кількість. Входи *J* та *K* призначені для подачі на них сигналів, що управляють, входи *S* та *R* використовуються для встановлення та скидання. Тригер вважається за готовий до роботи, якщо *S*=1 та *R*=1 (в цьому стані входи *S* та *R* не впливають на його функціонування).

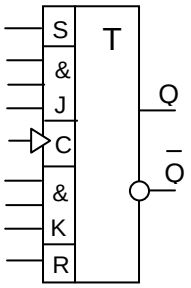


Рис. 2.11. Умовне графічне зображення JK-тригера

Входи *J* та *K* заводяться на наявні в тригері логічні елементи AND. У тригері виконуються операції

$$J = J_1 \cdot J_2 \cdot J_3$$

$$K = K_1 \cdot K_2 \cdot K_3$$

Момент часу, що безпосередньо передує зміненню сигналу синхронізації з «0» в «1», позначений t_n, а момент часу, що безпосередньо наступає за зворотним переходом сигналу синхронізації, позначений t_{n+1} (рис. 2.12).

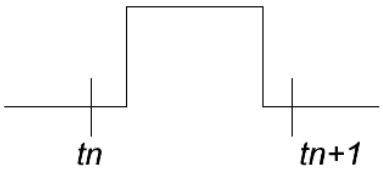


Рис. 2.12. Часова діаграма змінення сигналу синхронізації

Роботу *JK*-тригера пояснює таблиця 2.7, яка називається таблицею переходів *JK*-тригера.

Таблиця 2.7

Таблиця переходів JK-тригера|

Стани входів та виходів			
На момент t_n		На момент t_{n+1}	
$J = J_1 J_2 J_3$	$K = K_1 K_2 K_3$	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	Q_n	$\overline{Q_n}$
0	1	0	1
1	0	1	0
1	1	$\overline{Q_n}$	Q_n

Якщо *J*=0 та *K*=0, стан виходів в моменти часу t_n та t_{n+1} однакові (перший рядок таблиці).

Якщо *J*=0, то сигнал *K*=1 передається за зрізом сигналу *C* синхронізації на виходи та переводить їх у стани $Q = 0$ та $\overline{Q} = 1$ (другий рядок).

Якщо *J*=1, то сигнал *K*=0 передається за зрізом сигналу *C* синхронізації на виходи та переводить їх в стан $Q = 1$ та $\overline{Q} = 0$ (третій рядок).

Тригер в другому та третьому випадках працює як елемент, що запам'ятовує.

Якщо *J*=1 та *K*=1, то стан виходів змінюється на протилежний по завершенні дії синхросигналу, тобто $Q_{n+1} = \overline{Q_n}$. Інакше кажучи, стан виходу *Q* у моменти часу t_n та t_{n+1} протилежні – вихід *Q* перемикається. У цьому режимі тригер працює як схема ділення частоти вхідного синхросигналу на два, якщо на входах *J* та *K* присутні одиничні сигнали.

2.2.3. Регістри

Регістри призначаються для запам'ятовування двійкових чисел. Регістри є сукупністю тригерів, що призначені для зберігання двійкового числа певної довжини. Оскільки тригер може у двох станах, він здатний зберігати двійкові значення «0» чи «1». Якщо ж потрібно запам'ятати двійкове число, складене з певного набору нулів одиниць, має застосовуватись відповідна кількість тригерів. Для запам'ятовування 8-розрядного числа потрібні вісім тригерів: по одному на кожен біт. На рис. 2.13 наведений регістр, що здатен зберігати 4-бітові слова, причому

тригер *T4* використовується для запам'ятовування розряду з найбільшою вагою. Оскільки даний регістр може зберігати слова довжиною не більше 4 біт, у ньому можуть бути записані десяткові числа від 0 до 15, двійкове представлення яких вимагає саме не більше чотирьох розрядів.

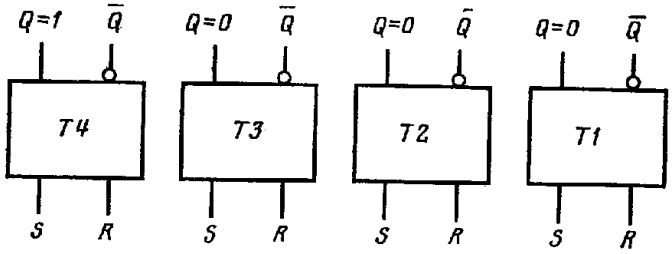


Рис. 2.13. Чотирирозрядний паралельний регістр

Встановлення тригерів регістра у стан «0» або «1» відповідно до значень розрядів вхідного слова називають прийомом інформації до регістру. Прийом слова до регістру можна виконати, подаючи відповідні сигнали на входи встановлення *S* та скидання *R* тригерів.

Мікропроцесор в основному містить такі регістри, що зберігають слова завдовжки 8 бітів.

2.2.4. Зсувні регістри

За значимістю для ЕОМ зсувні регістри поступаються тільки лічильникам. Зсувний регістр – це регістр, в якому можна здійснювати зсування слова на необхідну кількість розрядів.

У зсувному регістрі інформація, що зберігається в тригерах, зсувається вправо або вліво при появі сигналу синхронізації. Окрім цьогорегістри використовуються для виконання операцій множення та ділення.

Залежно від способу прийому інформації зсувні регістри можна розділити на регістри з послідовним входом та паралельним входом. На рис. 2.15 наведений регістр з послідовним прийомом інформації.

До даного регістру код слова передається послідовно. У початковому стані всі тригери скинуті в «0».

Спочатку на вхід регістра (вхід тригера *T1*) надходить сигнал «1». Після появи синхросигналу:

- «0» з виходу тригера *T2* передається на тригер *T3*;
- «0» з виходу тригера *T1* передається на тригер *T2*;
- «1» з входу регістра передається на вихід тригера *T1*.

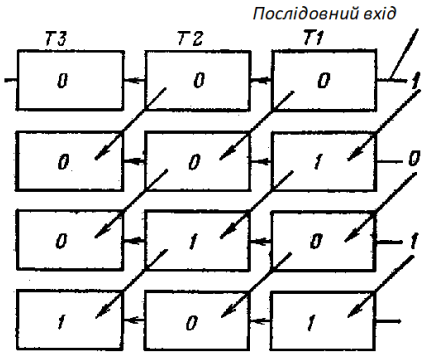


Рис. 2.15. Схема послідовного регістра

Далі на вхід регістра подається «0». Синхросигнал, що з'являється одразу після цього, здійснює:

- передачу «0» з виходу тригера *T2* на тригер *T3*;
- передачу «1» з виходу тригера *T1* на тригер *T2*;
- передачу «0» зі входу регістра на вихід тригера *T1*.

Далі на вхід регістра подається наступна «1». Знову відбувається її передача на вихід тригера *T1* і наступний біт даних зсувається на один розряд вліво. Таким чином, відбувся прийом 3-бітового слова до зсувного регістр шляхом послідовної передачі його розрядів на вхід регістра. Для цього необхідно було подати на регістр три синхроімпульси.

На рис. 2.16 наведений регістр з паралельним прийомом інформації або паралельний регістр.

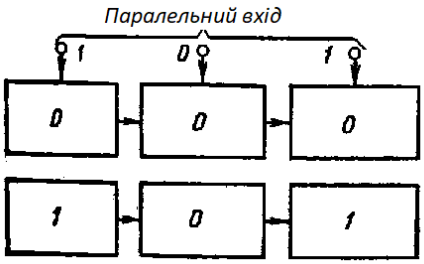


Рис. 2.16. Схема паралельного регістра

На входи усіх тригерів регістра інформація одночасно. При паралельному способі прийому інформації довжина (загальна кількість біт) слова, що на вхід регістра, має дорівнювати числу паралельних вхідних ліній, що утворюють шину.

Рис. 2.17 ілюструє, яким чином відбувається передача інформації зі зсувного регістра з послідовним виходом.

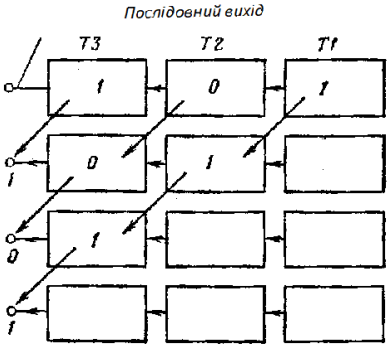


Рис. 2.17.Схема видачі інформації зі зсувного регістра

Передача даних здійснюється послідовно з виходу тригера *Т3*. Під час надходження кожного синхросигналу дані зсуваються на один розряд вліво. Після надходження першого синхросигналу інформація, що зберігалася у тригері *Т2*, передається на вихід регістра. Таким чином, розряди слова, що зберігалося в зсувному регістрі, послідовно, один за одним, «виштовхуються» з регістра.

Для регістрів паралельним виходом (рис. 2.18) з після появи синхросигналу інформація передається з тригерів на виходи регістра. Зсувні регістри використовуються для множення та ділення двійкових чисел.

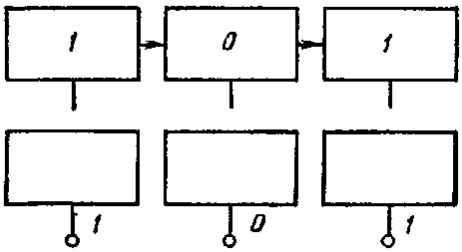


Рис.2.18. Схема видачі інформації з паралельного регістра

2.3. Компаратори, драйвери та тристабільні схеми

Компаратором називається пристрій, що призначається для порівняння двох аналогових або цифрових сигналів та формування вихідного дискретного сигналу «1», якщо значення сигналів збігаються. У протилежному випадку на виході компаратора формується «0».

На рис. 2.19 наведена схема порівняння сигналів компаратором.

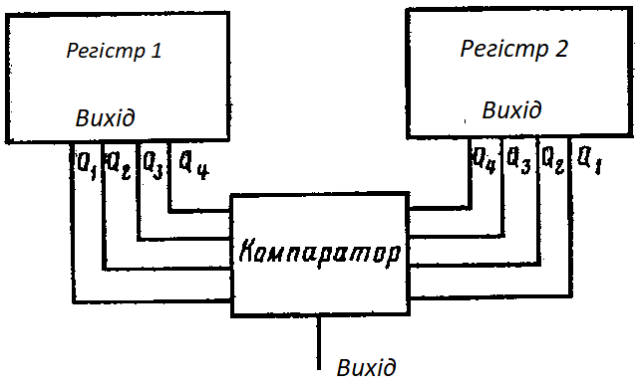


Рис. 2.19. Схема порівняння двох сигналів

Дана схема здійснює порівняння вмісту двох регістрів. Якщо стан тригерівв обох регістрах однаковий, вміст двох регістрів збігається.

Драйвером називають схему, яка працює як підсилювач струму. Драйвери використовують для:

- управління лампами цифрової інформації або семисегментними індикаторами;
- з'єднання логічних схем двох різних систем, якщо вони мають різні рівні напруги;
- передачі цифрових даних по кабельних лініях.

Буферним регістром з трьома станами (рис. 2.20) називають пристрій, призначений для відключення однієї частини мікро-ЕОМ від іншої. Ця схема працює подібно до ключа. Якщо сигнал, що управляє (сигнал дозволу прийому) дорівнює «0», такого буферного регістру переходить у *стан високого імпедансу*. Даний стан подібний до розімкненого ключа, у ньому виходи буферного регістра нібито відключаються від електричних кіл, до яких вони приєднані. Якщо ж сигнал дозволу прийому дорівнює «1», виходи буферного регістра набувають тих самих значень, що й сигнали на входах регістра. Даний стан подібний до замкнутого ключабуферного регістра нібито підключаються до електричних кіл.

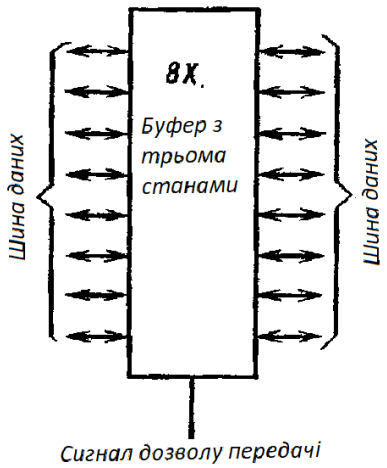


Рис. 2.20. Схема буферного регістра з трьома станами

Контрольні запитання та завдання

1. Назвіть типи логічних елементів.
2. Надайте визначення логічній схемі, яка називається тригер.
3. Які типи тригерів вам відомі? Для чого вони застосовуються в мікро-ЕОМ?
4. Чим відрізняються асинхронні схеми від синхронних?
5. Що являє собою регістр?
6. Які типи регістрів застосовують в обчислювальній техніці?
7. Як відбувається завантаження та вивантаження даних в послідовних регістрах?
8. В чому полягає призначення компараторів та драйверів?
9. Коли виникає потреба у застосування буферних регістрів з трьома станами?

3. БУДОВА ТА ФУНКЦІОНУВАННЯ МІКРОПРОЦЕСОРІВ

3.1. Пам'ять мікропроцесорних систем

3.1.1. Типи пам’яті. Постійна та оперативна пам’ять

Загальна пам'ять системи ЕОМ складається з пам'яті програм та пам'яті даних. Команди зберігаються в пам'яті програм, а дані для обробки – в пам'яті даних. Постійний запам’ятовувальний пристрій (ПЗП) використовується як пам'ять програм, а оперативний запам’ятовувальний пристрій (ОЗП) – пам'ять даних. Обидва типи пам'яті реалізуються на напівпровідникових елементах [1,2].

Інформація (найчастіше програма) зберігається у ПЗП. Її можна зчитувати і не можна змінювати або оновлювати. Існують три типи ПЗП:

- *read-only memory* (ROM) - пам'ять тільки для читання. Мікросхеми ROM програмуються під час їх виробництва. Виробник вводить до пам'яті інформацію відповідно до вимог користувача і подальше її оновлення є неможливим. Цей тип пам’яті використовується для серійного виробництва великої кількості однакових ПЗП;

- *programmable read-only memory* (PROM) – програмовані ПЗП, які на відміну від ROM споконвічно виробляються чистими, у той час як в ROM дані заносяться в процесі виробництва. Користувач може сам запрограмувати мікросхеми PROM за допомогою спеціального пристрою (програматора), який випалює у кристалі мікросхеми (у матриці) створені там діоди, що адресуються. Після такого програмування подальша зміна вмісту пам'яті стає неможливою;

- *erasable programmable read-only memory* (EPROM) – програмована пам'ять тільки для читання, що стирається. Це спеціальний тип PROM, який може очищатися з використанням ультрафіолетових променів. Після стирання EPROM може бути перепрограмована;

- *electrically erasable programmable read-only memory* (EEPROM) – програмована пам'ять тільки для читання, що електрично стирається. По суті EEPROM подібна до EPROM, але для стирання вимагає електричних сигналів. Перевага EEPROM у порівнянні з EPROM полягає в можливості проводити зміну вмісту пам'яті, не витягуючи мікросхему пам’яті з електронної плати, на якій вона встановлена.

Оперативний запам’ятовувальний пристрій ОЗП – *Random Access Memory* (RAM) – це пам'ять, з комірок якої процесор може зчитувати або до комірок якої може записувати інформацію. Її використовують для зберігання проміжних результатів обчислень та змінних. Якщо живлення мікро-ЕОМ припиняється вміст ОЗП зникає на відміну від ПЗП.

Залежно від принципу зберігання інформації розрізняють статичні та динамічні ОЗП.

За способом звернення напівпровідникову пам'ять можна розділити на два класи:

- пам’ять з довільним доступом;
- пам’ять з послідовним доступом.

У пам'яті з довільним доступом до елементів (комірок) пам'яті можна звертатися, визначаючи адреси довільно, і час зчитування з комірки не залежить від її адреси.

У послідовній пам'яті дані можна зчитувати тільки в тому порядку, в якому вони записувалися. Тому час звернення до послідовній пам'яті залежить від місця розташування комірки, в якій зберігаються дані. Зазвичай послідовна пам'ять застосовується як буферна пам'ять, наприклад, пам'ять на магнітній стрічці.

Пам'ять з довільним доступом може бути реалізована як за біполярною, так і за МОП-технологією. Біполярні мікросхеми застосовуються в основному в мікропрограмній пам'яті, що управляє. Вони мають високу швидкодію, але споживають більше енергії, мають малу щільність і високу вартість.

Пам'ять, як правило, складається з величезної кількості тригерних елементів. У кожному тригері зберігається 1 біт («0» або «1»). В енергозалежній пам'яті інформація, що зберігається, при виключенні напруги живлення втрачається. Це відбувається як в статичних, так і в динамічних ОЗУ. В енергонезалежній пам'яті інформація зберігається при виключенні напруги живлення. Прикладом такої пам’яті є ROM, PROM, EPROM та EEPROM.

Для вирішення проблеми енергонезалежності оперативної пам'яті при відключенні живлення використовується резервні джерела живлення, наприклад, резервне акумуляторне живлення (рис. 3.1). Це можливо тому, що

деякі мікросхеми МОП-пам’яті в режимі зберігання споживають набагато менше енергії у порівнянні з активним режимом.

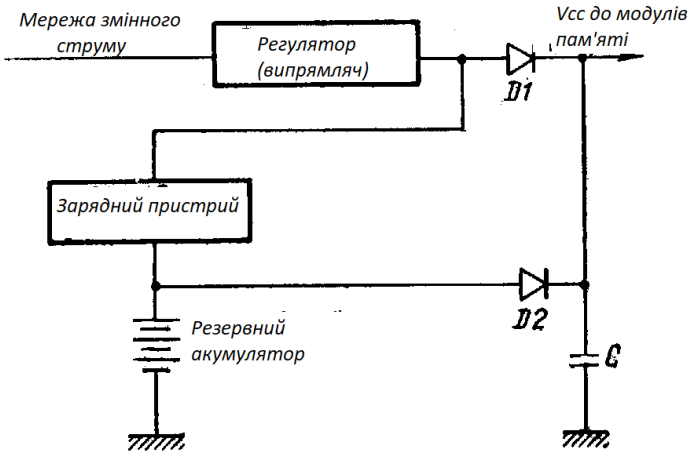


Рис. 3.1.Енергонезалежна система напівпровідникового ОЗП [3]

3.1.2. Будова елементів оперативної пам’яті. Статична та динамічна пам’ять

Оперативні пристрої, що запам'ятовують, можуть бути реалізовані у вигляді статичної або динамічної пам'яті.

Статична пам'ять. Для зберігання одного біта інформації необхідний один елемент, що запам’ятовує. У ***статичній пам'яті*** він, зазвичай, виконується на 6-ти МОП-транзисторах (рис. 3.2).

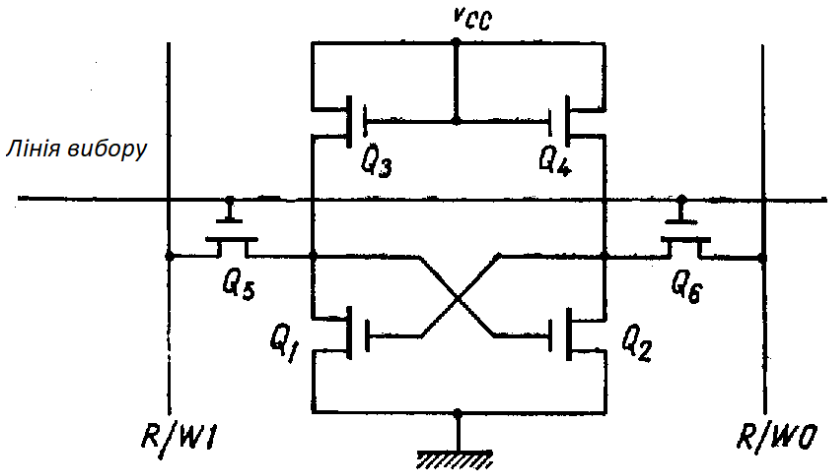


Рис. 3.2. Елемент статичної пам'яті [3]

Інформація, що зберігається, визначається станами транзисторів *Q1* та *Q2*: коли один з них увімкнений, інший - вимкнений й навпаки. Стану, коли *Q2* увімкнений, а *Q1* вимкнений, привласнюється значення «1», а протилежному стану – значення «0». Транзистори *Q3* та *Q4* прислуговують резисторами, а *Q5* та *Q6* працюють як вентилі дозволу. При запису спочатку обирається елемент за допомогою завдання високого рівня напруги на лінії вибору. Транзистори *Q5* та *Q6* вмикаються і лінія зчитування/запису *R/W1* підключається до затвору *Q2*, а лінія *R/W0* – до затвору *Q1*. Для запису до комірки одиниці на лінії *R/W1* встановлюється «1», а на лінії *R/W0* встановлюється «0»; при цьому *Q2* вмикається, а *Q1* – вимикається. Для запису до комірки нуля значення на лініях *R/W* змінюються на протилежні.

У будь-якому випадку після встановлення стани *Q1* та *Q2* не змінюються до наступної операції запису. Зчитування здійснюється подачею напруги на лінію вибору: стан *Q1* передається на лінію *R/W0*, а стан *Q2* – на лінію *R/W1*.

Загальну організацію статичної пам'яті можна описати за допомогою блок-схеми (рис. 3.3), де наведена пам'ять 1Кх1. Елементи, що запам'ятовують, організовані в матрицю з 32 рядків та 32 стовпців. Біти адреси А9-А0 розділені на адреси рядка та стовпця і визначають один з 1024 елементів, що запам'ятовують.

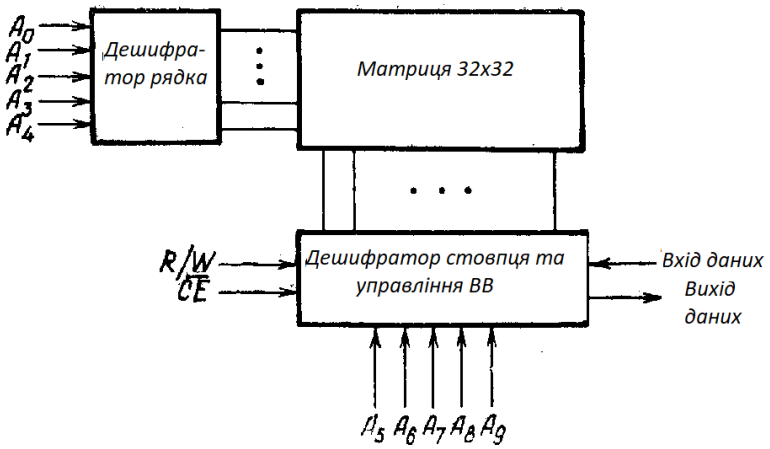


Рис. 3.3. Схема типового статичного ЗУПВ 1Кх1[3]

Дешифратор входів адреси рядків А4-А0 вибирає один з 32 рядків. Входи адреси стовпців А9-А5 не тільки вибирають стовпець, але й вирішують відповідні схеми ВВ, що містять формувачі та підсилювачі зчитування. Ці схеми забезпечують виведення біта, що зберігається, в операції зчитування та зміну його в операції запису.

Вхід управління *R/W* визначає тип операції (високий рівень – для зчитування, низький – для запису). Вхід дозволу кристалу $\overline{CE}$ вибирає відповідний набір БІС у модулі пам'яті, який містить більше слів, ніж ємність одного набору БІС. Якщо кристал «дозволений», відповідно до рівня на лінії *R/W* виконується операція зчитування або запису. Інакше сигнал *R/W* не сприймається й вихід переводиться у стан високого імпедансу. Це допускає пряме об'єднання виходів даних декількох БІС. Живлення БІС пам'яті здійснюється напругою +5 В.

Найбільш важливий параметром пам’яті є час звернення: часовий інтервал між стабілізацією входу і дійсними вихідними даними. Час звернення визначає швидкодію пам'яті.

Не менш важливим є час відновлення пам’яті – інтервал часу, необхідний для завершення внутрішніх операцій перед наступною операцією пам'яті. Сума часу звернення та часу відновлення утворює час циклу зчитування пам'яті.

Динамічна пам'ять має три основні властивості, які особливо виявляються в модулях пам'яті великої ємності:

1) висока щільність: елемент (комірку) динамічної пам'яті можна реалізувати на трьох і навіть на одному транзисторі, що дозволяє розмістити на кристалі більше елементів;

2) низьке споживання енергії: менше 0,05 мВт;

3) економічність: питома вартість динамічної пам'яті значно менша за статичну.

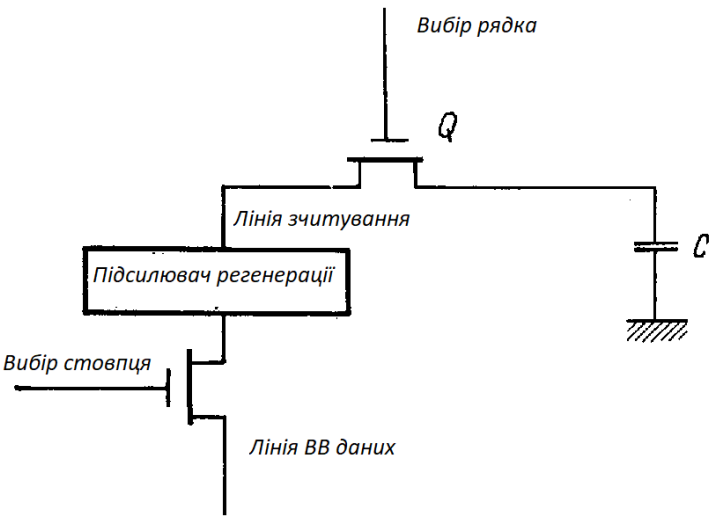


Рис.3.4.Типовий однотранзисторний елемент динамічної пам'яті [3]

Інтегральна мікросхема динамічної пам'яті організована в матрицю з рядків та стовпців комірок, що запам'ятовують. Елементарна комірка пам’яті складається з одного транзистора та однієї ємності. Зберігання «1» або «0» визначається наявністю або відсутністю заряду ємності. В операції зчитування на одній з ліній вибору рядка відповідно до молодших біт адреси (адреса рядка) встановлюється високий рівень електричного потенціалу. Він вмикає ключовий транзистор *Q* для усіх елементів обраного рядка, що запам'ятовують. При цьому підсилювач регенерації, що пов'язаний із кожним стовпцем, сприймає рівень напруги на відповідній ємності як «0» або «1». Адреса стовпця (старші біти адреси) визначає для виведення один з елементів у вибраному рядку. В процесі зчитування ємності у всьому рядку розряджаються. Щоб зберегти інформацію, підсилювачі регенерації здійснюють запис у той самий рядок елементів. Операції запису виконуються аналогічно, але у вибраному елементі запам'ятовуються вхідні дані, а решта елементів в цьому ж рядку просто регенерується.

Через розрядження ємності, що спричиняється струмом витоку через *pn*-перехід, Швидкість розрядження ємностей збільшується зізростанням температури, через що інтервал між регенерацією варіюється від 1 до 100 мс. Для збереження вмісту динамічної пам'яті здійснюється систематичне зчитування й запис вмісту комірок динамічної пам'яті. Цей процес називається *регенерацією* пам'яті. При температурі +70°С типовий інтервал регенерації становить 2 мс.

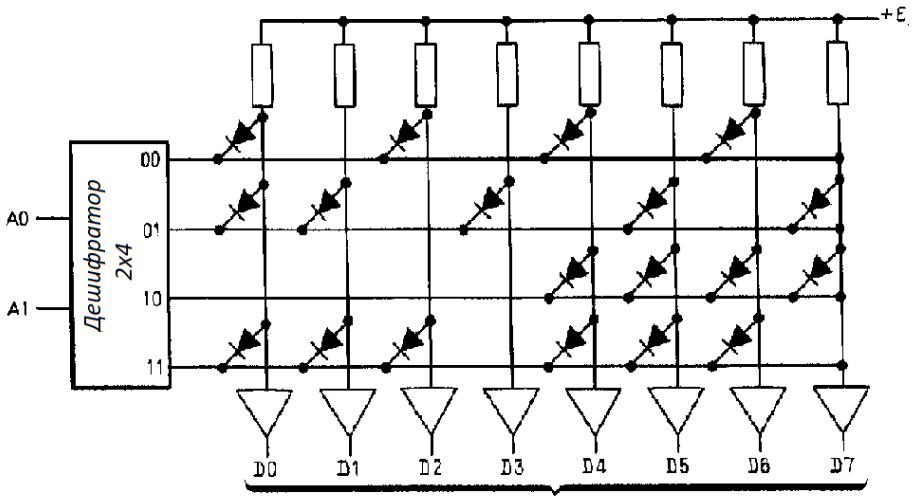
У кожному циклі регенерації динамічної пам'яті до інтегральної мікросхеми надходить адреса рядка і для вибираного рядка елементів ініціюється операція зчитування та запису.

Основний недоліки динамічної пам'яті випливає саме з необхідності її регенерації, адже до завершення циклу регенерації не може бути ініційована операція зчитування або запису даних до відповідної комірки пам’яті. Отже при попаданні на цикл регенерації час зчитування або запису збільшується. Недоліком динамічної пам’яті є також необхідність у використанні кількох номіналів напруги живлення: +12 В, +5 В та -5 В.

3.1.3. Будова постійних запам’ятовуючих пристроїв

Постійними пристроями, що запам'ятовують, є діодні матриці, які пропалюються виробником під час виготовленні мікросхеми (рис. 3.5) або користувачем під час програмування мікросхеми за допомогою

програматора. Під час програмування мікросхеми на відповідні її входи подається підвищена напруга живлення, внаслідок чого плавкі перемички розплавляються електричне коло між цими точками розривається (рис. 3.6).



Уміст ПЗП

Слово	*Двійкове представлення*								*Шістнадцяткове представлення*
	D0	D1	D2	D3	D4	D5	D6	D7	
00	0	1	0	1	0	1	0	1	55
01	0	0	1	0	1	0	1	0	2A
10	1	1	1	1	0	0	0	0	F0
11	0	0	0	1	0	0	0	1	11

Рис. 3.5. Матриця ПЗП [3]

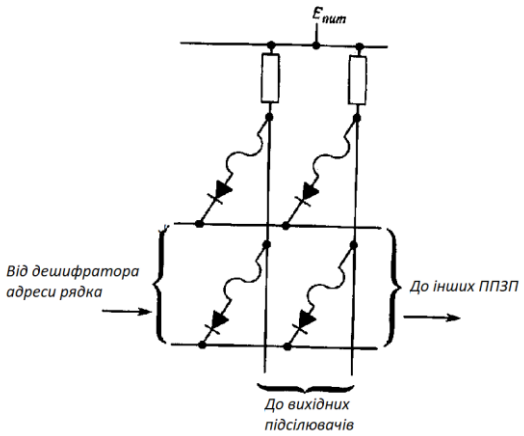


Рис. 3.6. Пристрій ППЗП [3]

Пам'ять програм і пам'ять даних не обов'язково мають бути розташовані окремо. Якщо елемент пам'яті містить команду, то ця комірка є частиною пам'яті програм, а якщо елемент пам'яті містить дані, то він є частиною пам'яті даних незалежно від того, де він фізично розташований.

Під час виконання програми команди та дані послідовно витягуються з пам'яті і передаються до центрального процесора. Кожен елемент (комірка) пам'яті має власну адресу. За кожною адресою в пам'яті знаходиться одне 8-розрядне слово. Адреса визначає місцеположення слова, а не біта. Вміст елементу пам'яті програміст може інтерпретувати по-різному:

1) як дані, що використовуються для виконання операцій:
   а) 8-розрядне число;
   б) частка числа, що має формат більше восьми розрядів;
   в) число, або інший символ відповідно до використовуваного коду (такого як ASCII), або
2) як команди:
   а) код операції;
   б) частка багатобайтної команди.

3.1.4. Машинні слова

Кількість різних команд, які може виконувати ЕОМ, частково визначається розрядністю машинних слів. Більшість мікро-ЕОМ має вісім розрядів для уявлення максимум 2^8 або 256 команд. Цього зазвичай достатньо. У такому випадку пам'ять організована таким чином, що елемент (комірка) пам'яті містить 8 бітів, а ЕОМ називається 8-розрядною з довжиною слова у 8 бітів. Слово з 8 біт називають *байтом*. Деякі мікро-ЕОМ працюють з довжиною слова у 4 або 16 бітів.

Ємність пам'яті, тобто максимальна кількість машинних слів, що можуть зберігатися в пам'яті, часто виражають в *кілобайтах* (Кбайт) або мегабайтах (Мбайт). Якщо пам'ять має ємність 1 Кбайт, то вона може зберігати 1024 8-розрядних слів. В обчислювальній техніці префікс "кіло" виражає не 1000, як в метричній системі, а найближчу до тисячі ступінь числа 2, яка дорівнює 2^{10}=1024. 1 Мбайт дорівнює 2^{20}= 1048576.

3.1.5. Модулі пам'яті

Окрім довжини слова важливою характеристикою ЕОМ є розрядність слів, що використовуються для адресації машинних слів. Якщо для адресації використовується 8 розрядів (1 байт), то мікро-ЕОМ може адресувати лише $2^8 = 256$ різних машинних слів, що є недостатнім для нормальної роботи ЕОМ. Тому використовують 16 розрядні та 8-розрядні ЕОМ, що мають 2-байтні адреси, щоб можна було адресувати $2^{16} = 65536$ слів. В цьому випадку

наймолодшою є адреса $0000000000000000_2 = 0000_{16}$, а найстаршою – адреса $1111111111111111_2 = FFFF_{16}$ (рис. 3.7).

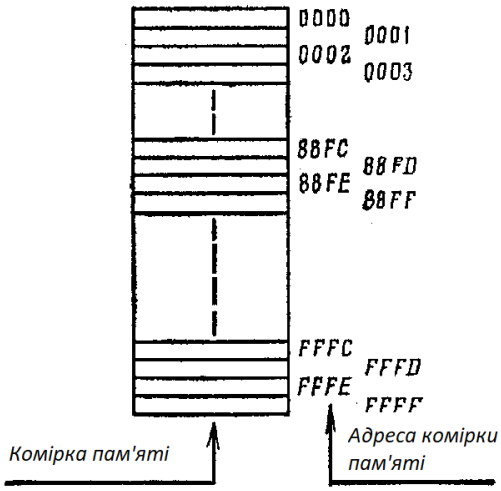


Рис. 3.7. Розташування адресного простору

Часто розряди одного машинного слова фізично розташовані в різних інтегральних мікросхемах. Інтегральні мікросхеми, які разом утворюють машинне слово, називають модулем пам'яті. Варіанти модулів пам'яті наведені на рис. 3.8. Один модуль пам'яті може складатися з 8-ми однобітової, 2-х чотирьохбітових, або 1-ї восьмибітової мікросхеми. Кількість бітів, які можуть зберігатися на одному кристалі, завжди є ступенем числа 2.

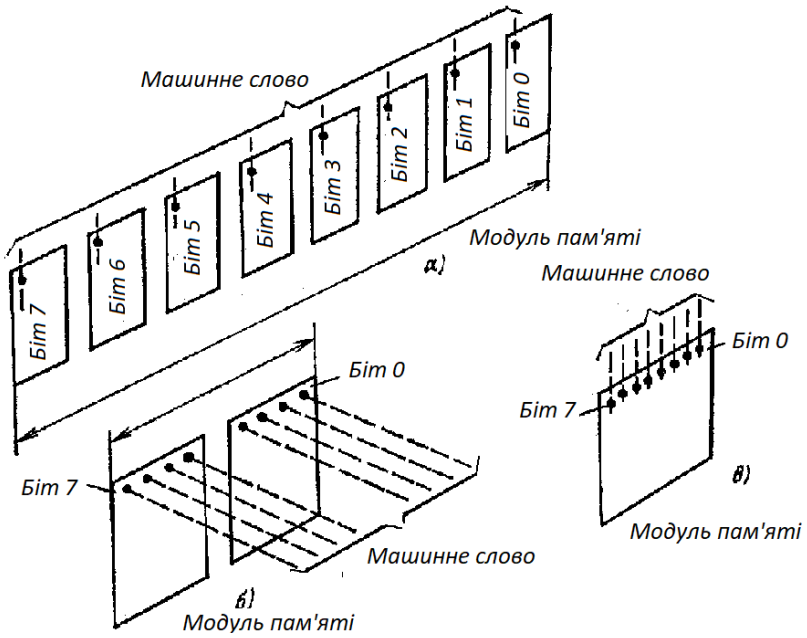


Рис. 3.8. Модуль пам'яті [2]

3.1.6. Мапа пам’яті. Адресація

Кількість розрядів адреси пам'яті залежить від місткості пам'яті, тобто від кількості машинних слів, що можуть зберігатись. Якщо, наприклад, місткість пам'яті становить 1 Кбайт = 1024 слів, то необхідно використовувати 10-розрядну адресу (2^{10}=1024).

Якщо основна пам'ять ЕОМ містить більше одного модуля пам'яті, частина коду адреси повинна вказувати, в якому модулі пам'яті розташоване дане слово. Ця частина адреси називається кодом вибору модуля або кодом вибору кристалу. Частина коду адреси, яка вибирає слово пам'яті модуля, називається адресою слова (рис. 3.9).

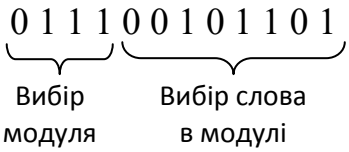


Рис. 3.9. Структура адреси

Декодування адреси слова здійснюється в самих кристалах пам'яті. Для декодування адреси модуля використовується окрема мікросхеми.

Від особливостей організації основної пам'яті залежить спосіб адресації, тобто які розряди адреси використовуються для вибору модуля, а які для вибору слова всередині модуля.

ЕОМ може містити різні модулі пам'яті, наприклад, модуль ОЗУ і модуль ПЗП, й ці модулі матимуть свої коди вибірки модуля (рис. 3.10).

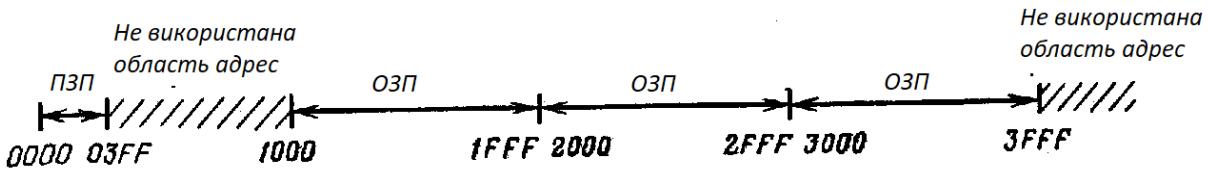


Рис. 3.10. Мапа пам'яті мікро-ЕОМ [2]

Приклад: Якщо пам'ять ЕОМ містить 4 модулі: один модуль ПЗП та три модулі ОЗП; місткість модуля ПЗП становить 1 Кбайт = 2^{10} слів; місткість кожного з модулів ОЗП становить 4 Кбайти = 2^{12} слів, адресація пам'яті має здійснюватись з використанням 16-розрядної шини.

Щоб вибрати слово в модулі ПЗП необхідно мати 10 розрядів для вибору слова і шість розрядів для вибору модуля. Якщо вибрати код 000000 як код вибору цього модуля ПЗП, то адреси слів в модулі ПЗП лежатимуть в межах від $0000\ 0000\ 0000\ 0000_2 = 0000_{16}$ до $0000\ 0011\ 1111\ 1111_2 = 03FF_{16}$.

Щоб вибрати слово в одному з модулів ОЗП, потрібно 12 розрядів для адреси слова $2^{12} = 4096 = 4К$. Для вибору модуля використовуються 4 розряди.

Якщо вибрати код 0001 для вибору першого модуля ОЗП, адреси слів в цьому модулі знаходитимуться в межах від

$0001\ 0000\ 0000\ 0000_2 = 1000_{16}$ до $0001\ 1111\ 1111\ 1111_2 = 1FFF_{16}$.

У другому модулі ОЗП, для вибору якого використовується код 0010, адреси знаходитимуться в межах від

$0010\ 0000\ 0000\ 0000_2 = 2000_{16}$ до $0010\ 1111\ 1111\ 1111_2 = 2FFF_{16}$.

У третьому модулі ОЗП, для вибору якого використовується код 0011, адреси знаходитимуться в межах від

$0011\ 0000\ 0000\ 0000_2 = 3000_{16}$ до $0011\ 1111\ 1111\ 1111_2 = 3FFF_{16}$.

На рис. 3.10 схематично наведені адреси усіх машинних слів. Така схема називається мапою пам'яті.

Машинне слово може інтерпретуватися різним чином. Це можуть бути дані (8-розрядне число, частина числа, символ) або команди.

8-розрядне число. Під час обчислень в ЕОМ машинне слово розміщується у так званому в арифметико-логічному пристрої (АЛП) і сукупність «0» та «1» являє собою в цьому випадку двійкове число.

Якщо число має розрядність більше 8, для його представлення використовується декілька машинних слів. Ця кількість слів є необмеженою і може бути доволі великою.

Під символами розуміються літери та символи: 26 прописних та 26 рядкових літер латинської абетки, 25 різноманітних символів, як то !, №, %, & тощо, 10 цифр від 0 до 9. Ці символи представляються сукупністю цифрових кодів. Коди привласнюються символам відповідно до кодових таблиць: ASCII (американський стандартний код обміну інформацією); EBCDIC (розширений десятковий двійково-кодований код обміну) тощо.

Усього є 87 різних знаків, для кодування яких потрібні 7 бітів. Вільний (8-й) біт використовується для здійснення контролю передачі даних і називається бітом паритету. Значення біта паритету (біта контролю)

встановлюється таким, щоб слово, включаючи контрольний біт, завжди містило парну (або непарну) кількість «1», що обумовлюється заздалегідь. При цьому говорять про контроль парності або контроль по непарності.

Контрольні запитання та завдання

1. Які типи пам’яті застосовуються в обчислювальній техніці?
2. Опишіть відмінності енергозалежної та енергонезалежної пам’яті.
3. Які типи енергонезалежної пам’яті вам відомі?
4. Яка логічна схема застосовується для реалізації статичної пам’яті?
5. Яким чином реалізується динамічна оперативна пам’ять?
6. Чому в комп’ютерах як оперативну застосовують динамічну пам’ять?
7. Що називають модулем пам’яті?
8. Для чого призначена мапа пам’яті комп’ютера?
9. Яким чином розрядність шини адреси пов’язана з ємністю модуля пам’яті?

3.2. Архітектура мікропроцесора

Під архітектурою мікропроцесора розуміють перелік основних блоків, з яких він складається, та опис зв'язків та взаємодії між ними. Архітектура мікропроцесора містить:

- структурну схему МП;
- програмну модель МП (опис функцій регістрів);
- інформацію про організацію пам'яті (ємність пам'яті та способи адресації);
- опис організації процедур введення-виведення.

3.2.1. Формат команд мікропроцесора

Мікропроцесор 8-розрядної мікро-ЕОМ складається в основному з регістрів та шин, здатних зберігати і передавати слова завдовжки 8 бітів. Регістри можуть містити результати попередньої операції, а також використовуватись для тимчасового зберігання даних протягом періоду виконання якої-небудь команди.

1. Команди можуть складатися з 1, 2 або 3 байт, які послідовно розташовуються в пам'яті (рис. 3.11).

Під час виконання попередньої команди лічильник команд містить значення, яке є адресою того слова пам'яті, за яким розташовується перший байт наступної призначеної для виконання команди.

2. Перший байт команди відводиться для запису коду операції (КОП). Код операції вказує, які дії мають бути виконані над даними, а також вид оброблюваних даних.

3. Таким чином, цикл виконання команди починається зі зчитування з пам'яті першого байта команди, який містить код операції.

4. Якщо відповідно до коду операції виявляється, що другий та третій байти команди разом утворюють адресу даних, призначених для обробки, то ці два байти мають бути переписані до центрального процесора (ЦП). Після цього ЦП знає, де слід шукати необхідні дані.

5. Якщо ж код операції безпосередньо вказує на місце розташування даних, то їх обробка може розпочинатися відразу після зчитування першого байта. Код операції може, наприклад, вказувати, що при обробці повинен піддатися другий байт коду команди.

6. Отже, після зчитування команди стає відомо, де розташовуються призначені для обробки дані, і яка над ними має виконуватись операція. Оскільки перший байт команди містить КОП, його передача з пам'яті до ЦП є частиною загального процесу зчитування команди. Якщо другий та третій байти містять адресу даних, то зчитування цих байтів з пам'яті до ЦП також є частиною загального процесу зчитування команди.

7. Виконання команди зводиться до виконання відповідних операцій над даними. Під час виконанні команди також може здійснюватись передача даних (наприклад, звід пристрою введення до ЦП).

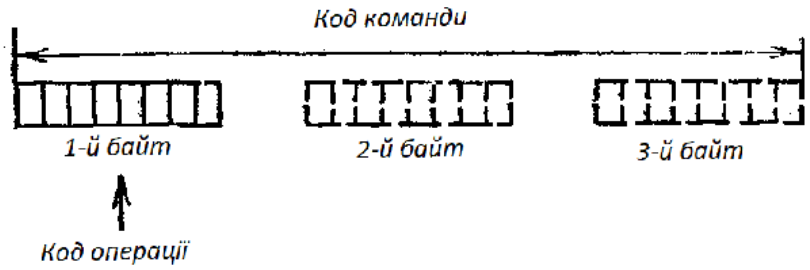


Рис. 3.11. Формат команди

3.2.2. Блок-схема 8-розрядного мікропроцесора Intel 8080

На рис. 3.12 наведена блок-схема 8-розрядного мікропроцесора Intel 8080.

Існують два основні типи архітектури – фон-нейманівська та гарвардська. Фон-нейманівську архітектуру запропонував в 1945 р. американський математик Джо фон Нейман. Особливість цієї архітектури полягає в тому, що програма та дані знаходяться в загальній пам'яті, доступ до якої здійснюється по одній шині даних та команд (рис. 3.12).

Гарвардська архітектура вперше була реалізована в 1944 р. Її особливістю є те, що пам'ять даних та пам'ять програм розділені і мають відокремлені шину даних і шину команд, що дозволяє підвищити швидкість мікропроцесорних систем.

Мікропроцесор містить у своєму складі арифметико-логічний пристрій (АЛП), пристрій управління, низку регістрів та внутрішню шину даних.

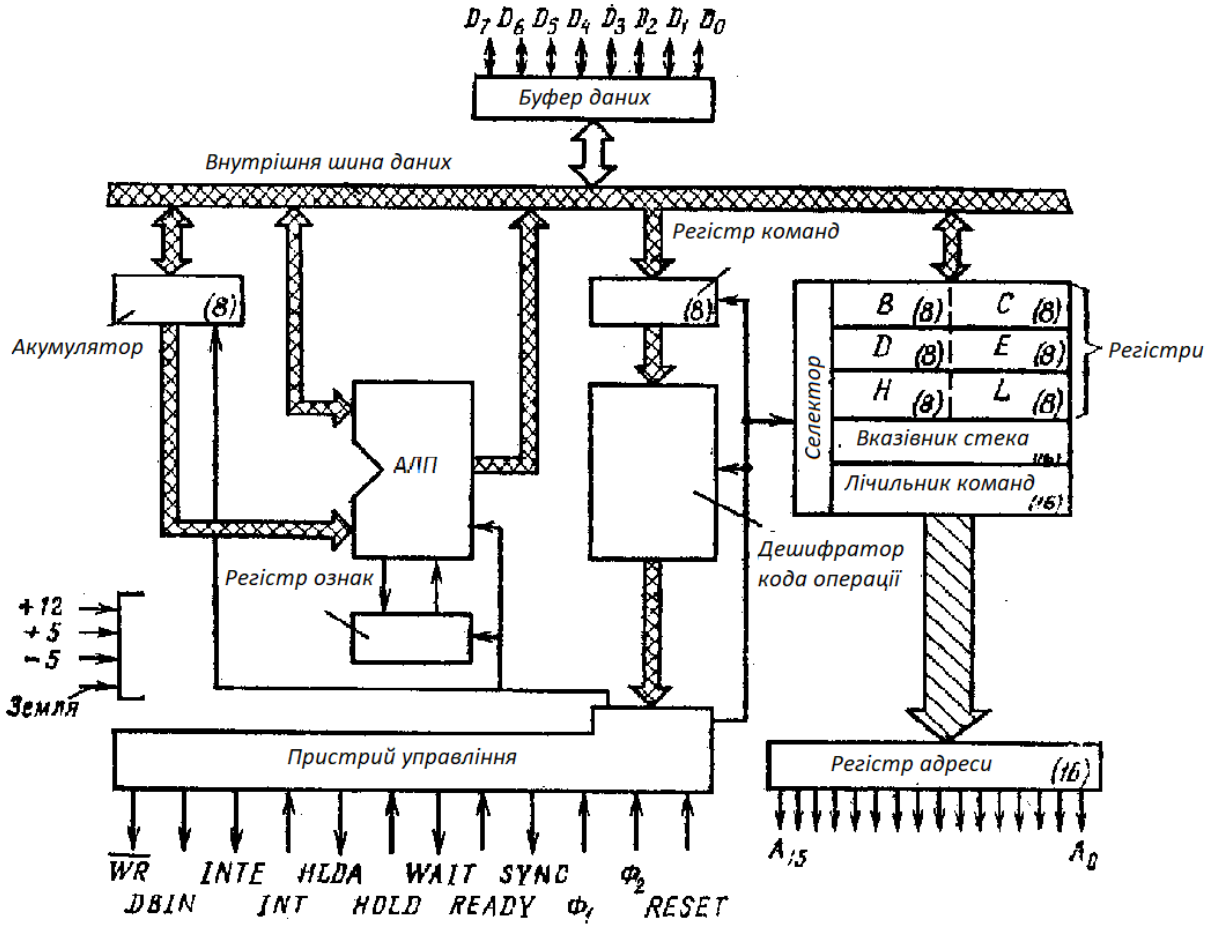


Рис. 3.12. Блок-схема 8-розрядного мікропроцесора Intel 8080 [2]

3.2.3. Аріфметико-логічний пристрій. Регістр акумулятор

Арифметико-логічний пристрій призначений для виконання арифметичних та логічних операцій. До арифметичних операцій відносяться, як правило, операції додавання і віднімання, інкременту та декременту. До логічних операцій відносяться логічні операції AND, OR, NOT та ВИКЛЮЧНЕ OR.

Дані, що призначені для обробки в АЛП (операнди) можуть до АЛП одночасно з різних місць:

- з акумулятора та регістрів загального призначення;
- з пам'яті через буфер шини даних.

Схема організації ЦП відповідає одноадресній машині, оскільки при цьому один з операндів завжди з акумулятора, інший передається або з одного з регістрів загального призначення, або з пам'яті. Таким чином, один з операндів завжди необхідно заздалегідь розмістити в акумуляторі. Результат обчислення розміщується в акумуляторі.

3.2.4. Регістри мікропроцесора

3.2.4.1. Регістр ознак

Виконання будь-якої операції може ставитися в залежність від значення результату виконання попередньої операції. Подібна ситуація виникає у тому випадку, коли при додаванні з'являється одиниця перенесення.

Щоб можна було звернутися до інформації про результати обчислень, АЛП з'єднується із спеціальним набором тригерів, які встановлюються у стан «1» або скидаються в «0» залежно від результату проведених обчислень. Кожен з тригерів зберігає якусь ознаку, а в сукупності ці тригери утворюють регістр ознак [2]. Схема бітів регістру ознак наведена на рис. 3.13.

Ознака перенесення	Ознака знаку	Ознака нуля	Ознака двійково-десяткового перенесення	Ознака переповнення	Ознака паритету

Рис. 3.13. Схема бітів регістра ознак [2]

Таким чином, регістр ознак містить інформацію про результат арифметичної або логічної операції після її виконання.

Ознака перенесення. Однією з найважливіших ознак є ознака перенесення. При додаванні в АЛП двох 8-розрядних чисел ця ознака показує, чи потрібно переносити одиницю в молодший значущий розряд

$$\begin{array}{r} 10111000 \\ +\ 11011010 \\ \hline 110010010 \end{array}$$

наступного байта. Як можна бачити, у наведеному нижче прикладі ознака перенесення дорівнює «1».

Ознака перенесеннятакож вказує, чи потрібно запозичувати одиницю при відніманні двох 8-розрядних чисел.

Якщо АЛП може виконувати операцію віднімання, то воно здійснюється відповідною командою. Якщо ж в АЛП команди віднімання немає, необхідно отримати додатковий код числа і потім здійснити додавання.

Ознака двійково-десяткового перенесення встановлюється у стан «1», якщо відбувається перенесення з розряду b3 до розряду b4. Ця ознака використовується при додаванні чисел, записаних у двійково-десятковому коді BCD. Незалежно від причини, що породжує перенесення з розряду b3 до розряду b4, необхідне застосування десяткової корекції. При будь-якій обробці чисел, записаних у коді BCD, програміст повинен враховувати у відповідних командах можливість використання ознаки двійково-десяткового перенесення. Якщо цього не робити, то ознака автоматично ігнорується.

Розглянемо приклад. При додаванні чисел 19 та 09, записаних в коді BCD, одиниця перенесення з розряду b3 до розряду b4 в процесі додавання використовується звичайним способом. Разом з тим, ця одиниця впливає на значення ознаки двійково-десяткового перенесення, яке встановлюється відповідним чином. Це означає, що необхідне застосування десяткової корекції, яка виконується автоматично відповідно до вказівок, що містяться в команді.

$$\begin{array}{l} 10011001\ \ (\text{BCD}) = 19 \\ +\ 00001001\ \ (\text{BCD}) = 09 \\ \hline 00100010\ \ (\text{BCD}) = 22, \text{ а має бути } 28 \\ +\ 00100010\ \ (\text{BCD}) = 22 \\ \ \ \ 00000110\ \ (\text{BCD}) = 06 - \text{корекція} \\ \hline 00111000\ \ (\text{BCD}) = 28 \end{array}$$

Ознаку нуля відзначає випадок появи в АЛП після виконання певної операції результату 00000000. Ця ознака використовується, наприклад, для організації циклів очікування.

Ознака знаку. Від'ємні числа представляють в обчислювальній машині у вигляді додаткових кодів. В цьому випадку старший значущий розряд може нести не тільки цифрове значення, але й знак числа: якщо в старшому значущому розряді стоїть «1», число є від'ємним, якщо «0», – додатне. Старший значущий розряд запам'ятовується в ознаці знаку для подальшого використання.

Ознака переповнювання. Обчислення з використанням додаткових кодів проводяться над словами певної довжини. Якщо в процесі обчислень виходить результат більшої довжини, то повинен вироблятися сигнал, що вимагає розширення довжини слова. Якщо таке розширення неможливе, то обчислення повинні зупинятися. Засобом індикації того, що виникла подібна ситуація, є ознака переповнювання.

Ознака паритету. Ця ознака встановлюється у стан «1», якщо в результаті операції загальна кількість одиниць є парною. Ознака парності використовується для контролю на парність даних під час їх передавання. Такий контроль дозволяє виявити помилки, які при цьому можуть виникати.

3.2.4.2. Регістр команд

Вид кожної операції визначається за допомогою коду відповідної команди. У 8-бітовій ЕОМ можна розрізнити 256 кодів. Кількість байт, що відводяться для запису команди, визначається типом операції. У мікро-ЕОМ максимальна кількість цих байтів зазвичай дорівнює трьом.

Код операції, який вказує, що треба робити для обробки даних при виконанні команди, завжди розміщується в першому байті. Якщо вся команда займає 1 байт, то для коду операції відводиться цього байта, наприклад, 2 біта. Якщо команда займає 3 байти, то для коду операції відводиться весь перший байт.

Весь перший байт коду команди зчитується з пам'яті і передається до регістру команди в перебігу циклу зчитування незалежно від того, яка його частина відведена для запису коду операції. Декодування вмісту першого байта дозволяє визначити:

- скільки байтів міститься в команді;

- чи є вміст другого та третього байтів в сукупності адресою пам'яті, за якою зберігаються призначені для обробки дані;

- яка операція має виконуватись.

Для декодування перший байт передається з регістра команди до дешифратора коду операції, за роботи якого під впливом тактових сигналів виробляється потрібна послідовність сигналів управління. Це призводить до зчитування другого та третього байтів з пам'яті (якщо це необхідно), а також до власне виконання операції, що визначена командою.

3.2.4.3. Регістри загального призначення

Процесор містить певний набір регістрів загального призначення. Ці регістри забезпечують швидкий доступ до операндів, що зберігаються в них. Для адресації регістрів загального призначення використовується укорочене адресне поле завдовжки три біти. За допомогою слова такої довжини можна розрізнити 8 регістрів. Це регістри B, C, D, E, H та L. Ці регістри можуть бути об'єднані попарно, що дозволяє обробляти слова завдовжки як 8 бітів, так і 16 бітів.

3.2.4.4. Лічильник команд

Лічильник команд вказує, де в пам'яті розташовані байти даної команди. Пристрій управління збільшує вміст лічильника команд на одиницю кожн разу, коли байт коду команди передається з пам'яті до процесора. Якщо код команди складається з двох байт, її зчитування відбувається у два кроки.

Перед початком зчитування лічильник команд вже містить адресу байта поточної команди, оскільки вміст його був збільшений на одиницю. Наприкінці процедури зчитування попередньої команди перший байт відразу може передаватися до процесора, після чого вміст лічильника команд знову збільшується на одиницю. Тепер лічильник містить адресу другого байта поточної команди, після передачі якого до процесора вміст лічильника команд знову збільшується на одиницю і визначає адресу першого байта наступної команди.

3.2.4.5. Покажчик стека

Покажчик стека – це 16 розрядний регістр, який призначений для адресації елементів стекової пам'яті. Він визначає адреси тих комірок пам'яті у стеку, де зберігається потрібна адреса повернення.

3.2.4.6. Регістр адреси

Зчитування та запис інформації до пам'яті може відбуватись, якщо визначене значення відповідної адреси пам'яті. Ця адреса визначає комірку пам'яті, що призначена для запису або зчитування байта команди або байта даних. Процесор передає адресу з регістра до пам'яті по шині адреси. Для доступу до пам'яті потрібний певний час, через що можливість звернутись до потрібного слова в пам'яті з'являється не відразу. Існування такої затримки обумовлює необхідність зберігання адреси, сформованої процесором, протягом певного проміжку часу. Для цього в більшість ЕОМ вбудовується спеціальний регістр, призначений для зберігання адреси пам'яті, який має однойменну назву.

3.2.5. Генератор тактових сигналів

Події в мікро-ЕОМ мають відбуватися у потрібній послідовності і скоординовано, що вимагає організації управління ними в часі. Таке управління здійснюється з використанням інтегрального генератора тактових сигналів. Генератор тактових сигналів може розміщуватись на тому ж кристалі, що й мікропроцесор, або в окремій мікросхемі.

Тактові сигнали, які зазвичай позначаються *Ф*, можуть бути послідовністю прямокутних сигналів. В більшості випадків використовується пара сигналів для тактування: *Ф1* та *Ф2*, що є двома послідовностями прямокутних сигналів з однаковою амплітудою, частотою та шпаруватістю, але зсунутих за фазою на 180° (рис. 3.14).

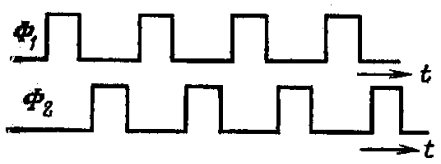


Рис. 3.14. Тактові імпульси мікро-ЕОМ

При розгляді роботи мікропроцесора в часі оперують термінами: цикл команди, машинний цикл та стан (рис. 3.15).

Циклом команди називають часовий інтервал, необхідний для зчитування команди з пам'яті та її виконання. Цикл команди реалізується впродовж 1-5 машинних циклів, точна кількість яких залежить від складності команди і дорівнює кількості звернень процесора до пам'яті або одного з

пристроїв введення-виведення. Таким чином, кількість машинних циклів у циклі команди визначається тим, скільки разів використовується шина даних. Цикл будь-якої команди складається щонайменше з одного машинного циклу, оскільки в найпростішому випадку необхідно витягувати з пам'яті 1 байт команди і передавати його до процесора.

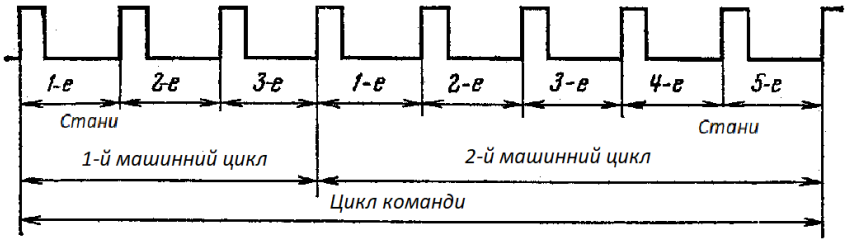


Рис. 3.15. Діаграма циклу команди

Кожний машинний цикл складається з певної послідовності елементарних дій, що називаються станами (тактами). Наприклад, для зчитування команди необхідно спочатку визначити значення потрібної адреси пам'яті та декодувати її. Тільки після цього перший байт команди можна передавати до процесора і записувати його до регістру команд.

Таким чином, стан – це проста дія, яка може виконуватись в мікро-ЕОМ. Стан виконується протягом одного періоду тактового сигналу. В окремому машинному циклі може бути від трьох до п'яти станів.

Для визначення тривалості виконання команди потрібно знати, яка кількість станів міститься в циклі команди та чому дорівнює період тактового сигналу.

3.2.6. Дешифратор команд. Пристрій управління

Пристрій управління є одним з найважливіших блоків мікропроцесора. Спільно з генератором тактових сигналів пристрій управління забезпечує правильну послідовність подій в мікро-ЕОМ. Після витягання команди з пам'яті та її дешифрування пристрій управління генерує необхідну для виконання команди послідовність сигналів.

Окрім цього, пристрій управління здатний самостійно реагувати на різні зовнішні сигнали, наприклад, на сигнали переривання від зовнішніх пристроїв. Сигнали готовності, що надходять від пам'яті або портів введення-виведення, також сприймаються пристроєм управління.

3.3. Часові діаграми роботи мікропроцесора

Характеристики мікропроцесора зазвичай наводяться їх виробниками у вигляді часових діаграм, на яких послідовність дій представляється як функція часу. Часову діаграму можна побудувати для будь-якої операції, що виконується мікро-ЕОМ.

Нижче розглядається часова діаграма виконання команди введення (рис. 3.16).

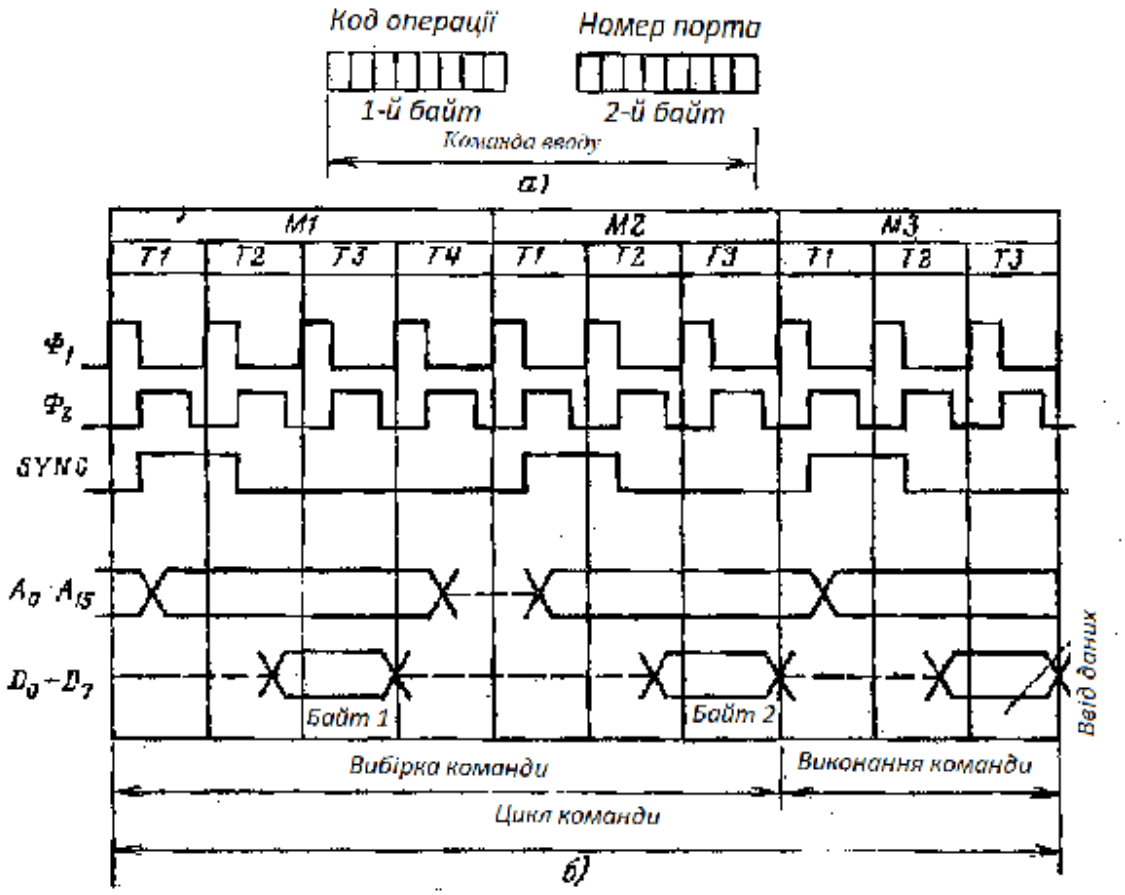


Рис. 3.16. Часова діаграма виконання команди введення [2]

Команда введення займає два байти. Перший байт містить код операції і вказує, які дії потрібно здійснити (прийняти дані з порту введення). Другий байт вказує на операнд, тобто з якого порту мають бути прийняті дані.

На рис. 3.16. зображені:

- тактові сигнали *Ф1* та *Ф2*, що змінюються у протифазі;

- сигнал синхронізації SYNC, що породжується сигналом *Ф2* (поява кожного сигналу SYNC означає початок нового машинного циклу);

- наявність або відсутність передачі сигналів A0-A15 шиною адреси (оскільки в загальному випадку неможливо вказати конкретні значення цих сигналів, зазначається, що шиною адреси передається (або не передається) певна інформація. Відсутність інформації позначається пунктирною лінією);

- наявність або відсутність передачі сигналів D0-D7 шиною даних.

3.4. Робота мікропроцесорної системи

На часовій діаграмі зображено три машинні цикли: *M1-M3*. Код операції зчитується протягом циклу *M1*. Протягом циклу *M2* з пам'яті зчитується адреса операнду, а протягом циклу *M3* відбувається виконання команди, тобто за адресою порту введення дані зчитуються і передаються до мікропроцесора. Таким чином, зчитування команди здійснюється протягом циклів *M1* та *M2*, а її виконання – протягом циклу *M3*.

Кожний машинний цикл складається з певної кількості станів. Код адреси передається шиною адреси протягом стану *T2* кожного машинного циклу. У циклі *M* – це адреса коду операції в пам'яті, в циклі *M2* – це адреса операнда (номер порту) в пам'яті, в циклі *M3* - це номер порту введення.

Під час стану *T2* кожного машинного циклу здійснюється перевірка умов, які можуть зробити необхідною затримку у виконанні даного машинного циклу. Однією з причин, які можуть затримку, може бути різниця у швидкодії між процесором та портом введення-виведення, або між процесором та пам'яттю.

Якщо існують причини для такої затримки, то процесор переходить в стан очікування. Якщо ж причин для затримки немає, перший і другий байти команди зчитуються з пам'яті протягом станів *T3* машинних циклів відповідно *M1* та *M2*. Ця інформація передається до процесора шиною даних у вигляді сигналів D0-D7. Протягом стану *T3* машинного циклу *M3* команда виконується, тобто дані приймаються з порту.

З часової діаграми випливає, що машинний цикл *M1* містить не три, а чотири стани. Четвертий стан відводиться для процесору для дешифрування коду операції.

Введення інформації пов'язане із зверненням до порту введення-виведення, внаслідок чого для виконання команди введення потрібно додати окремий машинний цикл. Якщо виконання команди реалізується виключно засобами процесора, а необхідність звернення до пам'яті або портів введення-

виведення відсутня, виконання команди може відбуватись протягом четвертого а, можливо, п'ятого стану попереднього машинного циклу. Отже, існує можливість реалізації циклу команди за один машинний цикл.

3.5. Призначення зовнішніх виводів мікропроцесора

Мікропроцесор підключається за допомогою зовнішніх виводів, призначення яких наведене на рис. 3.17.

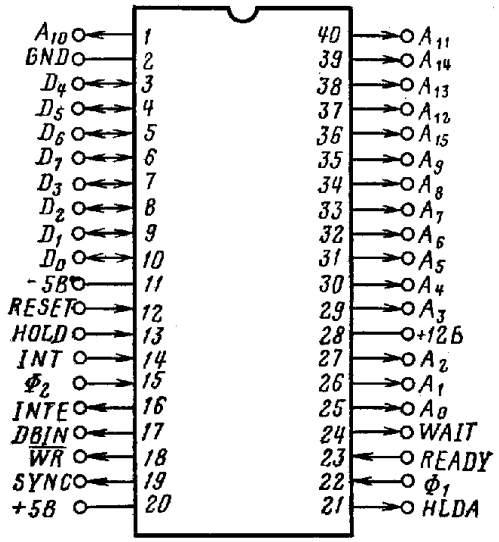


Рис. 3.17. Конфігурація системи зовнішніх виводів мікропроцесора [2]

A0-A15 – виводи шини адреси. Всі мікропроцесори мають з'єднання з шиною адреси, яка використовується для передачі адреси комірок пам'яті або одного з пристроїв введення-виведення. Оскільки адресація 16-розрядна, для цього необхідно використовувати 16 зовнішніх виводів: A0-A15. У деяких мікропроцесорів певні виводи шини адреси виконують подвійну функцію: за ними передаються або значення відповідних біт адреси, або інформація про процеси запису-зчитування в пам'яті. Іноді усі шістнадцять виводів шини адреси можуть використовуватись для підключення до шині даних в режимі мультиплексування.

D0-D7 – виводи шини даних. Шина даних є двоспрямованою, тобто ці виводи використовуються як для передачі інформації до процесора, так і від нього.

Ф1-Ф2 – тактові входи, на які подаються тактові сигнали від зовнішнього генератора тактових сигналів.

SYNC – вихід синхронізації. З цього виводу знімається сигнал синхронізації, який оповіщає пам'ять або пристрої введення-виведення про початок нового машинного циклу, чим забезпечується узгодження в часі роботи цих пристроїв з роботою мікропроцесора.

Виводи для підключення напруги живлення. На мікропроцесор подається напруга живлення +12 В, -5 В та +5 В зі точкою GND (*ground* - земля) джерела живлення. Деякі типи мікропроцесорів можуть мати одну напругу живлення +5 В.

RESET – вхід скидання. За цим входом можна скидати лічильник команд, тобто встановлювати його значення $0000_{(16)}$, а потім передавати в регістр адреси нульову адресу першої виконуваної команди. Після подачі сигналу скидання RESET вміст всіх інших регістрів мікропроцесора залишається невизначеним.

WR – вихід сигналу "Запис". Інформація передається шиною у двох напрямках: від мікропроцесора до пам'яті чи порту введення-виведення або від пам'яті чи порту введення-виведення до нього. Мікропроцесор має інформувати пам'ять або порт введення-виведення про виконання ним операції запису (WRITE) або читання (READ). Вказівка на те, що виконується операція запису, здійснюється мікропроцесором шляхом видачі на вивід WR сигналу високого рівня, а вказівка на операцію читання – шляхом видачі на вивід WR сигналу низького рівня.

READY – вхід сигналу готовності. Якщо необхідно здійснювати обмін даними між мікропроцесором та пам'яттю або пристроями введення-виведення, то перед початком передачі даних слід визначити адресу модуля, до якого передається інформація або, навпаки, від якого вона приймається. Проте, перш ніж зможе розпочатись власне обмін даними, витримується певний проміжок часу, який називається часом доступу. Фіксація того, що цей проміжок часу закінчився, здійснюється пам'яттю або портом введення-виведення шляхом надсилання сигналу готовності на вхід READY мікропроцесора. Це надає мікропроцесору інформацію про те, що може розпочатися обмін даними.

WAIT – вихід сигналу очікування. Мікропроцесор зазначає, що він в режимі очікування, шляхом видачі сигналу на вивід WAIT.

DBIN – вихід сигналу "шина даних в режимі введення". Якщо до мікропроцесора передаються дані при зчитуванні їх з пам'яті або при виконанні операції введення, він вказує на це шляхом генерації сигналу на

виводі DBIN (*data bus in*). При зчитуванні з пам'яті сигнал DBIN відображає команду READ, а при виконанні операції введення – команду IN.

HOLD – вхід сигналу захоплення шини.

HLDA – вихід сигналу підтвердження захоплення шини. Великі об'єми інформації, що зберігаються на магнітних дисках, можуть передаватися до основної пам'яті мікро-ЕОМ без втручання мікропроцесора. Такий режим називають прямим доступом до пам'яті (ПДП). У цьому режимі мікропроцесор повинен відключатися від шин адреси та даних. Це відбувається при появі сигналу захоплення шин на вході HOLD. Результатом появи сигналу захоплення шин є те, що мікропроцесор відключається від своїх шин та інформує про це інші модулі за допомогою сигналу HLDA (*Hold acknowledge output*) підтвердження захоплення шин, що подається на відповідний вихід.

INT – вхід сигналу «запит переривання».

INTE – вихід сигналу дозволу переривання. Якщо виникає необхідність передачі даних від пристрою введення, то відповідний пристрій посилає запит переривання до мікропроцесора. Цей сигнал на вихід INT (*interrupt*). У мікропроцесорі 8080 передбачена можливість використання спеціальної команди, що забороняє процесору реагувати на запит переривання. Якщо такий сигнал відсутній, то мікропроцесор видає сигнал дозволу переривання на вихід INTE (*interrupt enable*). Якщо сигнал INTE дорівнює «1», то процесор відпрацьовуватиме запит переривання; якщо ж сигнал INTE дорівнює «0», запит переривання від пристрою введення ігнорується мікропроцесором.

Контрольні запитання та завдання

1. Опишіть формат команд мікропроцесора.
2. Надайте визначення мікропроцесора.
3. Для чого призначений акумулятора та регістр команд?
4. Які ознаки містяться в регістрі ознак?
5. З якими шинами працює мікропроцесор?
6. Для чого призначений лічильник команд?
7. Яку роль в роботі мікропроцесора відіграє генератор тактових сигналів?
8. Скільки станів може міститись в машинному циклі?
9. Скільки машинних циклів триває в цикл команди?
10. Що відбувається у стані *T1* машинного циклу?

4. БУДОВА ТА ФУНКЦІОНУВАННЯ МІКРОКОНТРОЛЕРІВ

Розглянемо будову та функціонування мікроконтролерів на прикладі контролерів PIC (*Programmable Input/output Controller*), які виробляються фірмою Microchip Technology.

PIC16CXX – це 8-розрядні мікроконтролери з RISC архітектурою, Вони приваблюють користувачів низькою ціною, низьким енергоспоживанням та високою швидкодією. Мікроконтролери PIC мають вбудовану EEPROM для зберігання програм, ОЗУ даних та випускаються у 18-, 28- та 40-вивідних корпусах [4].

Мікроконтролери мають різні конструкції – одноразово програмовані користувачем контролери, призначені для повністю тестованих та закінчених виробів. Випускаються у дешевих пластмасових корпусах із попередньо заданим типом зовнішнього генератора – кварцовим або RC.

Для відлагодження програм та макетування випускається варіант контролерів з ультрафіолетовим стиранням. Допускають велику кількість циклів стирання/запису і мають дуже короткий час стирання – 1-2 мс. Ціна вище.

Для виробів, програма яких може змінюватись або містить будь-які змінні частини, таблиці, параметри калібрування, ключі тощо. випускається контролер, що електрично стирається та перепрограмовується. Він також містить EEPROM даних.

Мікроконтролери сімейства PIC мають дуже ефективну систему команд, що містить 35 інструкцій. Усі інструкції, за виключенням умовних переходів, виконуються впродовж одного циклу. Один цикл виконання інструкції складається з 4-х періодів тактової частоти. Кожна інструкція складається з 14 біт, які розділяються на код операції та операнд.

Висока швидкість виконання команд досягається за рахунок використання двошинної Гарвардської архітектури замість одношинної Фон-Нейманівської. Гарвардська архітектура ґрунтується на наборі регістрів з розділеними шинами та адресним простором для команд та даних. Набір регістрів означає, що всі програмні об'єкти (порти введення/виведення, комірки пам'яті, таймер) є фізично реалізованим апаратним регістром.

Пам'ять даних (ОЗП) має розрядність 8 біт, пам'ять програм (ППЗУ) – 12 або 14 біт. Використання Гарвардської архітектури дозволяє досягти високої швидкості виконання бітових, байтових та регістрових операцій. До

того ж Гарвардська архітектура допускає конвеєрне виконання інструкцій, коли одночасно виконується поточна інструкція та зчитується наступна.

4.1. Технічні характеристики мікроконтролерів

Мікроконтролери PIC (МК PIC) мають високошвидкісну RISC архітектуру з набором команд з 35 інструкцій. Усі команди, за виключенням команд переходів, виконуються за один цикл [4].

Тактова частота МК PIC – 20 МГц, тактовий сигнал – 200 нс, один машинний цикл.

До технічних характеристик МК PIC належать такі:

- FLASH пам'ять програм містить до 8К×14 слів;
- RAM-пам'ять даних містить до 368 × 8 байт;
- EEPROM пам'ять даних містить до 256 × 8 байт;
- обслуговує до 14 джерел переривань;
- наявні:
    - 8-рівневий апаратний стек;
    - режими адресації: прямий, непрямий та відносний;
    - скидання живлення (POR);
    - таймер скидання (PWRT) та таймер очікування запуску генератора (OST) після включення живлення;
    - сторожовий таймер WDT з власним RC-генератором;
    - програмований захист пам'яті програм;
    - режим енергозбереження SLEEP;
    - вибір параметрів тактового генератора;
- підтримується:
    - високошвидкісна енергозберігаюча CMOS FLASH/EEPROM технологія;
    - повністю статична архітектура;
    - програмування у готовому пристрої.
    - режим внутрішньосхемного налагодження;.
- допускається широкий діапазон напруги живлення: від 2 до 5 В;
- підвищена здатність навантаження портів введення/виведення 25 мА.
- мале енергоспоживання.

МК PIC використовують такі ***периферійні модулі***:

- таймер 0: 8-розрядний таймер/лічильник з 8-розрядним програмованим дільником;
- таймер 1: 16-розрядний таймер/лічильник з 8-розрядним програмованим попередником і вихідним дільником;
- таймер 2: 8-розрядний таймер/лічильник з 8-розрядним програмованим попередником та вихідним дільником;
- два модулі порівняння/захоплення/ШІМ (PCP);
- багатоканальний 10-розрядний АЦП;
- послідовний синхронний порт MSSP:
  - провідний/відомий режим SPI;
  - провідний/відомий режим I2C.
- послідовний синхронно-асинхронний приймач USART з підтримкою детектування адреси;
- ведений 8-розрядний паралельний порт PSP з підтримкою зовнішніх сигналів RD, WR, CS;
- детектор зниженої напруги (BOD) для скидання за зниження напруги живлення (BOR).

Схема розташування виводів мікроконтролера PIC наведена на рис. 4.1, а їх призначення наведене у табл. 4.1.

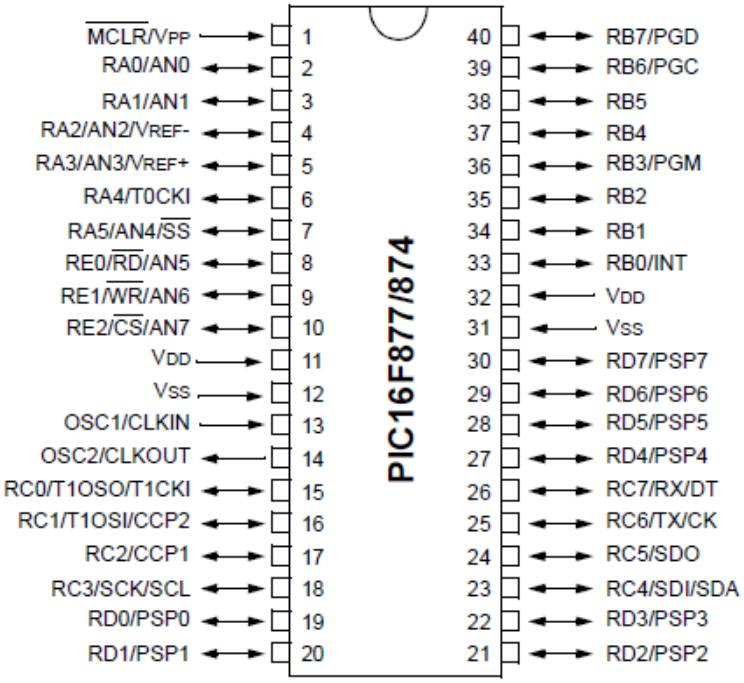


Рис. 4.1. Схема розташування виводів мікроконтролера PIC [4]

Таблиця 4.1

Призначення виводів мікроконтролерів PIC16F874 та PIC16F877

Позначення виводу	Номери виводів				Тип буфера	Опис
	DIP	PLCC	QFP	I/O/P		
OSC1/CLKIN	13	14	30	I	ST/ CMOS	Вхід генератора/вхід зовнішнього тактового сигналу
OSC2/CLKOUT	14	15	31	O	-	Вихід генератора. Підключається кварцовий або керамічний резонатор. В RC режимі тактового генератора на виході OSC2 присутній тактовий сигнал CLKOUT, якій дорівнює $F_{OSC}/4$
- MCLR/V_{pp}	1	2	18	I/P	ST	Вхід скидання мікроконтролера або вхід напруги програмування. Скидання мікроконтролера відбувається при низькому логічному рівні сигналу на вході.
RA0/AN0	2	3	19	I/O	TTL	Двонаправлений порт введення/виведення PORTA RA0 може бути налаштований як аналоговий канал 0
RA1/AN1	3	4	20	I/O	TTL	RA1 може бути налаштований як аналоговий канал 1
RA2/AN2/V_{REF-}	4	5	21	I/O	TTL	RA2 може бути налаштований як аналоговий канал 2 або вхід від'ємної опорної напруги
RA3/AN3/V_{REF+}	5	6	22	I/O	TTL	RA3 може бути налаштований як аналоговий канал 3 або вхід додатно\ опорної напруги
RA4/TOCKI	6	7	23	I/O	ST	RA4 може застосовуватися як вхід зовнішнього тактового сигналу для TMR0.
RA5/-SS/AN4	7	8	24	I/O	TTL	Вихід з відкритим стоком RA5 може бути налаштований як аналоговий канал 4 або вхід вибору мікросхеми в режимі веденого SPI
RB0/INT	33	36	8	I/O	TTL/ST	Двонаправлений порт вводу/виводу PORTB. PORTB має програмно підключаємі підтягуючі резистори на входах. RB0 може застосуватися як вхід зовнішних переривань.
RB1	34	37	9	I/O	TTL	RB1
RB2	35	38	10	I/O	TTL	RB2
RB3/PGM	36	39	11	I/O	TTL	RB3 може застосовуватися як вхід для режиму низковольтного програмування.
RB4	37	41	14	I/O	TTL	RB4 переривання по зміненню рівня вхідного сигналу.
RB5	38	42	15	I/O	TTL	RB5 переривання по зміненню рівня вхідного сигналу.
RB6/PGC	39	43	16	I/O	TTL/ST	RB6 переривання по зміненню рівня вхідного сигналу або вивід для режиму внутрішньосхемного налаштування ICD. Тактовий сигнал в режимі програмування.
RB7/PGD	40	44	17	I/O	TTL/ST	RB7 переривання по зміненню рівня вхідного сигналу або вивід для режиму внутрішньосхемного налаштування ICD. Вивод даних в режимі програмування.

Позначення виводу	Номери виводів				Тип буфера	Опис
	DIP	PLCC	QFP	I/O/P		
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	Двонаправлений порт вводу/виводу PORTC. RC0 може застосуватися як вихід генератора TMR1 або входу зовнішнього тактового сигналу для TMR1.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 може застосовуватися як вхід генератора для TMR1 або вивід модуля CCP2.
RC2/CCP1	17	19	36	I/O	ST	RC2 застосовуватися як вивід модуля CCP1
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 може застосовуватися як вхід/вихід тактового сигналу в режимі SPI і I^2C.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 може застосовуватися як вхід даних в режимі SPI або вхід/вихід даних в режимі I^2C.
RC5/SDO	24	26	43	I/O	ST	RC5 може застосовуватися як вихід даних в режимі SPI.
RC6/TX/CK	25	27	44	I/O	ST	RC6 може застосовуватися як вивід передавача USART в асинхронному режимі або як вивід синхронізації USART в синхронному режимі.
RC7/RX/DT	26	29	1	I/O	ST	RC7 може застосовуватися як вивід приймача USART в асинхронному режимі або як вивід синхронізації USART в синхронному режимі.
RD0/PSPO	19	21	38	I/O	ST/TTL	Двонаправлений порт вводу/виводу PORTD або ведений паралельний порт для підключення до шини мікропроцесора..
RD1/PSP1	20	22	39	I/O	ST/TTL	
RD2/PSP2	21	23	40	I/O	ST/TTL	
RD3/PSP3	22	24	41	I/O	ST/TTL	
RD4/PSP4	27	30	2	I/O	ST/TTL	
RD5/PSP5	28	31	3	I/O	ST/TTL	
RD6/PSP6	29	32	4	I/O	ST/TTL	
RD7/PSP7	30	33	5	I/O	ST/TTL	
RE0/-RD/AN5	8	9	25	I/O	ST/TTL	Двонаправлений порт вводу/виводу PORTE. RE0 може застосовуватися як управляючій вхід читання PSP або аналогового каналу 5
RE1/-WR/AN6	9	10	26	I/O	ST/TTL	RE1 може застосовуватися як управляючій вхід запису PSP або аналогового каналу 6
RE2/-CS/AN7	10	11	27	I/O	ST/TTL	RE0 може застосовуватися як управляючій вхід вибору PSP або аналогового каналу 7
V_{SS}	12,31	13,34	6,29	P	-	Загальний вивід для внутрішньої логіки і портів введення/виведення
V_{DD}	11,32	12,35	7,28	P	-	Додатна напруга живлення для внутрішньої логіки і портів введення/виведення
NC	-	1,17, 28,40	12,13, 33,34	-	-	Ці виводи всередині мікросхеми не підключені

4.2. Будова мікроконтролера

Структура мікроконтролера PIC наведена на рис. 4.2.

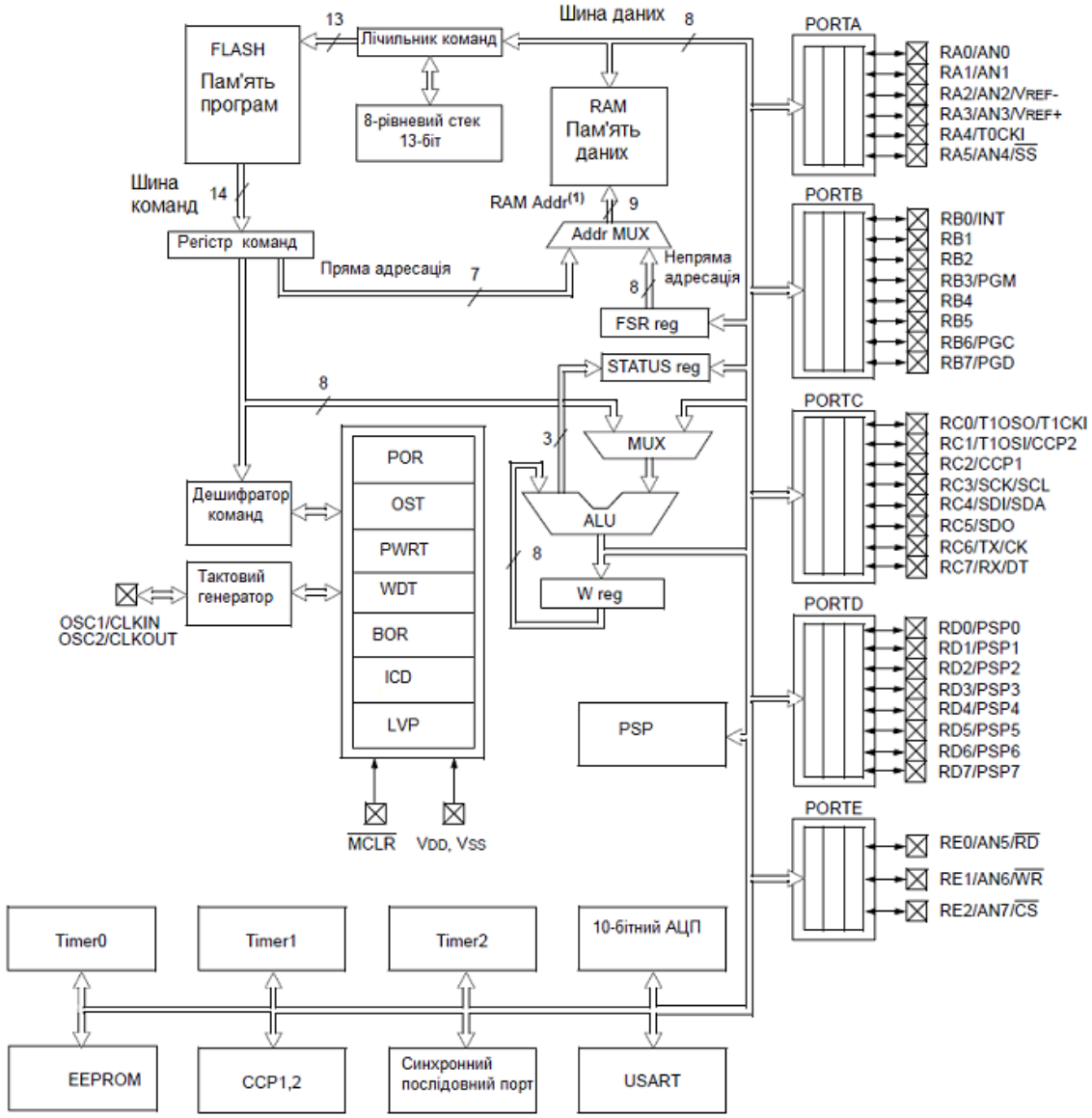


Рис. 4.2. Структура мікроконтролера PIC [4]

Мікроконтролер складається з регістрів, 8-розрядної шини даних, 13-розрядної адресної шини, 14-розрядної шини програм, арифметико-логічного пристрою, регістру акумулятора W, пам'яті програм, пам'яті даних, що складається з регістрів загального призначення та регістрів спеціального призначення, регістру інструкцій, регістру ознак STATUS, регістру непрямої адресації FSR, дешифратора інструкцій, пристрою керування, тактового генератора, портів введення/виведення A, B, C, D, E, таймерів, АЦП,

послідовних та паралельних портів, пам'яті даних, що перепрограмовується та 8-рівневого стека даних.

Принцип роботи мікроконтролера аналогічний до роботи мікропроцесорів. До лічильника команд завантажується адреса комірки пам'яті, де знаходиться перша команда програми. Адреса по 13-розрядній шині адреси надходить до пам’яті програм. Перша команда витягується з пам'яті та по шині програм надходить до регістру інструкцій. З регістру інструкцій код операції надходить у детектор інструкцій та пристрій керування. Інструкція декодується та пристрій керування починає її виконувати.

4.3. Пам'ять мікроконтролерів

У мікроконтролерах PIC функціонує три види пам'яті:

- пам'ять програм;
- пам'ять даних;
- EEPROM пам'ять даних.

Пам'ять програм та пам'ять даних мають окремі шину даних та шину адреси, що дозволяє виконувати паралельний доступ до них.

4.3.1. Організація пам'яті програм

Мікроконтролери мають 13-розрядний лічильник команд PC, здатний адресувати 8К × 14 слів пам'яті програм. Фізично реалізовано FLASH пам'яті 8К × 14 для PIC16F877. Звернення до фізично нереалізованої пам'яті програм призведе до адресації реалізованої пам'яті.

Адреса вектора скидання – 0000h. Адреса вектора переривань – 0004h.

Пам'ять програм складається з 4 сторінок (рис. 4.3).

Сторінка 0 має адреси: 0005h-07FFh;

сторінка 1 має адреси: 0800h-0FFFh;

сторінка 2 має адреси: 1000h-17FFh;

сторінка 4 має адреси: 1800h-1FFFh.

4.3.2 Організація пам'яті даних

Пам'ять даних поділяється на 4 банки, які містять регістри загального та спеціального призначення (SFR). Для управління банками призначені біти

RP1 (STATUS<6>) та RP0 (STATUS<5>). У табл. 4.2 наведені стани управляючих бітів при зверненні до банків пам'яті даних.

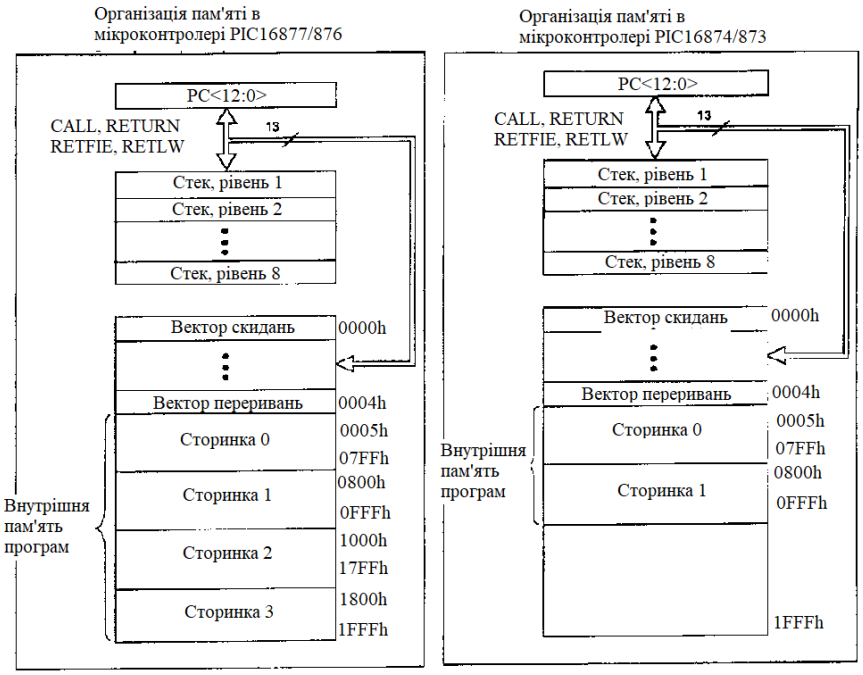


Рис. 4.3. Організація пам'яті програм МК PIC [4]

Таблиця 4.2

Стан управляючих бітів при зверненні до банків пам'яті даних

RP1:RP0	Банк
00	0
01	1
10	2
11	3

Об'єм пам'яті банків даних – до 128 байт (7Fh). На початку банку розміщуються регістри спеціального призначення, потім – регістри загального призначення, виконані як статичне ОЗП. Усі реалізовані банки містять регістри спеціального призначення. Регістри спеціального призначення, що часто використовуються, можуть відображатись також в інших банках пам'яті.

До регістрів загального призначення можна звертатися прямою чи непрямою адресацією, через регістр FSR.

За допомогою регістрів спеціального призначення виконується керування функціями ядра та периферійними модулями мікроконтролера. Регістри спеціального призначення реалізовані як статичне ОЗП. У табл. 4.3 та табл. 4.4 описані регістри, що керують функціями ядра мікроконтролера [4].

Таблиця 4.3

Карта пам’яті даних мікроконтролерів PIC16F877/876

Регістри	Адреса	Регістри	Адреса	Регістри	Адреса	Регістри	Адреса
Банк 0		Банк 1		Банк 2		Банк 3	
Регістр непрямої адресації	00h	Регістр непрямої адресації	80h	Регістр непрямої адресації	100h	Регістр непрямої адресації	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Резерв	18Eh
TMR1H	0Fh		8Fh	EEADRH	110Fh	Резерв	18Fh
T1CON	10h		90h	Регістри загального призначення 16 байтів	110h	Регістри загального призначення 16 байтів	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
Регістри загального призначення 96 байтів	20h	Регістри загального призначення 80 байтів	A0h	Регістри загального призначення 80 байтів	120h	Регістри загального призначення 80 байтів	1A0h
			EFh		16Fh		1EFh
		Доступ до 70h -7Fh	F0h	Доступ до 70h -7Fh	170h	Доступ до 70h -7Fh	1F0h
	7Fh		FFh		17F		1FFh

Таблиця 4.4

Регістри спеціального призначення

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR
Банк 0										
00h	INDF	Звернення до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
01h	TMR0	Регістр таймера TMR0								xxxx xxxx
02h	PCL	Молодші біти лічильника команд PC								0000 0000
03h	STATUS	IRP	RP1	RP0	-TO	-PD	Z	DC	C	0001 1xxx
04h	FSR	Регістр адреси м при непрямій адресації								xxxx xxxx
05h	PORTA	-	-	Запис до вихідної заскочки PORTA, читання стану вив. PORTA						--0x 0000
06h	PORTB	Запис до вихідної заскочки PORTB, читання станів виводів PORTB								xxxx xxxx
07h	PORTC	Запис до вихідної заскочки PORTC, читання станів виводів PORTC								xxxx xxxx
08h	PORTD	Запис до вихідної заскочки PORTD, читання станів виводів PORTD								xxxx xxxx
09h	PORTE	-	-	-	-	-	RE2	RE1	RE0	----- - xxx
0Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					---0 0000
0Bh	INTCON	GIE	PEIE	TOIE	INTE	PBIE	TOIF	INTF	RBIF	0000 000x
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
0Dh	PIR2	-	-	-	EEIF	BCLIF	-	-	CCP2IF	-r-0 0--0
0Eh	TMR1L	Молодший байт 16-розрядного таймера 1								xxxx xxxx
0Fh	TMR1H	Старший байт 16-розрядного таймера 1								xxxx xxxx
10h	T1CON	-	-	T1SKPS1	T1SKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000
11h	TMR2	Регістр таймера TMR2								0000 0000
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000
13h	SSPBUF	Буфер приймача MSSP /регістр передавача								xxxx xxxx
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000
15h	SSPR1L	Молодший байт захоплення/порівняння /ШИМ CCP1								xxxx xxxx
16h	CCPR1H	Старший байт захоплення/порівняння /ШИМ CCP1								xxxx xxxx
17h	CCP1CON	-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	- -00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
19h	TXREG	Регістр даних передавача USART								0000 0000
1Ah	RCREG	Регістр даних приймача USART								0000 0000
1Bh	CCPR2L	Молодший байт захоплення/ порівняння ШИМ CCP2								xxxx xxxx
1Ch	CCPR2H	Старший байт захоплення/ порівняння ШИМ CCP2								xxxx xxxx
1Dh	CCP2CON	-	-	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000
1Eh	ADRESH	Старший байт результату перетворення АЦП								xxxx xxxx
1Fh	ADCON0	ADCS1	ADCS2	CHS2	CHS1	CHS0	GODONE	-	ADON	0000 00-0

Продовження табл. 4.4

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR
Банк 1										
80h	INDF	Звернення до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
81h	OPTION_REG	Регістр таймера TMR0								1111 1111
82h	PCL	Молодші біти лічильника команд PC								0000 0000
83h	STATUS	IRP	RP1	RP0	-TO	-PD	Z	DC	C	0001 1xxx
84h	FSR	Регістр адреси м при непрямій адресації								xxxx xxxx
85h	TRISA	-	-	Напрямок виводів PORTA						--11 1111
86h	TRISB	Напрямок виводів PORTB								1111 1111
87h	TRISC	Напрямок виводів PORTC								1111 1111
88h	TRISD	Напрямок виводів PORTD								1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	-	Напрямок виводів PORTE			----- - 111
8Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					---0 0000
8Bh	INTCON	GIE	PEIE	TOIE	INTE	PBIE	TOIF	INTF	RBIF	0000 000x
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
8Dh	PIE2	-	-	-	EEIE	BCLIE	-	-	CCP2IE	-r-0 0--0
8Eh	PCON	-	-	-	-	-	-	-POR	-BOR	---- --qq
8Fh	-	Не реалізовано								-
90h	-	Не реалізовано								-
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000
92h	PR2	Регістр періоду таймера TMR2								1111 1111
93h	SSPADD	Регістр адреси/ Регістр генератора швидкості обміну								0000 0000
94h	SSPSTAT	SMP	CKE	D/-A	P	S	R/-W	UA	BF	0000 0000
95h	-	Не реалізовано								-
96h	-	Не реалізовано								-
97h	-	Не реалізовано								-
98h	TXSTA	CSRC	CKE	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010
99h	SPBRG	Регістр даних передавача USART								0000 0000
9Ah		Не реалізовано								-
9Bh		Не реалізовано								-
9Ch		Не реалізовано								-
9Dh		Не реалізовано								-
9Eh	ADRESL	Молодший байт результату перетворення АЦП								xxxx xxxx
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000

Продовження табл. 4.4

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR
Банк 2										
100h	INDF	Звернення до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
101h	TMR0	Регістр таймера TMR0								xxxx xxxx
102h	PCL	Молодші біти лічильника команд PC								0000 0000
103h	STATUS	IRP	RP1	RP0	-TO	-PD	Z	DC	C	0001 1xxx
104h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
105h	-	Не реалізовано								-
106h	PORTBB	Запис в вихідну заскочку PORTB, читання станів виводів PORTB								xxxx xxxx
107h	-	Не реалізовано								-
108h	-	Не реалізовано								-
109h	-	Не реалізовано								-
10Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					---0 0000
10Bh	INTCON	GIE	PEIE	TOIE	INTE	PBIE	TOIF	INTF	RBIF	0000 000x
10Ch	EEDATA	Регістр даних, молодший байт								xxxx xxxx
10Dh	EEADR	Регістр адреси, молодший байт								xxxx xxxx
10Eh	EEDATH	-	-	Регістр даних, старший байт						xxxx xxxx
10Fh	EEADRH	-	-	-	Регістр адреси, старший байт					xxxx xxxx
Банк 3										
180h	INDF	Звернення до регістра, адреса якого записана в FSR (не фізичний регістр)								-
181h	OPTION_REG	Регістр таймера TMR0								0000 0000
182h	PCL	Молодші біти лічильника команд PC								1111 1111
183h	STATUS	IRP	RP1	RP0	-TO	-PD	Z	DC	C	0001 1xxx
184h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
185h	-	Не реалізовано								-
186h	TRISB	Напрямок виводів PORTB								1111 1111
187h	-	Не реалізовано								-
188h	-	Не реалізовано								-
189h	-	Не реалізовано								-
18Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					---0 0000
18Bh	INTCON	GIE	PEIE	TOIE	INTE	PBIE	TOIF	INTF	RBIF	0000 000x
18Ch	EECON1	EEPGD	-	-	-	WRERR	WREN	WR	RD	x--- x000
18Dh	EECON2	Регістр управління 2 (фізично не реалізований)								---- ----
18Eh	-	Резерв								-
18Fh	-	Резерв								-

Позначення: - – не – застосовується, читається як 0; u – не змінюється; x – не відомо; q – залежить від умов; r – резерв

Примітки до табл. 4.4:

1. Старший байт лічильника команд PC програмно не доступний. В регістрі PCLATH зберігаються старші біти <12:8>, які переписуються в старший байт лічильника команд.
2. Біти PSPIE і PSPIF в мікроконтролерах PIC16F873/876 не реалізовані, завжди мають дорівнювати 0.
3. Звернення до цих регістрів можна виконати з деякого банку.
4. Регістри PORTD, PORTE, TRISD, TRISE не реалізовані в мікроконтролерах PIC16F873/876, вони читаються як 0.
5. Резервні біти PIR2<6> і PIE2<6> при запису в регістр PIR2 завжди мають дорівнювати 0.

4.3.3. EEPROM пам'ять даних. FLASH пам'ять програм

Дані з EEPROM пам'яті даних та FLASH пам'яті програм можуть бути прочитані/перезаписані у нормальному режимі роботи мікроконтролера у всьому діапазоні напруги живлення. Операції виконуються з одним байтом для EEPROM пам'яті даних та одним словом для FLASH пам'яті програм. Запис здійснюється за принципом «стирання-запис» для кожного байта або слова. Сформована кодом програми операція стирання не може бути виконана при увімкненому захисті запису.

Доступ до пам'яті програм дозволяє виконати обчислення контрольної суми. Дані, що записані до пам'яті програм, можуть бути використані у вигляді: 14-розрядних чисел; калібрувальної інформації; серійних номерів; упакованих 7-розрядних символів ASCII-кодів тощо. У разі виявлення недійсної команди в пам'яті програм виконується порожній цикл NOP.

Кількість циклів стирання/запису визначається електричними характеристиками цих пристроїв. Кількість циклів стирання/запису для FLASH пам'яті програм є значно меншою у порівнянні з EEPROM пам'яттю даних. Тому EEPROM пам'ять даних має використовуватись для збереження даних, що часто змінюються. Тривалість запису даних керується внутрішнім таймером. Вона залежить від напруги живлення, температури та має невеликий технологічний «розкид».

Під час запису байта або слова автоматично стирається відповідна комірка, а потім виконується запис. Запис до EEPROM пам'яті даних не впливає на виконання програми, а під час запису до FLASH-пам'яті програм виконання програми зупиняється на час запису. Під час циклу запису не можна звернутися до пам'яті програм. Впродовж операції запису тактовий генератор продовжує працювати, периферійні модулі увімкнені та генерують

переривання, які «ставляться у чергу» до завершення циклу запису. Після завершення запису виконується завантажена команда та відбувається перехід за вектором переривань, якщо переривання дозволене і воно виникло під час запису.

Доступ до функцій запису/читання EEPROM пам'яті даних та FLASH-пам'яті програм виконується шістьма регістрами спеціального призначення:

- EEDATA;
- EEDATH;
- EEADR;
- EEADRH;
- EECON1;
- EECON2.

Операції читання/запису EEPROM пам'яті даних не припиняють виконання програми. У регістрі EEADR зберігається адреса комірки EEPROM пам'яті даних. Дані зберігаються/читаються з регістру EEDATA. У мікроконтролерах 877 обсяг EEPROM пам'яті даних дорівнює 256 байт.

Читання FLASH-пам'яті програм не впливає на виконання програми, а під час операції запису виконання програми призупинено. У спарених регістрах EEADRH:EEADR зберігається 13-розрядна адреса комірки пам'яті програм, до якої необхідно здійснювати звернення. Спарені регістри EEDATH:EEDATA містять 14-розрядні дані для запису та відображають значення з пам'яті програм під час читання. У регістри EEADRH:EEADR має бути завантажена адреса фізично реалізованої пам'яті програм (від 0000h до 1FFFh), тому що циклічна адресація не підтримується.

4.3.4. Регістри EECON1, EECON2

Регістр EECON1 містить біти управління непрямого запису/читання у EEPROM пам'ять даних, FLASH-пам'ять програм. Регістр EECON2 фізично не реалізований, він використовується лише при операціях запису з метою запобігання випадковому запису.

Значення біта EEPGD у регістрі EECON1 визначає тип пам'яті, до якої буде виконано звернення. Якщо EEPGD = 0, то операції відносяться до EEPROM пам'яті даних. Коли EEPGD =1, звернення відбувається до FLASH-пам'яті програм.

В операції читання використовується лише один додатковий біт RD, який ініціалізує операцію читання із зазначеної пам'яті. Встановивши біт RD

в «1», значення комірки пам'яті буде доступне у регістрі даних. Біт RD не може бути скинутий програмно в «0», він автоматично скидається після закінчення операції читання. При читанні з EEPROM пам'яті дані будуть доступні в регістрі EEDATA у наступному машинному циклі після встановлення біта RD. При читанні з FLASH-пам'яті програм дані будуть доступні в регістрі EEDATA:EEDATH на другому машинному циклі після встановлення біта RD.

В операції запису використовуються два службові біти WR, WREN і два біти статусу WRERR, EEIF. Біт WREN застосовується для дозволу/заборони операції запису (WREN = 0 – операція запису заборонена). Перед виконанням запису біт WREN необхідно встановити у «1». Біт WR призначений для ініціалізації запису, апаратно скидається в «0» після завершення операції запису. Прапор переривання EEIF встановлюється в «1» після завершення запису. Цей прапор повинен бути скинутий програмно в «0» перед встановленням біта WR.

Для EEPROM пам'яті даних:

Після встановлення бітів WREN, WR в «1» стирається вказана в регістрі EEADR комірка EEPROM пам'яті, а потім відбувається запис даних з регістру EEDATA. Операція запису супроводжується виконанням програмного коду. Після завершення запису встановлюється прапор переривання EEIF «1»

Для FLASH-пам'яті програм:

Після встановлення бітів WREN, WR в «1» мікроконтролер зупиняє виконання програми. Стирається комірка пам'яті програм, яка зазначена у регістрі EEADRH:EEADR, після чого відбувається запис до неї даних з регістру EEDATAH:EEDATA. Після завершення запису прапор переривання EEIF встановлюється в «1», а мікроконтролер продовжує виконувати код програми.

Біт WRERR визначає, що відбулось скидання мікроконтролера під час виконання операції запису. Біт WRERR встановлюється в «1», якщо під час виконання запису даних відбулось скидання даних за сигналом -MCLR або переповнення сторожового таймера WDT у нормальному режимі. Перевіривши стан біта WRERR, користувач може повторити запис (регістри EEDATA та EEADR не змінюють свого значення). Після скидання мікроконтролера за сигналом -MCLR, або після переповнення сторожового

таймера WDT в нормальному режимі вміст регістрів даних, адреси та біт EEPGD не змінюється.

Таблиця 4.5

Регістр EECON1

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	-	-	-	WRERR	WREN	WR	RD
Біт 7	EEPGD: Біт вибору EEPROM пам'яті даних/FLASH пам'ять програм 1 - FLASH пам'ять програм 0 - EEPROM пам'ять даних						
Біт 6-4	Не застосовуються: читаються як «0»						
Біт 3	WRERR: Прапор помилки запису даних 1 - запис перерваний (відбулося одне з скидань: за сигналом – MCLR, за переповненням WDT нормальному режимі) 0 - запис закінчений						
Біт 2	WREN: Дозвіл запису даних 1 - запис дозволений 0 - запис заборонений						
Біт 1	WR: Ініціалізація запису даних (програмно може бути встановлена тільки в «1») 1 - ініціалізувати запис (скидається в «0» апаратно) 0 - запис закінчений						
Біт 0	RD: Ініціалізація читання даних (програмно може бути встановлена тільки в «1») 1 - ініціалізувати читання (скидається в «0» апаратно) 0 - читання закінчене						

4.3.5. EEPROM пам'ять даних

Читання з EEPROM пам'яті даних. Для читання даних з EEPROM пам'яті необхідно здійснювати таку послідовність дій:

1) записати адресу до регістра EEADR;

2) скинути в «0» біт EEPGD для звернення до EEPROM пам'яті даних;

3) ініціалізувати операцію читання установкою біта RD в «1»;

4) прочитати дані з регістру EEDATA.

Після встановлення в «1» біта RD дані будуть доступні у регістрі EEDATA на наступному машинному циклі. Дані в регістрі EEDATA зберігаються до виконання наступної операції читання або запису EEDATA.

Приклад програми читання з EEPROM пам’яті даних:

```
BSF     STATUS,RP1      ;
BCF     STATUS,RP0      ;Обрати банк 2
MOV F   ADDR,W          ;Записати адресу
MOVWF   EEADR           ;комірки пам’яті
BSF     STATUS,RP0      ;Обрати банк 3
BCF     EECON1,EEPGD    ;Обрати EEPROM пам’ять
BSF     EECON1,RD       ;Ініціалізувати читання
BCF     STATUS,RP0      ;Обрати банк 2
MOVF    EEDATA,W            ;W = EEDATA
```

Запис EEPROM пам'ять даних.

Запис даних до EEPROM пам'яті даних трохи складніше читання. Адреса комірки пам'яті та записувані дані повинні бути поміщені у відповідні регістри спеціального призначення, біт EEPGD скидається в «0». Біт WREN повинен завжди дорівнювати нулю, якщо не здійснюється безпосередній запис в пам’ять. Біт WR може бути встановлений в «1» тільки, якщо біт WREN був встановлений в попередніх командах, тобто, біти WR, WREN не можуть встановлюватись в «1» однією командою. Біт WREN повинен бути скинутий у «0» після ініціалізації запису (на процес запису він не впливає).

Перед записом до EEPROM пам'яті має бути виконана обов'язкова послідовність команд, що запобігає випадковому запису. Обов'язкова послідовність виконується при вимкнених перериваннях.

Для запису даних до EEPROM пам'яті необхідно здійснювати таку послідовність дій:

1) якщо крок 10 не був виконаний, необхідно перевірити, чи не відбувається запис (WR=0);

2) записати адресу до регістра EEADR. Перевірте, чи записана адреса коректна для цього типу мікроконтролера;

3) записати 8-розрядне значення до регістра EEDATA;

4) скинути в «0» біт EEPGD для звернення до EEPROM пам'яті даних;

5) встановити біт WREN в «1», дозволивши запис до EEPROM пам'яті;

6) заборонити переривання, якщо їх дозволено;

7) виконати обов'язкову послідовність із п'яти команд:

- записати значення 55h до регістра EECON2 (дві команди, спочатку до W потім до EECON2);

- записати значення AAh до регістра EECON2 (дві команди, спочатку до W потім до EECON2);

- встановити біт WR в «1»;

8) дозволити переривання (якщо потрібно);

9) скинути біт WREN в «0»;

10) після завершення циклу запису скидається в «0» біт WR, встановлюється в «1» прапор переривання EEIF (скидається програмно), якщо крок 1 не виконується, необхідно перевірити стан бітів EEIF, WR перед початком запису.

Приклад програми запису в EEPROM пам’яті даних:

```
BSF     STATUS,RP1      ;
BSF     STATUS,RP0      ;Обрати банк 3
BTFSC   EECON1,WR       ;Перевірити завершення
GOTO $ - 1              ;операції запису
BCF     STATUS,RP0      ;Обрати банк 2
MOV F   ADDR,W          ;Записати адресу
MOVWF   EEADR           ;комірки пам’яті
MOVF    VALUE,W                 ;Вказати дані для запису
MOVWF   EEDATA          ;
BSF     STATUS,RP0      ;Обрати банк 3
BCF     EECON1,EEPGD    ;Обрати EEPROM пам’ять
BSF     EECON1,WREN     ;Дозволити запис в EEPROM пам’ять даних
BCF     INTCON,GIE      ;Заборонити переривання
MOVLW   0x55            ;Записати 55h в регістр EECON2
MOVWF   EECON2          ;
MOVLW   0xAA            ; Записати AAh в регістр EECON2
MOVWF   EECON2          ;
BSF     EECON1,WR       ;Ініціалізувати запис
BSF     INTCON,GIE      ;Дозволити переривання
BCF     EECON1,WREN     ;Заборонити запис в EEPROM пам’ять  ;даних
```

4.3.6. FLASH-пам'яті програм

Читання з FLASH-пам'яті програм.

Читання з FLASH-пам'яті програм дуже схоже на процедуру читання з EEPROM пам'яті даних, тільки необхідно виконати дві інструкції NOP після встановлення RD біта в «1». Два порожні цикли NOP використовуються мікроконтролером для читання даних із FLASH-пам'яті програм та збереження їх у регістрах EEDATH:EEDATA. Дані в регістрах будуть доступні після виконання другої інструкції NOP. Дані в регістрах

EEDATH:EEDATA зберігаються до наступної операції читання або запису в EEDATH:EEDATA.

Для читання даних з FLASH-пам'яті необхідно здійснювати таку послідовність дій:

1) записати адресу до регістрів EEADR: EEADRH. Перевірте, чи записана адреса коректна для цього типу мікроконтролера;

2) встановити в «1» біт EEPGD звернення до FLASH-пам'яті програм;

3) ініціалізувати операцію читання встановленням біта RD в «1»;

4) виконати дві команди NOP, щоб дозволити мікроконтролеру здійснити читання з FLASH-пам'яті програм;

5) прочитати дані з регістрів EEDATH: EEDATA.

Приклад програми читання з FLASH пам’яті даних:

```
BSF     STATUS,RP1      ;
BCF     STATUS,RP0      ;Обрати банк 2
MOV F   ADDRL,W         ;Записати адресу
MOVWF   EEADR           ;комірки пам’яті
MOV F   ADDRH,W         ;Записати адресу
MOVWF   EEADRH          ;комірки пам’яті
BSF     STATUS,RP0      ;Обрати банк 3
BSF     EECON1,EEPGD    ;Обрати FLASH пам’ять
BSF     EECON1,RD       ;Ініціалізувати читання
NOP                     ;Дві інструкції NOP
NOP
BCF     STATUS,RP0      ;Обрати банк 2
MOVF    EEDATA,W        ;DATAL = EEDATA
MOVWF   DATAL           ;
MOVF    EEDATAH,W               ;DATAH = EEDATAH
MOVWF   DATAH           ;
```

Запис до FLASH-пам'яті програм.

Протягом операції запису виконання програми зупиняється, тактовий генератор продовжує працювати, периферійні модулі включені та генерують переривання, які "становляться у чергу" до завершення циклу запису. Після завершення запису мікроконтролер продовжує виконувати код програми з місця зупинки. Іншою істотною відмінністю запису до FLASH-пам'яті програм є наявність біта захисту WRT в слові конфігурації, що запобігає будь-якому запису в пам’ять програм.

Запис даних до FLASH-пам'яті програм трохи складніше читання. Адреса комірки пам'яті програм і дані, що записуються, повинні бути поміщені у відповідні регістри спеціального призначення, біт EEPG встановлюється в «1». Біт WREN повинен завжди дорівнювати нулю, крім безпосереднього запису до FLASH-пам'яті програм. Біт WR може бути встановлений «1» тільки, якщо біт WREN був встановлений в попередніх командах, тобто біти WR, WREN не можуть встановлюватись в «1» однією командою. Біт WREN повинен бути скинутий програмно у «0» після ініціалізації запису (на процес запису він не впливає).

Перед записом до FLASH-пам'яті програм має бути виконана обов'язкова послідовність команд, що запобігає випадковому запису. Обов'язкова послідовність виконується при вимкнених перериваннях. Після обов'язкової послідовності повинні розміщуватись дві інструкції NOP, що дозволяють мікроконтролеру зробити запис. Виконання програми після запису починається з інструкції після двох команд NOP.

Для запису даних до FLASH-пам'яті необхідно здійснювати таку послідовність дій:

1) записати адресу до регістрів EEADR: EEADRH. Перевірити, чи записана адреса коректна для даного типу мікроконтролера;

2) записати 14-розрядне значення до регістрів EEDATH: EEDATA;

3) встановити в «1» біт EEPGD звернення до FLASH пам'яті програм;

4) встановити біт WREN в «1», дозволивши запис до FLASH-пам'яті програм;

5) заборонити переривання, якщо їх дозволено;

6) виконати обов'язкову послідовність із п'яти команд:

- записати значення 55h до регістра EECON2 (дві команди, спочатку до W потім до EECON2);

- записати значення AAh до регістра EECON2 (дві команди, спочатку до W потім до EECON2);

- встановити біт WR в «1»;

7) виконати дві команди NOP, щоб дозволити мікроконтролеру зробити запис до FLASH-пам'яті програм;

8) дозволити переривання (якщо потрібно);

9) скинути біт WREN «0».

Після завершення операції запису апаратно скидається в «0» біт WR і встановлюється в «1» прапор переривання EEIF (прапор EEIF скидається в

«0» програмно). Для закінчення операції записи перевіряти біт WR і EEIF необов'язково, тому що мікроконтролер не виконує програму під час запису у FLASH-пам'ять програм

Приклад програми запису до FLASH пам’яті програм:

```
BSF     STATUS,RP1      ;
BCF     STATUS,RP0      ;Обрати банк 2
MOV F   ADDRL,W         ;Записати адресу
MOVWF   EEADR           ;комірки пам’яті програм
MOV F   ADDRH,W         ;Записати адресу
MOVWF   EEADRH          ;комірки пам’яті програм
MOVF    VALUEL,W        ;Значення, яке записується
MOVWF   EEDATA          ;в пам’ять програм
MOVF    VALUEH,W        ;
MOVWF   EEDATAH         ;
BSF     STATUS,RP0      ;Обрати банк 3
BSF     EECON1,EEPGD    ;Обрати FLASH пам’ять програм
BSF     EECON1,WREN     ;Дозволити запис в FLASH пам’ять програм
BCF     INTCON,GIE      ;Заборонити переривання
MOVLW   0x55            ;Записати 55h в регістр EECON2
MOVWF   EECON2          ;
MOVLW   0xAA            ; Записати AAh в регістр EECON2
MOVWF   EECON2          ;
BSF     EECON1,WR       ;Ініціалізувати запис
NOP                     ;
NOP                     ;
BSF     INTCON,GIE      ;Дозволити переривання
BCF     EECON1,WREN     ;Заборонити запис в FLASH пам’ять програм
```

Доступність операцій читання/запису FLASH пам’яті програм залежно від стану бітів захисту наведена в таблиці 4.6. Регістри та біти, що пов’язані зі зверненням до EEPROM пам’яті даних/FLASH-пам’яті програм, наведені в таблиці 4.7.

Таблиця 4.6

Біти захисту FLASH пам’яті програм

Біти конфігурації			Область пам’яті	Внутрішнє читання	Внутрішній запис	ICSP читання	ICSP запис
CP1	CP0	WRT					
0	0	x	Уся пам’ять програм	Є	Немає	Немає	Немає
0	1	0	Незахищена область	Є	Немає	Є	Немає
0	1	0	Захищена область	Є	Немає	Немає	Немає
0	1	1	Незахищена область	Є	Є	Є	Немає
0	1	1	Захищена область	Є	Немає	Немає	Немає

Продовження табл. 4.6

Біти конфігурації			Область пам'яті	Внутрішнє читання	Внутрішній запис	ICSP читання	ICSP запис
CP1	CP0	WRT					
1	0	0	Незахищена область	Є	Немає	Є	Немає
1	0	0	Захищена область	Є	Немає	Немає	Немає
1	0	1	Незахищена область	Є	Є	Є	Немає
1	0	1	Захищена область	Є	Немає	Немає	Немає
1	1	0	Уся пам'ять програм	Є	Немає	Є	Є
1	1	1	Уся пам'ять програм	Є	Є	Є	Є

Таблиця 4.7

Регістри та біти, що пов'язані зі зверненням до EEPROM пам'яті даних/FLASH-пам'яті програм

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh 8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Dh	PIR2	-	(1)	-	EEIF	BCLIF	-	-	CCP2IF	- r - 0 0 - - 0	- r - 0 0 - - 0
8Dh	PIE2	-	(1)	-	EEIE	BCLIE	-	-	CCP2IE	- r - 0 0 - - 0	- r - 0 0 - - 0
10Dh	EEADR	Регістр адреси, молодший байт								xxxx xxxx	uuuu uuuu
10Fh	EEADRH	-	-	-	Регістр адреси, старший байт					xxxx xxxx	uuuu uuuu
10Ch	EEDATA	Регістр даних, молодший байт								xxxx xxxx	uuuu uuuu
10Eh	EEDATH	-	-	Регістр даних, старший байт						xxxx xxxx	uuuu uuuu
18Ch	EECON1	EEPGD	-	-	-	WRERR	WREN	WR	RD	x - - - x000	x - - - u000
18Dh	EECON2	Регістр управління 2 (фізично не реалізований)								-	-

Позначення: - – не застосовується, читається як 0; u – не змінюється; x – не відомо; q – залежить від умов.

Примітка. Резервні біти. При зверненні завжди мають бути дорівнені нулю.

Контрольні запитання та завдання

1. У чому полягає відмінність мікроконтролера від мікропроцесора?
2. Чим відрізняються виводи мікропроцесора та мікроконтролера?
3. Опишіть склад пам'яті мікроконтролера.
4. Яким чином побудована пам'ять програм мікроконтролера?
5. Яким чином побудована пам'ять даних мікроконтролера?
6. Навіщо потрібні регістри спеціального призначення?
7. Зі скількох банків складається пам'ять даних?
8. .Які регістри застосовуються при роботі з EEPROM та FLASH пам'яттю?
9. В чому полягає програмний тип захисту EEPROM і FLASH пам'яті?
10. Який порядок читання даних з EEPROM пам'яті?

4.4. Система команд

4.4.1. Опис полів команд та формат команд

Кожна команда мікроконтролера PIC16F87X складається з одного 14-розрядного слова, поділеного на код операції (OPCODE), що визначає тип команди, та один або кілька операндів, що визначають операцію команди. Команди розділені на такі групи:

- байт орієнтовані команди;
- біт орієнтовані команди;
- команди управління та операцій із константами.

Опис полів команд наведено у табл. 4.8, а формат команд – на рис. 4.4 [4].

Таблиця 4.8

Опис полів коду операції

Поле	Опис
f	Адреса регістра (від 0x00 до 0x7F)
w	Робочий регістр (акумулятор)
b	Номер біта у 8-розрядному регістрі
k	Константа (дані або мітка)
x	Не має значення (0 або 1). Асемблер генерує x=0 для сумісності програми мікроконтролера з інструментальними засобами
d	Вказівник адреси результату операції: d=0 – результат зберігається в регістрі W d=1 – результат зберігається в регістрі f За замовчуванням d=1
Label	Ім’я мітки
TOS	Вершина стеку
PC	Лічильник команд
PCLATH	Буфер старшого байту лічильника команд
GIE	Біт глобального дозволу переривань
WDT	Вартовий таймер
-TO	Прапор переповнення WDT
-PD	Прапор скидання за ввімкненням живлення
Dest	Приймач, регістр W або регістр пам’яті
[ ]	Додаткові параметри
{ }	Вміст
→	Привласнення
< >	Бітове поле
є	З набору
Курсив	Термін, визначений користувачем

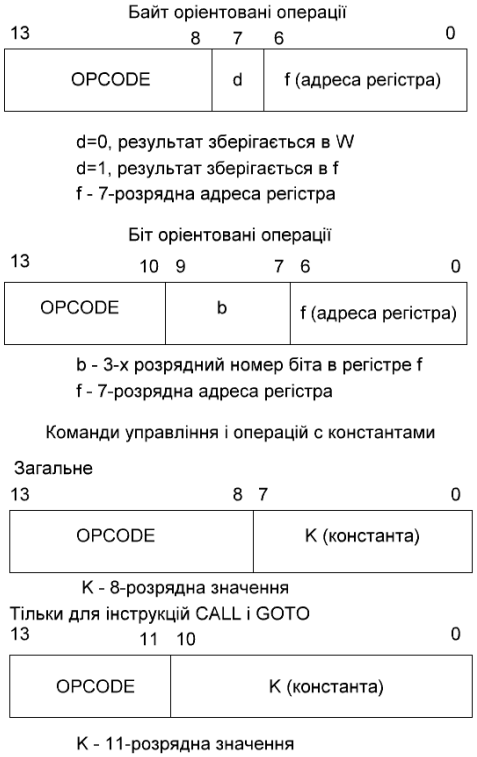


Рис. 4.4. Формат команд мікроконтролера [4]

Для байт орієнтованих команд "f" є покажчиком регістру, а "d" покажчиком адресата результату. Покажчик регістру визначає, який регістр має використовуватись у команді. Вказівник адресата визначає, де буде збережено результат. Якщо "d"=0, результат зберігається в регістрі W. Якщо "d"=1, результат зберігається в регістрі, який використовується в команді.

У біт орієнтованих командах "b" визначає номер біта, що задіяний у операції, а "f" – покажчик регістру, який містить цей біт.

У командах управління чи операціях з константами "k" представляє вісім чи одинадцять біт константи чи значення літералів.

Усі команди виконуються за один машинний цикл, крім команд умови, в яких отримано справжній результат та інструкції, що змінюють значення лічильника команд PC. У разі виконання команди за два машинні цикли, у другому циклі виконується інструкція NOP. Один машинний цикл складається із чотирьох тактів генератора. Для тактового генератора із частотою 4 МГц всі команди виконуються за 1 мкс, якщо команди умовні і умова виконується або змінюється лічильник команд PC, команда виконується за 2 мкс.

4.4.2. Перелік команд

Система команд мікроконтролера PIC16F87x наведена у табл. 4.9 [4].

Таблиця 4.9

Система команд мікроконтролера PIC16F87x

Мнемоніка команди	Опис	Циклів	14-розрядний код Біт 13 Біт 0	Змін. прапорів	Прим.
Байт орієнтовані команди					
ADDWF f,d	Додати W до f	1	00 0111 dfff ffff	C, DC, Z	1,2
ANDWF f,d	Побітне І W і f	1	00 0101 dfff ffff	Z	1,2
CLRF f	Очистити f	1	00 0001 1fff ffff	Z	2
CLRW	Очистити W	1	00 0001 0xxx xxxx	Z	
COMF f,d	Інвертувати f	1	00 1001 dfff ffff	Z	1,2
DECF f,d	Відняти 1 з f	1	00 0011 dfff ffff	Z	1,2
DECFSZ f,d	Відняти 1 з f і пропустити, якщо 0	1(2)	00 1011 dfff ffff		1,2,3
INCF f,d	Додати 1 до	1	00 1010 dfff ffff	Z	1,2
INCFSZ f,d	Додати1 до f і пропустити якщо 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF f,d	Побітне АБО W і f	1	00 0100 dfff ffff	Z	1,2
MOVF f,d	Переслати f	1	00 1000 dfff ffff	Z	1,2
MOVWF f	Переслати W до f	1	00 0000 dfff ffff		
NOP	Немає операції	1	00 0000 0xx0 0000		
RLF f,d	Циклічний зсув f ліворуч через перенос	1	00 1101 dfff ffff	C	1,2
RRF f,d	Циклічний зсув f праворуч через перенос	1	00 1100 dfff ffff	C	1,2
SUBWF f,d	Відняти W з f	1	00 0010 dfff ffff	C, DC, Z	1,2
SWAPF f,d	Поміняти місцями напівбайти в регістрі f	1	00 1110 dfff ffff		1,2
XORWF f,d	Побітне виключне АБО W і f	1	00 0110 dfff ffff	Z	1,2

Продовження табл. 4.9

Мнемоніка команди	Опис	Циклів	14-розрядний код Біт 13 Біт 0	Змін. прапорів	Прим.
Біт орієнтовані команди					
BCF f,b	Очистити біт в регістрі f	1	01 00bb bfff ffff		1,2
BSF f,b	Встановити біт в регістрі f	1	01 01bb bfff ffff		1,2
BTFSC f,b	Перевірити біт в регістрі f, пропустити якщо 0	1(2)	01 10bb bfff ffff		3
BTFSS f,b	Перевірити біт в регістрі f, пропустити якщо 1	1(2)	01 11bb bfff ffff		3
Команди управління і операцій з константами					
ADDLW k	Додати константу до W	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW k	Побітне І константи і W	1	11 1001 kkkk kkkk	Z	
CALL k	Виклик підпрограми	2	11 0kkk kkkk kkkk		
CLRWDT	Очистити WDT	1	00 0000 0110 0100	-TO,-PD	
GOTO k	Безумовний перехід	2	10 1kkk kkkk kkkk		
IORLW k	Побітне АБО константи і W	1	11 1000 kkkk kkkk	Z	
MOVLW k	Переслати константу до W	1	11 00xx kkkk kkkk		
RETFIE	Повернення з підпрограми з дозволом переривань	2	00 0000 0000 1001		
RETLW k	Повернення з підпрограми з завантаженням константи в W	2	11 01xx kkkk kkkk		
RETURN	Повернення з підпрограми	2	00 0000 0000 1000		
SLEEP	Перейти до режиму SLEEP	1	00 0000 0110 0011	-TO,-PD	
SUBLW k	Відняти W з константи	1	11 110x kkkk kkkk	C, DC, Z	
XORLW k	Побітне виключне АБО константи і W	1	11 1010 kkkk kkkk	Z	

Примітки:1. При виконанні операції «читання – модифікація – запис» з портом вводу-виводу початкові значення зчитуються з виводів порту, а не з вихідних заскочок. Наприклад, якщо в вихідний заскочці була записана «1», а на відповідному виході низький рівень сигналу, то знову буде записано значення «0».

2. При виконанні запису в TMR0 (і d=1) переддільник TMR0 скидається, якщо він підключений до модуля TMR0.

3.Якщо умова істинна або змінюється значення лічильника команд PC, то інструкція виконується за два цикли. В другому циклі виконується команда NOP.

Контрольні запитання та завдання

1. На які групи команд поділяється система команд?
2. Які арифметичні команди застосовуються в системі команд?
3. Які логічні команди застосовуються в системі команд?
4. Які команди пересилання застосовуються в системі команд?
5. Що означає літера b в полі команд?
6. Що означає літера d в полі команд?
7. Які команди умовних переходів є в системі команд?
8. Які команди застосовуються при роботі з підпрограмами?
9. Що означає літера w в полі команд?
10. Яке призначення має літера k?

4.5. Переривання в мікроконтролерах

Мікроконтролери PIC16F877 мають 14 джерел переривань. Регістр INTCON містить прапори окремих переривань, біти дозволу цих переривань та біт глобального дозволу переривань.

Якщо біт GIE (INTCON<7>) встановлений в «1», дозволені усі переривання, що не маскуються. Якщо GIE=0, усі переривання заборонені. Кожне переривання окремо може бути дозволене/заборонене установкою/скиданням відповідного біта в регістрах INTCON, PIE1 та PIE2. При скиданні мікроконтролера біт GIE скидається в «0».

При поверненні з підпрограми обробки переривань, за командою RETFIE, біт GIE апаратно встановлюється в «1», дозволяючи усі немасковані переривання.

У регістрі INTCON знаходяться прапори переривань зовнішнього сигналу INT, зміни рівня сигналу на входах RB7: RB4 та переповнення TMR0.

У регістрах PIR1, PIR2 містяться прапори переривань периферійних модулів мікроконтролера, а в регістрах PIE1, PIE2 – відповідні біти дозволу переривань. У регістрі INTCON знаходиться біт дозволу переривань від периферійних модулів.

При переході на підпрограму обробки переривань біт GIE апаратно скидається в «0», забороняючи переривання. При цьому адреса повернення з підпрограми обробки переривань переміщується до стеку, а до лічильника команд PC завантажується вектор переривання 004h.

Джерело переривань може бути визначене перевіркою прапорів переривань, які мають бути скинуті програмно перед дозволом переривання, щоб уникнути повторного виклику.

Для зовнішніх джерел переривань (сигнал INT, зміни рівня сигналу на входах RB7:RB4) час переходу на підпрограму обробки переривань становить 3-4 машинні цикли. Точний час переходу залежить від конкретної нагоди, він є однаковим для 1-циклових та 2-циклових команд. Прапори переривань встановлюються незалежно від стану відповідних бітів маски та біта GIE.

Індивідуальні прапори переривань встановлюються незалежно від стану відповідних бітів маски та біта GIE.

На рис. 4.5 наведена структурна схема логіки переривання.

Для реалізації переривань в програмі необхідно застосувати директиву вектора переривань «org 0×04», що в програмі будуть застосовуватись переривання, та вказати перехід на підпрограму переривань «goto prer» (рис. 4.6).

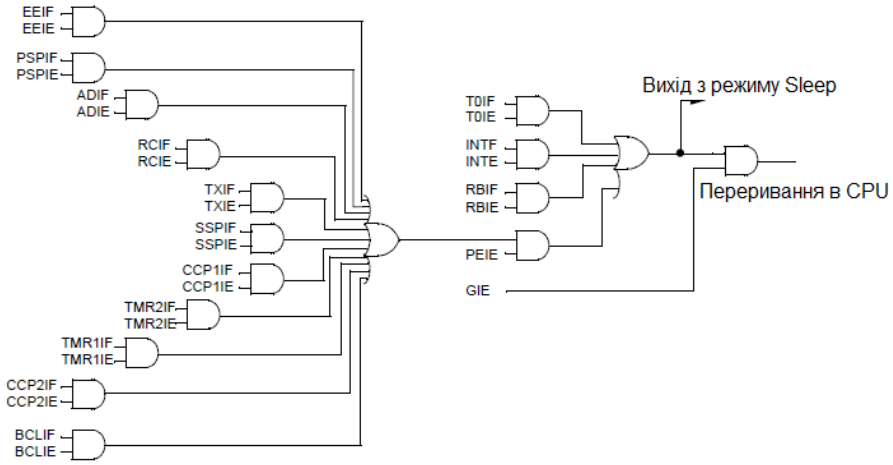


Рис. 4.5. Структурна схема логіки переривання [4]

```
        list p=16f877a
        #include <p16f877a.inc>
tempa   equ h'020'
tempb   equ h'021'
        org 0x00
        goto begin
        org 0x04; переривання
        goto prer; перехід на переривання
        org 0x100
begin   clrw
        clrf    tempa
```

Рис. 4.6. Приклад програми з застосуванням вектора переривань [4]

4.5.1. Регістри, які застосовуються системою переривань

Опис регістрів, що застосовуються системою переривань наведений у табл. 4.10 – 4.16.

Таблиця 4.10

Регістр INTCON

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Біт 7	GIE: Глобальний дозвіл переривань 1 - дозволені всі немасковані переривання 0 - всі переривання заборонені
Біти 6	PEIE:Дозвіл переривань від периферійних модулів 1 - дозволені всі немасковані переривання від периферійних модулів 0 - переривання від периферійних модулів заборонені
Біт 5	TOIE: Дозвіл переривань з переповнення TMR0 1 - переривання дозволено 0 - переривання заборонено
Біт 4	INTE: Дозвіл зовнішнього переривання INT 1 - переривання дозволено 0 - переривання заборонено
Біт 3	RBIE: Дозвіл переривання по зміненню сигналу на входах RB7:RB4 PORTB 1 - переривання дозволено 0 - переривання заборонено
Біт 2	TOIF: Прапор переривання по переповненню TMR0 1 - відбулося переповнення TMR0 (скидається програмно) 0 - переповнення TMR0 не було
Біт 1	INTF: Прапор зовнішнього переривання INT 1 - виконано умову зовнішнього переривання на виводі RB0/INT (скидається програмно) 0 не було зовнішнього переривання
Біт 0	RBIF: Прапор переривання по зміненню сигналу на входах RB7:RB4 PORTB 1 - зафіксовано змінення рівня сигналу на одному з входів RB7:RB4 (скидається програмно) 0 - не було змінення рівня сигналу ни на одному з входів RB7:RB4

Таблиця 4.11

Регістр OPTION_REG

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
-RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Біт 7	-RBPU: Підключення підтягуючих резисторів на входах PORTB 1 - підтягуючі резистори відключені 0 - підтягуючі резистори підключені
Біти 6	INTEDG: Вибір активного фронту сигналу на вході зовнішнього переривання 1 - переривання по фронту сигналу 0 - переривання по зрізу сигналу
Біт 5	TOCS: Вибір тактового сигналу для TMR0 1 - зовнішній тактовий сигнал з вивода RA4/TOCKI 0 - внутрішній тактовий сигнал CLKOUT
Біт 4	TOSE: Вибір фронту збільшення TMR0 при зовнішньому тактовому сигналі 1 - збільшення по зрізу сигналу (з високого до низького рівня) на виводі RA4/TOCI 0 - збільшення по фронту сигналу (з низького до високого рівня) на виводі RA4/TOCI

Продовження табл. 4.11

<table>
<tr><td>Біт 3</td><td>PSA: Вибір включення переддільника
1 - переддільник включений перед WDT
0 - переддільник включений перед TMR0</td></tr>
<tr><td>Біт 2-0</td><td>PS2:PS0: Встановлення коефіцієнта ділення переддільника
<table>
<tr><th>Значення</th><th>Для TMR0</th><th>Для WDT</th></tr>
<tr><td>000</td><td>1:2</td><td>1:1</td></tr>
<tr><td>001</td><td>1:4</td><td>1:2</td></tr>
<tr><td>010</td><td>1:8</td><td>1:4</td></tr>
<tr><td>011</td><td>1:16</td><td>1:8</td></tr>
<tr><td>100</td><td>1:32</td><td>1:16</td></tr>
<tr><td>101</td><td>1:64</td><td>1:32</td></tr>
<tr><td>110</td><td>1:128</td><td>1:64</td></tr>
<tr><td>111</td><td>1:256</td><td>1:128</td></tr>
</table></td></tr>
<tr><td>Примітка</td><td>При застосуванні режиму низьковольтного програмування і підключених підтягуючих резисторах на PORTB необхідно скинути в «0» 3-й біт регістра TRISB для відключення підтягуючого резистора на виводі RB3.</td></tr>
</table>

Таблиця 4.12

Регістр PIE1

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	**Біт 1**	Біт 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
Біт 7	PSPIE: Дозвіл переривання запису/читання веденого паралельного порту 1 - переривання дозволено 0 - переривання заборонено						
Біти 6	ADIE:Дозвіл переривання по закінченню перетворення АЦП 1 - переривання дозволено 0 - переривання заборонено						
Біт 5	RCIE: Дозвіл переривання від приймача USART 1 - переривання дозволено 0 - переривання заборонено						
Біт 4	TXIE: Дозвіл переривання від передавача USART 1 - переривання дозволено 0 - переривання заборонено						
Біт 3	SSPIE: Дозвіл переривання від модуля синхронного послідовного порту 1 - переривання дозволено 0 - переривання заборонено						
Біт 2	CCP1IE: Дозвіл переривання від модуля CCP1 1 - переривання дозволено 0 - переривання заборонено						
Біт 1	TMR2IE: Дозвіл переривання по переповненню TMR2 1 - переривання дозволено 0 - переривання заборонено						
Біт 0	TMR1IE: Дозвіл переривання по переповненню TMR1 1 - переривання дозволено 0 - переривання заборонено						
Примітка	Біт PSPIE в мікроконтролері PIC16F873/876 не реалізований, завжди має дорівнювати 0						

Таблиця 4.13

Регістр PIR1

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

Біт 7	PSPIF: Прапор переривання від веденого паралельного порту 1 - відбулася операція читання або запису (скидається програмно) 0 - не відбулася операція читання або запису
Біт 6	ADIF:Прапор переривання від модуля АЦП 1 - перетворення АЦП закінчено 0 - перетворення АЦП не закінчено
Біт 5	RCIF: Прапор переривання від приймача USART 1 - буфер приймача USART повний 0 - буфер приймача USART порожній
Біт 4	TXIF: Прапор переривання від передавача USART 1 - буфер передавача USART повний 0 - буфер передавача USART порожній
Біт 3	SSPIF: Прапор переривання від модуля MSSP 1 - виконано умову появи переривання від модуля MSSP (скидається програмно) Умова появи переривання -SPI - виконано прийом/передачу даних -Ведений I2C - виконано прийом/передачу даних -Ведучий I2C - виконано прийом/передачу даних - закінчено формування на шині біта START - закінчено формування на шині біта STOP - закінчено формування на шині біта повторний START - закінчено формування на шині біта підтвердження - виявлено на шині формування біта START (для режиму з декількома ведучими) - виявлено на шині формування біта STOP (для режиму з декількома ведучими) 0 - умова виникнення переривання від модуля MSSP не виконано
Біт 2	CCP1IF: Прапор переривання від модуля CCP1 <u>Режим захвату</u> 1 - виконано захват значення TMR1 (скидається програмно) 0 - захвата значення TMR1 не відбулося <u>Режим порівняння</u> 1 - значення TMR1 досягло вказаного в регістрах CCPR1H:CCPR1L (скидається програмно) 0 - значення TMR1 не досягло вказаного в регістрах CCPR1H:CCPR1L <u>ШИМ режим</u> не застосовується
Біт 1	TMR2IF: Прапор переривання по переповненню TMR2 1 - відбулося переповнення TMR2 (скидається програмно) 0 - переповнення TMR2 не було
Біт 0	TMR1IF: Прапор переривання по переповненню TMR1 1 - відбулося переповнення TMR1 (скидається програмно) 0 - переповнення TMR2 не було
Примітка	Біт PSPIF в мікроконтролері PIC16F873/876 не реалізований, завжди має дорівнювати 0

Таблиця 4.14

PIE2

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	Резерв	-	EEIE	BCLIE	-	-	CCP2IE

Біт 7	Не реалізований, читається як «0»
Біти 6	Резерв: завжди має дорівнювати «0»
Біт 5	Не реалізований, читається як «0»
Біт 4	EEIE: Дозвіл переривання по закінченню запису EEPROM даних 1 - переривання дозволено 0 - переривання заборонено
Біт 3	BCLIE: Дозвіл переривання при виникненні колізії на шині 1 - переривання дозволено 0 - переривання заборонено
Біт 2-1	Не реалізовані, читається як «0»
Біт 0	CCP2IE: Дозвіл переривання від модуля CCP2 1 - переривання дозволено 0 - переривання заборонено

Таблиця 4.15

PIR2

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	Резерв	-	EEIE	BCLIE	-	-	CCP2IE

Біт 7	Не реалізований, читається як «0»
Біти 6	Резерв: завжди має дорівнювати «0»
Біт 5	Не реалізований, читається як «0»
Біт 4	EEIF: Прапор переривання по закінченню запису EEPROM даних 1 - запис в EEPROM даних закінчений (скидається програмно) 0 - запис в EEPROM даних не закінчений або на була почата
Біт 3	BCLIF: Прапор переривання при виникненні колізії на шині 1 - на шині виявлено колізії (тільки в режимі ведучого I2C) 0 - колізій не виявлено
Біт 2-1	Не реалізовані, читається як «0»
Біт 0	CCP2IF: Прапор переривання від модуля CCP2 <u>Режим захвата</u> 1 - виконано захват значення TMR1 (скидається програмно) 0 - захвата значення TMR1 не було <u>Режим порівняння</u> 1 - значення TMR1 досягло вказаного в регістрах CCPR2H:CCPR2L (скидається програмно) 0 - значення TMR1 не досягло вказаного в регістрах CCPR2H:CCPR2L <u>ШИМ режим</u> Не застосовується

Таблиця 4.16

PCON

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
-	-	-	-	-	-	-POR	-BOR
Біт 7-2	Не реалізований, читаються як «0»						
Біт 1	-POR:Прапор скидання по увімкненню живлення 1 - скидання по увімкненню живлення не було 0 - відбулося скидання мікроконтролера по увімкненню живлення (програмно має бути встановлений в 1 для виявлення скидання POR)						
Біт 0	-BOR: Прапор скидання по зниженню напруги живлення 1 - скидання по зниженню напруги живлення не було 0 - відбулося скидання мікроконтролера по зниженню напруги живлення (програмно має бути встановлений в 1 для виявлення скидання BOR)						

4.5.2. Зовнішнє переривання зі входу RB0/INT

Зовнішнє переривання зі входу RB0/INT відбувається:

- за фронтом сигналу, якщо біт INTEDG (OPTION_REG<6>) встановлений в «1»;

- за зрізом сигналу, якщо біт INTEDG скинутий в «0».

Коли активний фронт сигналу з'являється на вході RB0/INT біт INTF (INTCON<1>) встановлюється в «1». Переривання може бути заборонено скиданням біту INTE (INTCON<4>) в «0».

Прапор переривання INTF повинен бути скинутий програмно у підпрограмі обробки переривань. Переривання INT може вивести мікроконтролер з режиму SLEEP, якщо біт INTE=1 до переходу у режим SLEEP. Стан біта GIE визначає, чи переходити на підпрограму обробки переривань після виходу з режиму SLEEP.

4.5.3. Переривання за переповненням TMR0

Переповнення таймера TMR0 (FFh → 00h) встановлює прапор TOIF (INTCON<2>) в «1». Переривання від TMR0 можна дозволити/заборонити установкою/скиданням біта TOIE (INTCON<5>). Опис роботи модуля TMR0 наведено раніше.

4.5.4. Переривання через зміну рівня сигналу на входах RB7:RB4

Зміна рівня сигналу на входах RB7:RB4 викликає встановлення прапора RBIF(INTCON<0>). Переривання можна дозволити/заборонити установкою/скиданням біта RBIE(INTCON<4>).

4.5.5. Збереження контексту при обробці переривань

При переході на підпрограму обробки переривань у стеку зберігається лише адреса повернення. Як правило, необхідно зберігати значення ключових регістрів при обробці переривань (наприклад, регістр W та STATUS), що виконується програмним способом.

Для PIC16F873/874 регістр W_TEMP має бути визначений в обох банках (0, 1) з однаковим зсувом щодо базової адреси банку (тобто якщо регістр W_TEMP визначено у банку 0 з адресою 0×20, то він має бути визначений у банку 1 з адресою 0×A0). Регістри PCLATH_TEMP та STATUS_TEMP можуть бути визначені лише в одному банку.

Через те, що старші 16 байт кожного банку мікроконтролерів PIC16F876/877 доступні у всіх банках, регістри STATUS_TEMP, PCLATH_TEMP та W_TEMP можуть бути розміщені в цій області. Нижче наведено текст збереження контексту.

Приклад програми: Збереження та відновлення регістрів STATUS, W та PCLATH

```
MOVWF  W_TEMP            ; Зберегти W в регістрі поточного банку
SWAPF   STATUS,W         ; Змінити місцями напівбайти та зберегти у W
CLRF     STATUS          ; Обрати банк 0
MOVWF  STATUS_TEMP       ; Зберегти регістр STATUS
MOVF    PCLATH,W
MOVWF  PCLATH_TEMP       ; Зберегти регістр PCLATH
:
:                        ;  Код програми обробки переривань
:
MOVF   PCLATH_TEMP,W
MOVWF PCLATH             ; Відновити регістр PCLATH
SWAPF  STATUS_TEMP,W     ; Прочитати регістр STATUS_TEMP
                         ; в  W, та відновити банк пам'яті програм
MOVWF STATUS             ; Переписати W до регістру STATUS
SWAPF          W_TEMP,F  ; Обміняти місцями напівбайти в W_TEMP
SWAPF    W_TEMP,W        ; Обміняти місцями напівбайти в W_TEMP і
                         ; записати до W
```

Якщо допускаються вкладення переривання, необхідно мати маску пріоритетів з програмною установкою. Це дозволяє змінити пріоритет переривань, дозволених під час виконання підпрограми обслуговування. Регістр маски вважається портом введення/виведення і в нього можна записувати 3-бітну маску за допомогою команди виведення.

Контрольні запитання та завдання

1. Для чого потрібні переривання в обчислювальних системах?
2. Яка кількість переривань передбачена в мікроконтролері PIC16F877a?
3. Для чого призначений регістр INTCON?
4. Яким чином побудована система переривань в мікроконтролері?
5. Які переривання реалізовані в регістрі INTCON?
6. Для чого призначений біт GIE?
7. Яким чином реалізуються переривання від периферійних модулів?
8. Опишіть переривання за переповненням TMR0.
9. Опишіть переривання через зміну рівня сигналу на входах RB7:RB4.
10. Для чого необхідне збереження контенту при реалізації переривань?

5. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ. АРИФМЕТИЧНІ ОПЕРАЦІЇ

5.1. Регістри мікроконтролера

5.1.1. Регістр STATUS

Регістр STATUS містить прапори стану АЛУ, прапори причини скидання мікроконтролера та біти управління банками пам'яті даних.

Регістр STATUS може бути адресований будь-якою командою так саме, як інший регістр пам'яті даних. Якщо звернення до регістру STATUS виконується командою, яка впливає на прапори Z, DC та C, зміни цих трьох бітів командою заблоковано. Ці біти скидаються або встановлюються згідно з логікою ядра мікроконтролера. Команди зміни регістру STATUS впливають також на біти –TO та –PD. Тому результат виконання команди з регістром

STATUS може відрізнятись від очікуваного. Наприклад, команда CLRF STATUS скине три старші біти та встановить біт Z (стан регістру STATUS після виконання команди 000uu1uu, де u - біт, що не змінюється).

При зміні бітів регістру STATUS рекомендується використовувати команди, які не впливають на прапори АЛУ (SWAPF, MOVWF, BCF та BSF).

Прапори C та DC використовуються як біти позики та десяткового позичання відповідно, наприклад, при виконанні команд віднімання SUBLW та SUBWF.

5.1.2. Реєстр OPTION_REG

Регістр OPTION_REG доступний для читання та запису, містить біти управління (табл. 5.1):

- попереднім дільником TMR0/WDT;
- активним фронтом зовнішнього переривання RB0/INT;
- резисторами на входах PORTB, що підтягують;

Якщо попередній дільник увімкнений перед WDT, коефіцієнт поділу тактового сигналу для TMR0 дорівнює 1:1.

Таблиця 5.1

Регістр STATUS (адреса 03h, 83h, 163h, 183h)

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	-TO	-PD	Z	DC	C
Біт 7	IRP: Біт вибору банка при непрямої адресації 1 - банк 2,3 (100h – 1FFh) 0 - банк 0,1 (000h – 0FFh)						
Біти 6-5	RP1:RP0: Біти вибору банка при безпосередньої адресації 11 - банк 3 (180 h – 1FFh) 10 - банк 2 (100 h – 17Fh) 01 - банк 1 (080 h – 0FFh) 00 - банк 0 (000 h – 07Fh)						
Біт 4	-TO: Прапор переповнення вартового таймера 1 - після POR або виконання команд CLRWDT, SLEEP 0 - після переповнення WDT						
Біт 3	-PD: Прапор увімкнення живлення 1 - після POR або виконання команди CLRWDT 0 - після виконання команди SLEEP						

Продовження табл. 5.1

Біт 2	Z: Прапор нульового результату 1 - нульовий результат виконання арифметичної або логічної операції 0 - не нульовий результат виконання арифметичної або логічної операції
Біт 1	DC: Прапор десяткового переносу/позики (для команд ADDWF, ADDWL, SUBWF, SUBWL), позика має інверсне значення 1 - був перенос з молодшого напівбайта 0 - не було переносу з молодшого байта
Біт 0	C: Прапор переносу/позики (для команд ADDWF, ADDWL, SUBWF, SUBWL), позика має інверсне значення 1 - був перенос з старшого біта 0 - не було переносу з старшого біта

Примітка: Прапор позики має інверсне значення. Віднімання виконується шляхом додавання додаткового коду другого операнда. При виконанні команд зсуву (RRF, RLF) біт C завантажується старшим або молодшим бітом зсувного регістра.

5.1.3. Регістри PCLATH та PCL

13-розрядний регістр лічильника команд PC вказує адресу виконуваної інструкції.

Молодший байт лічильника команд PCL доступний для читання та запису. Старший байт PCH, що містить <12:8> біти лічильника команд PC, не доступний для читання та запису.

Усі операції з регістром PCH відбуваються через додатковий регістр PCLATH. За будь-якого виду скидання мікроконтролера лічильник команд PC очищується. На рис 5.1. наведені дві ситуації завантаження значення до лічильник команд PC. На верхньому прикладі запис до лічильника команд PC відбувається під час запису значення до регістра PCL (PCLATH <4:0>→PCH). На нижньому прикладі запис значення до лічильника команд PC відбувається за виконання команди CALL чи GOTO (PCLATH<4:3>→PCH).

Таблиця 5.2

Регістр OPTION_REG

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
-RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Біт 7	-RBPU: Підключення підтягуючих резисторів на входах PORTB 1 - підтягуючі резистори відключені 0 - підтягуючі резистори підключені
Біти 6	INTEDG: Вибір активного фронту сигналу на вході зовнішнього переривання 1 - переривання по фронту сигналу 0 - переривання по зрізу сигналу
Біт 5	TOCS: Вибір тактового сигналу для TMR0 1 - зовнішній тактовий сигнал з вивода RA4/TOCKI 0 - внутрішній тактовий сигнал CLKOUT
Біт 4	TOSE: Вибір умиви збільшення TMR0 при зовнішньому тактовому сигналі 1 - збільшення по зрізу сигналу (з високого до низького рівня) на виводі RA4/TOCI 0 - збільшення по фронту сигналу (з низького до високого рівня) на виводі RA4/TOCI
Біт 3	PSA: Вибір підключення переддільника 1 - переддільник підключений перед WDT 0 - переддільник підключений перед TMR0
Біт 2-0	PS2:PS0: Встановлення коефіцієнта ділення переддільника

Значення	Для TMR0	Для WDT
0 0	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Примітка: При застосуванні режиму низьковольтного програмування і підключених підтягуючих резисторах на PORTB необхідно скинути в «0» 3-й біт регістра TRISB для відключення підтягуючого резистора на виводі RB3.

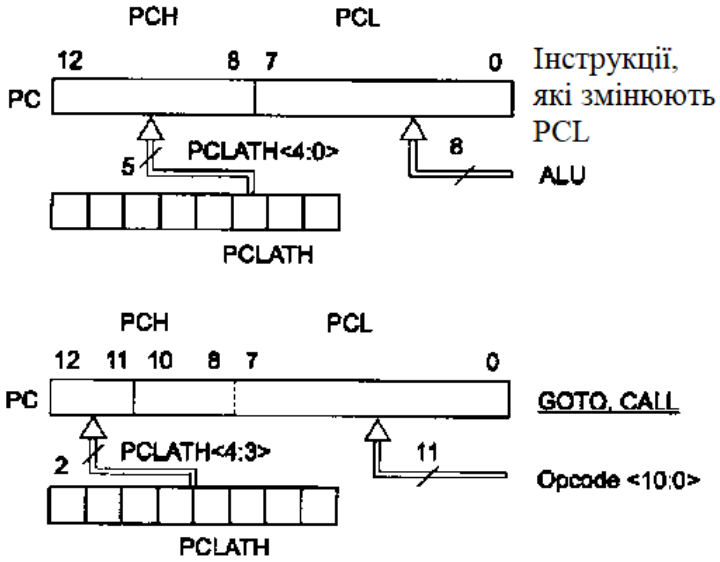


Рис. 5.1. Схема запису значень до лічильника команд [4]

Обчислюваний перехід. Перехід, що обчислюється, може бути виконаний командою збільшення до регістру PCL (наприклад, ADDWF PCL). При виконанні табличного читання переходом, що обчислюється, слід піклуватися про те, щоб значення PCL не перетнуло межу блоку пам'яті (кожний блок 256 байт).

Стек. Мікроконтролер PIC має 8-рівневий 13-розрядний апаратний стек. Стек не має відображення на пам’ять програм та пам'ять даних. Не можна записати або прочитати дані зі стека. Значення лічильника команд заноситься до вершини стека під час інструкцій переходу на підпрограму (CALL) чи обробки переривань. Читання зі стека та запис до лічильника команд PC відбувається при виконанні інструкцій повернення з підпрограми або обробки переривань (RETURN, RETLW, RETFIE), при цьому значення регістру PCLATH не змінюється.

Стек працює як циклічний буфер. Після 8 записів до стека дев'ятий запис запишеться на місце першого, а десятий запис замінить другий і так далі.

У мікроконтролерах немає жодних покажчиків про переповнення стека. Не передбачено також команд запису/читання зі стека, за виключенням команд виклику/повернення з підпрограм (CALL, RETURN, RETLW та RETFIE) або умов переходу за вектором переривань.

Сторінки пам'яті програм. Усі мікроконтролери здатні адресувати 8К слів пам'яті програм. Інструкції переходів (CALL та GOTO) мають 11-розрядне поле для вказівки адреси, що дозволяє безпосередньо адресувати 2К слів пам'яті програм. Для адресації верхніх сторінок програм використовуються 2 біти в регістрі PCLATH<4:3>. Перед виконанням команди переходу (CALL або GOTO) необхідно запрограмувати біти регістру PCLATH<4:3> для адресації потрібної сторінки.

При виконанні інструкцій повернення з підпрограми 13-розрядне значення для лічильника програм PC береться з вершини стека, тому маніпуляція бітами регістру PCLATH<4:3> не потрібна.

Вміст регістру PCLATH не змінюється на полі виконання інструкції RETURN або RETFIE. Користувач повинен сам змінити значення регістру PCLATH для подальшого виконання команд GOTO та CALL.

Приклад на рис. 5.2 ілюструє перехід зі сторінки 0 на сторінку 1 пам'яті програм. Цей приклад передбачає, що у підпрограмі зберігається відновлюється значення регістру PCLATH.

```
ORG   0x500
BSF   PCLATH,3    ;Вибір сторінки 1 (800h – FFFh)
CALL  SUB1_P1     ;Перехід до сторінки 1 (800h – FFFh)
:
:
ORG   0x900
SUB_P1:           ;Сторінка 1 (800h - FFFh)
RETURN            ;Повернення до сторінки 0 (000h – 7FFh)
```

Рис. 5.2. Приклад програми переходу зі сторінки 0 до сторінки 1 пам’яті програм [4]

5.1.4. Непряма адресація, регістри INDF і FSR

Для непрямої адресації необхідно звернутися до фізично не реалізованого регістру INDF. Звернення до регістру INDF фактично викличе дію з регістром, адреса якого вказана у FSR.

Непряме читання регістра INDF (FSR=0) дасть результат 00h.

Непрямий запис у регістр INDF не викликає жодних дій (але викликає вплив на прапори АЛУ в регістрі STATUS). 9-біт непрямої адреси IRP зберігається у регістрі STATUS<7>.

Приклад 9-розрядної непрямої адресації наведено на рис. 5.3.

Приклад програми непрямої адресації:

```
                BSF    STATUS,IRP ; Встановити банк 1
                MOVLW      0x20   ; Вказати перший регістр ОЗП
                MOVWF      FSR    ;
NEXT
                CLRF  INDF        ; Очистити регістр
                INCF   FSR,F      ; Збільшити адресу
                BTFSS FSR,4       ; Закінчити?
                GOTO NEXT         ; Ні, продовжити очищення
CONTINUE
                                  ; Так
```

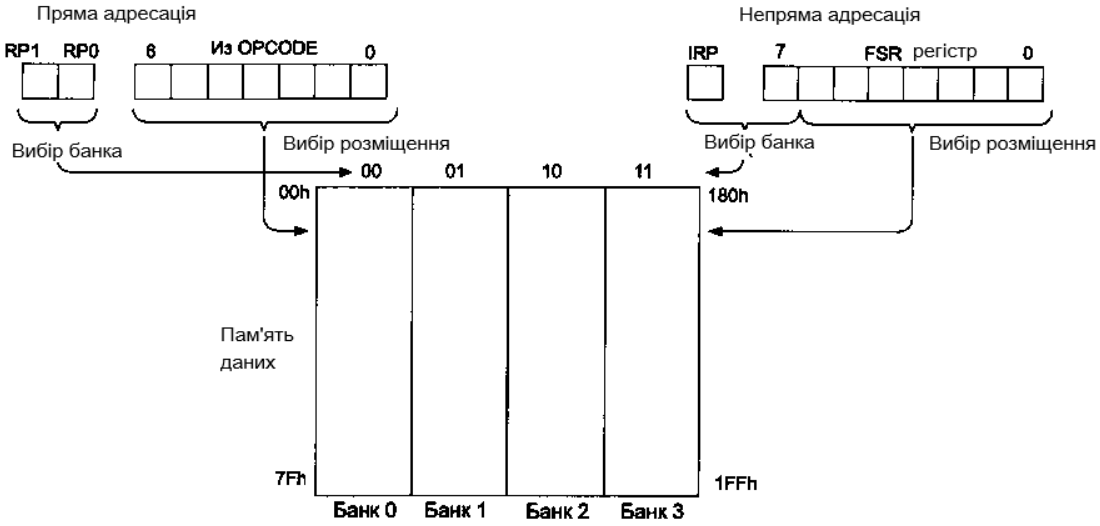


Рис. 5.3. Схема прямої та непрямої адресації [4]

Контрольні запитання та завдання

1. Для чого потрібні переривання в обчислювальних системах?
2. Яке призначення регістра STATUS?
3. Які ознаки результатів операцій має регістр STATUS?
4. Яким чином виконується перехід з банка до банку в пам’яті даних?
5. Яке призначення регістрів PCLATH, OPTION_REG, INDF та FCR?

5.2. Системи розробки програмного забезпечення мікроконтролерів

5.2.1. Системи розробки програмного забезпечення

Як системи розробки програмного забезпечення застосовуються пакети програмного забезпечення. Для мікроконтролерів компанії Microchip розроблений пакет програмного забезпечення MPLAB IDE.

MPLAB – інтегроване середовище розробки, яке уявляє собою набір програмних продуктів та призначене для облегшення процесу створювання, редактування та відладки програм для PIC мікроконтролерів компанії Microchip Technology. Середовище розробки складається із окремих додатків, пов'язаних один з одним, та містить в себе компілятор з мови асемблер, текстовий редактор, програмний симулятор та засоби роботи з проєктами. Також середовище дозволяє застосовувати компілятор С.

MPLAB 8.X працює під управлінням операційної системи Windows. Остання версія середовища розробки – MPLAB IDE v8.92.

MPLAB складається з таких основних модулів:

- MPLAB Project Manager – засоби роботи з проєктами;
- MPLAB-SIM Software Simulator – моделювання поведінки програми з метою пошуку та видалення помилок в алгоритмі;
- MPLAB Editor – повноцінний текстовий редактор файлів ASM;
- -MPASM Universal Macro Assembler – компілятор з асемблера, компонувальник;
- MPLAB ASM30 Macro Assembler – компілятор з асемблера, компонувальник для 16-бітних PIC- и dsPIC-мікроконтролерів;
- MPLAB-ICE 2000 – моделювання поведінки програми в реальному часі.

Процедура написання та відлагодження програм для PIC-микроконтроллерів із застосуванням системи розробки та відлагодження програм передбачає спочатку створення проєкту з подальшим включенням до нього початкових файлів програм, тобто програм, що написані у початкових кодах.

5.2.2. Створення проєкту

Для створення проєкту застосовується модуль MPLAB Project Wizard (Майстер проектів), в якому необхідно виконати таку послідовність дій:

1) запустити майстер проєктів обранням Project > Project Wizard, після чого на екрані з'явиться текст «Wellcome!» (рис.5.4);

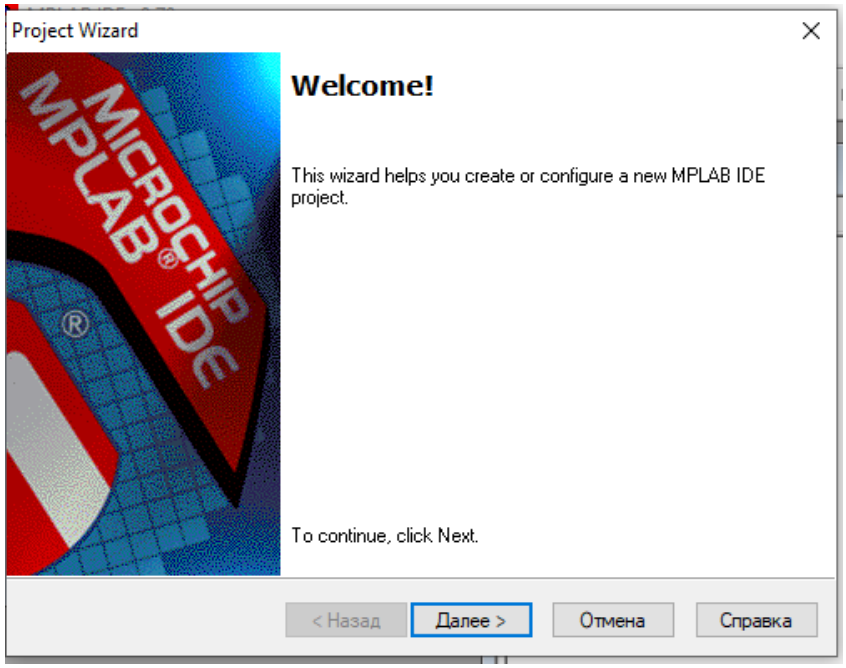


Рис.5.4. Вікно MPLAB Project Wizard

2) натиснути кнопку *Далі*, після чого у вікні, що відкриється, обрати зі списку пристроїв мікроконтролер *PIC18F877a*;

3) натиснути кнопку *Далі*, після чого у вікні вибору програмного забезпечення, що відкриється, підтвердити відображення *Microchip Toolsuite* у полі *Location*, або, якщо поле *Location* порожнє, розмістити у ньому *mpasmwin.exe* через *Browse*;

4) натиснути кнопку *Далі*, після чого у вікні введення проєкту, що відкриється, за допомогою *Browse* відкрити директорію, до якої ввести ім'я проекту та *зберегти* його (у вікні з'являться шлях та ім'я проєкту);

5) натиснути кнопку *Далі*, після чого у вікні, що відкриється, обрати (виділити) початковий файл;

6) натиснути кнопку *Далі*, після чого у вікні, що відкриється, має з'явитись повідомлення про те, що проект створено правильно;

7) натиснути кнопку *Finish* для виходу з майстра проектів, після чого на робочому столі з'явиться вікно проєкту (рис.5.5).

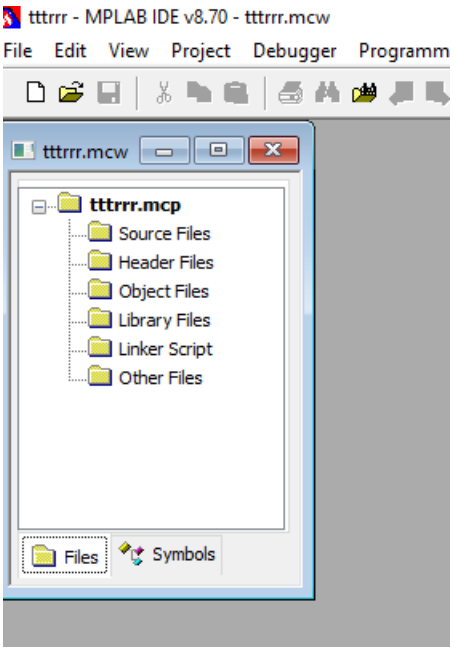


Рис. 5.5. Вікно проекту

За необхідності початкові файли можуть бути додані до проєкту або вилучені з нього застосуванням правої кнопки миші у вікні проєкту у полі *Source Files*.

5.2.3. Створення початкових програм

Для створення початкових кодів програм необхідно застосувати редактор текстів. Для цього слід обрати *File* > *New*, після чого у робочому просторі відкриється вікно редагування, до якого необхідно увести програму у кодах асемблера.

Після завершення введення кодів треба вибрати *File* > *Save* та зберегти файл у новий директорії, наприклад, *D:\MyProj\test.asm*. Файл повинен мати розширення *«.asm»*, шлях до файлу має бути написаний латиницею. Збережений текст програми відображається із застосуванням ідентифікуючих кольорів, які позначають коди, резервні слова, коментарі тощо.

5.2.4. Побудова або компіляція проєкту

По завершені створення проєкту необхідно здійснити його побудову, яка полягає в асемблюванні початкових кодів із застосуванням набору інструментів *Microchip MPASM toolsuite*.

Для цього необхідно обрати *Project* > *Build All*. Якщо файл асемблюється успішно, з’являється вікно з повідомленням «BUILD SUCCEEDED» (рис.5.6).

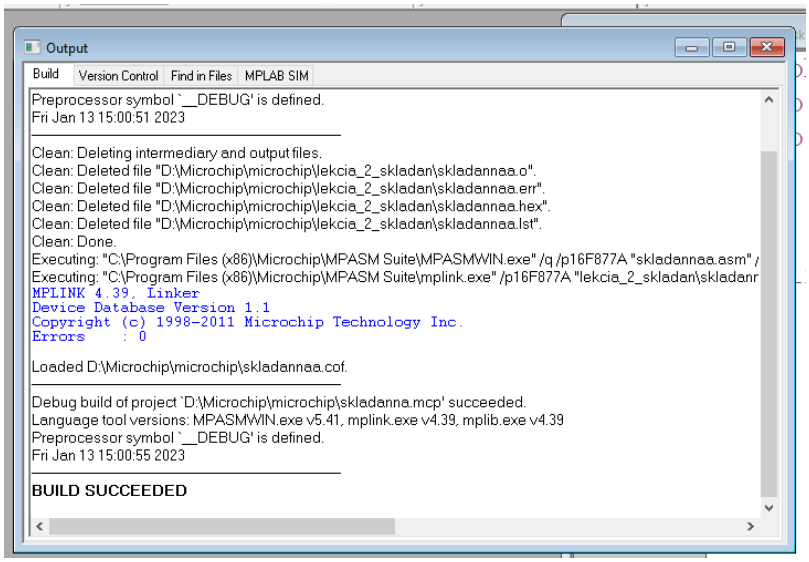


Рис.5.6. Вікно результатів успішного асемблювання початкового файлу

Якщо файл асемблюється не успішно, з'являється вікно з повідомленням «BUILD FAILED» (рис.5.7).

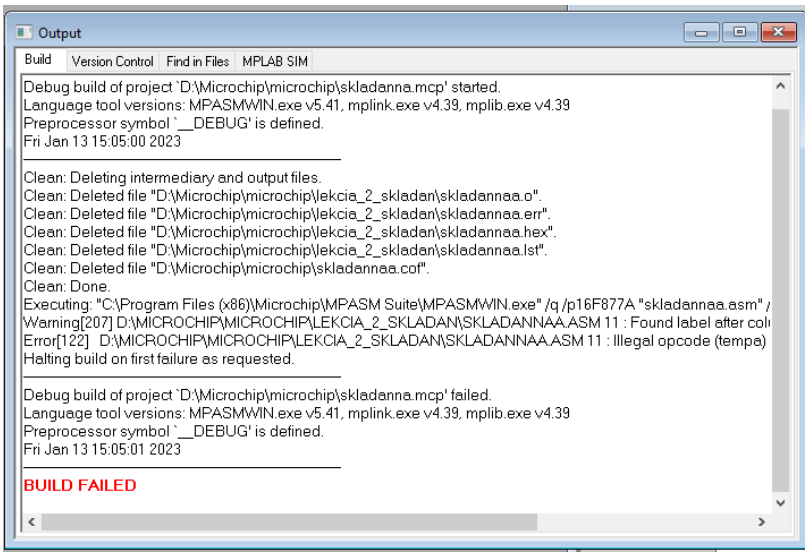


Рис.5.7. Вікно результатів неуспішного асемблювання початкового файлу

У цьому випадку необхідно переконатись у правільності застосовання відповідного асемблера та правильності синтаксису та формату кодів, що введені у вікні редактора.

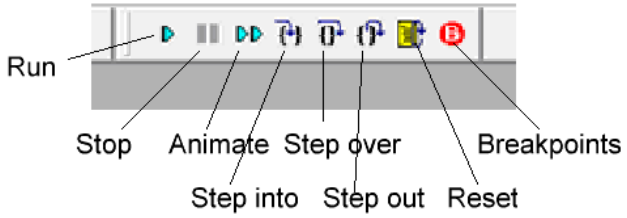
Якщо асемблер виводить повідомлення про помилки у вікні виходу, треба двічі клацнути на помилку і MPLAB відкриє вівдповідний рядок у початковому коді із зеленою стрілкою з лівого боку вікна початкових кодів.

Для цього оберіть *Project* > *Set Language Tool Locations*. Клацніть *MPASM Assembler* (*mpasmwin.exe*) та оберіть його розташування на дисплеі. Якщо розташування правильне, – клацнути *Cansel*, якщо ні, – змінити його і клацнути *OK*.

5.2.5. Налаштування проєкту

Після побудови проєкту, тобто після компіляції початкового тексту програми можна виконувати налаштування програми.

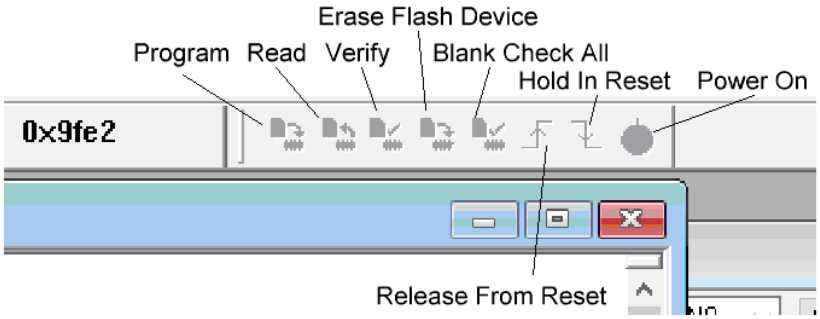
Для налаштування проєкту слід обрати *Debugger* > *Select Tool* > *MPLAB SIM*, після чого з'явиться панель інструментів (рис. 5.8).



Run – пуск програми; *Stop* – зупинка програми;
Animate – виконання програми з відображенням команди, що виконується;
Step into, Step over, Step out – покрокове виконання програми;
Reset – скидання програми; *Breakpoints* – призначення точок зупинки

Рис. 5.8. Інструменти для налаштування програми

В цьому режимі программа MPLAB SIM симулює виконання команд мікроконтролера, тому програми для роботи із зовнішними пристроями, які застосовують порти та АЦП, не можуть бути повністю налагоджені. Для них треба обирати інші типи налагоджувачів, наприклад, MPLAB ICD2, PICkit3, які завантажують програму до пам'яті мікроконтролера, який виконує команди..



Program - завантаження програми до мікроконтролера; *Read* – читання програми; *Verify* – перевірка; *Erase Flash Device* – стерти програму; *Blank Check All* – перевірка; *Release From Reset* – пуск програми; *Hold In Reset* – скидання; *Power On* – увімкнення

Рис. 5.9. Інструменти для завантаження та пуску програми

Після налаштування програма завантажується до мікроконтролера за допомогою завантажувача *Programer*. Для завантаження програми до мікроконтролера необхідно обрати *Programer > Select Programer > PICkit3*, після чого має з'явитись лінійка інструментів (рис. 5.9).

5.3. Асемблер. Система директив

Для програмування мікроконтролерів застосовується асемблер MPASM, який є безкоштовною універсальною програмою-компілятором початкового тексту програми, що написана на мові асемблер для мікроконтролерів PICmicro компанії Microchip Technology Incorporated.

Асемблер MPASM працює під керуванням ОС Windows на PC-сумісних комп'ютерах. MPASM забезпечує універсальний інструмент розробки програм для 12/14/16 – розрядних мікроконтролерів PICmicro.

Основні переваги асемблера MPASM:

- підтримка всіх інструкцій мікроконтролерів PICmicro;
- інтерфейс командного рядка;
- віконний інтерфейс;
- підтримка системи директив;
- підтримка макросів;
- сумісність с MPLAB IDE.

Асемблер забезпечує сумісність кодів програми. Тому що MPASM є універсальним рішенням для всіх типів мікроконтролерів PICmicro, то програма, яка написана для PIC16C54, може бути легко перенесена на PIC16C71. При перенесенні слід змінити інструкції, які пов'язані з апаратними особливостями мікроконтролерів, а інша частина директив та макрокоманд залишається без змінень.

Існує три версії MPASM:

- MPASM для MS DOS V5.0 та вище;
- розширена DOS версія;
- MPASM для Windows.

5.3.1. Стислий огляд асемблера

MPASM може застосовуватися в двох випадках:

- для генерації абсолютного кода, який може бути завантажений безпосередньо до мікроконтролера;

- для генерації об’єктних кодів, які зв’язуються з іншими компільованими модулями.

Абсолютний код – це режим роботи програми MPASM за замовчуванням.

Під час компіляції початкового файлу в цьому режимі усі значення мають бути вказані у початковому файлі або у файлах, які включаються. Якщо компіляція виконана без помилок, буде створений HEX-файл коду програми, який можна застосовувати для безпосереднього програмування мікроконтролера (рис.5.10).

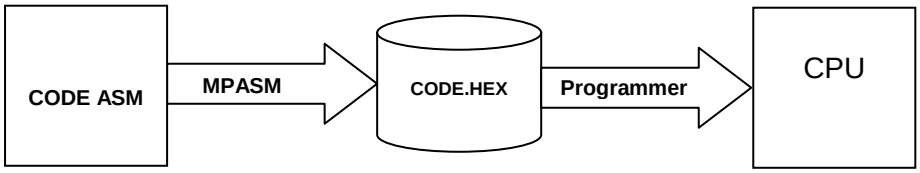


Рис.5.10. Схема компіляції початкового файла

MPASM має можливість генерувати об’єктні модулі, які можуть бути пов’язані один з одним із застосуванням лінкера MPLINK для остаточного формування коду, що виконується (абсолютного коду). Даний метод дозволяє багатократно застосовувати видлагоджені модулі програми (рис.5.11).

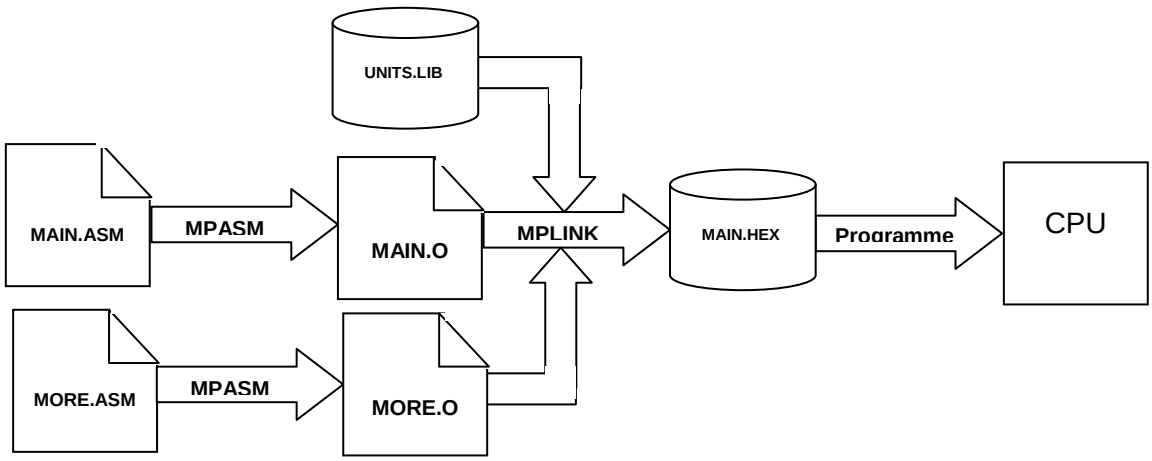
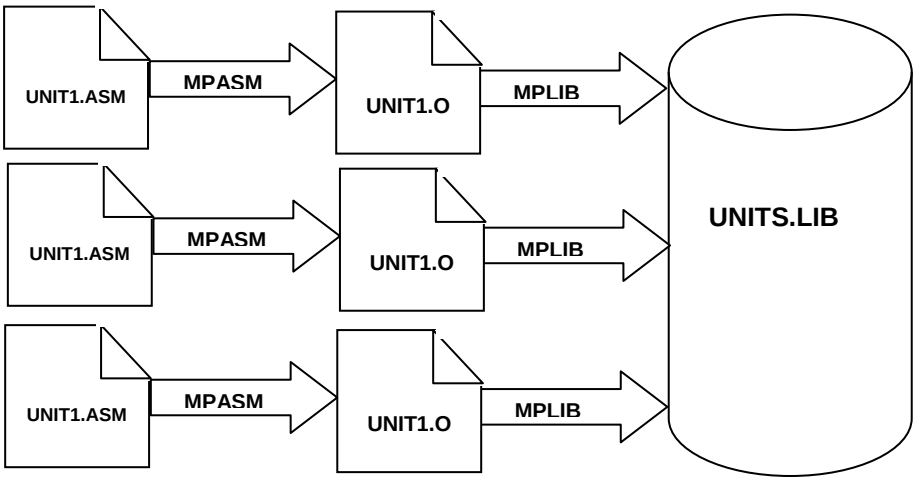


Рис.5.11. Компіляція проєкту з декількох файлів

Об’єктні файли можуть бути згруповані у бібліотечні файли за допомогою програми MPLIB. Бібліотеки можуть бути зазначені як параметр під час лінкування. Таким чином, до абсолютного коду будуть включені тільки необхідні процедури (рис.5.12).



.

Рис.5.12. Групування об’єктних файлів до бібліотеки

5.3.2. Вхідні і вихідні файли MPASM

Типи файлів, оj пов’язані із асемблером MPASM, наведені е табл. 5.3.

Таблиця 5.3

Типи файлів

Тип файлу	Опис
.ASM	Початковий файл MPASM, <Source_name>.ASM
.LST	Файл листинга програми, <Source_name>.LST
.ERR	Список помилок, які виникли при компіляції, <Source_name>.ERR
.HEX	Файл кода програми, <Source_name>.HEX
.HXL/HXH	Файли кода програми, окремо молодші і старші байти кода, <Source_name>.HXL, <Source_name>.HXH
.COD	Файл для відладчика, <Source_name>.COD
.O	Об’єктний файл програми, <Source_name>.O

Початковий файл (.ASM) програми може бути створений в будь-якому текстовому редакторі ASCII. Текст програми має відповідати наступним вимогам. Кожен рядок початкового тексту може мати до чотирьох інформаційних полів:

- мітка;
- мнемоніка команди;
- операнди команди;
- коментар.

Необхідно дотримуватись порядку розташування інформаційних полів в рядку. Мітки мають починатися з першої колонки. Мнемоніки команд мають починатися з другої (і далі) колонки. Операнди йдуть за мнемонікою команди. Коментар може розташовуватися за операндами, мнемоніками та мітками і може починатися в деякий колонці. Максимальна ширина колонці 255 символів. Мітки від мнемонік мають бути відділені двокрапкою, пробілом або символами табуляції, операнди мають розділятися комами. Приклад первинного файлу програми наведений на рис.5.13.

```
        list   p=16c54
Dest    equ    H'0B'

        org    H'01FF'
        goto   Start

        org    H'0000'

Start   movlw  H'0A'
        movwf  Dest
        bcf    Dest, 3
        goto   Start

        end
```

Рис.5.13. Приклад початкового файлу програми

Мітки. Мітка повинна починатись в колонці 1. За нею може бути двокрапка, пробіли, символи табуляції або кінец рядка.

Мітка має починатись з символу латинської абетки або символу підкреслення (_) та може складатись з літерно-цифрових символів латинської абетки, символу підкреслення або знаку питання (?).

Максимальна довжина мітки становить 32 символи.

За замовчуванням мітки чутливі до регістру символів. Цей параметр може бути змінений в командному рядку під час запуску MPASM. Якщо в

імені міткі застосовується двокрапка, відокремлена частина трактується як оператор, а не як частина імені мітки.

Мнемоніки. Мнемоніка інструкцій мікроконтролера, директиви асемблера та макрокоманди повинні починатися у другій (та далі) колонці. Якщо у тому ж рядку є мітка, вона має бути відділена двокрапкою або одним (або більше) символом пробілу (табуляції).

Операнди. Операнди мають бути відділені від мнемоніки одним (або більше) символом пробілу (табуляції). Багатократні операнди розділяються комами.

Коментар. Будь-який текст після крапки з комою (;) трактується як коментар та всі символи до кінця рядка ігноруються. Дозволяються рядкові константи, які містять (;), як коментар вони не сприймаються.

Формат файлу-лістнигу. Формат файлу лістингу наступний: Ім'я файлу та версія, дата та час компіляції, номер сторінки виводяться спочатку кожної сторінки.

Перша колонка цифр вказує базову адресу кода в пам'яті. Друга колонка показує 32-розрядне значення усіх символьних змінних, що створені директивами SET, EQU, VARIABLE, CONSTANT або CBLOCK. Третя колонка призначена для машиного коду, що виконується мікроконтролером. Четверта колонка містить номер рядка відповідного первинного файлу.

Залишок рядка зарезервований для первинного тексту, що спричинил машинний код. Приклад файлу лістингу наведений на рис.5.14.

Директиви асемлера. Директиви асемблера – це команди, які входять в склад початкового текста програми, но безпосередньо не включаються в початковий файл. Вони застосовуються для управління MPASM, параметрами введення/виведення та розподіленням даних.

Багато директив асемблера мають додаткові імена та формати. Вони призначені для забезпечення сумісності з більш ранніми версіями асемблера та індивідуальними методами програмування. Для отримання мінімального об'єма коду програми рекомендується застосовувати директиви MPASM.

Типи директив MPASM. Існує п'ять основних типів директив:

- директиви контролю – управляють створенням розділів умовно компільованого коду;

- директиви даних – управляють розподиленням пам’яті та призначенням символичних імен змінним та константам;

- директиви лістинга – визначають формат та склад файлу лістингу. Ці директиви дозволяють вказивати заголовки, нумерувати сторінки та налаштовувати інши параметри;

- макродирективи – управляють роботою макросів та розподіленням даних в тілі макроса;

- директиви об’єктного файлу – застосовуються тількі при створенні об’єктного файлу.

```
MPASM 01.99.21 Intermediate        MANUAL.ASM          5-30-1997 15:31:05 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT

VALUE
                  00001 ;
                  00002 ; Пример исходного файла MPASM.
                  00003 ;
                  00004         list p=16c54
0000000B          00005 Dest    equ    H'0B'
                  00006
01FF              00007         org    H'01FF'
01FF    0A00      00008         goto   Start
                  00009
0000              00010         org    H'0000'
                  00011
0000    0C0A      00012 Start   movlw  H'0A'
0001    002B      00013         movwf  Dest
0002    0A00      00014         goto   Start
                  00015
                  00016         end


MPASM 01.99.21 Intermediate        MANUAL.ASM          5-30-1997 15:31:05 PAGE 2

SYMBOL TABLE
  LABEL                VALUE

Dest                   0000000B
Start                  00000000
__16C54                00000001

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : XXX------------ ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used: 4
Program Memory Words Free: 508

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 0 reported, 0 suppressed
```

Рис. 5.14. Приклад файлу-лістнигу

Опис деяких директив.

CODE - початок секції об'єктного коду

Синтаксис:

[<label>] code [ROM address>]

Використовується при генерації об'єктних модулів. Оголошує початок секції програмного коду. Якщо <label> не вказана, секція буде названа code. Стартова адреса встановлюється рівною вказаному значенню або нулю, якщо адреса не була зазначена.

Приклад:

```
RESET code H'01FF'
goto START
```

#DEFINE - визначити мітку заміни тексту

Синтаксис:

#define <name> [<string>]

Директива задає рядок <string>, що заміщає мітку <name> всякий раз, коли та зустрічатиметься в початковому тексті.

Символи, які визначені директивою #DEFINE, не можуть бути проглянуті симулятором. Використовуйте замість цієї директиви EQU.

Приклад

```
#define length 20
#define control 0x19,7
#define position (X,Y,Z) (y-(2 * Z +X)).
test_label dw position(1, length, 512)
bsf control     ; встановити в 1 біт 7 в f19
```

END - кінець програмного блоку

Синтаксис:

End

Визначає кінець програми. Після зупинки програми таблиця символів скидається у файл лістингу.

Приклад:

```
start       ;виконуваний код;
end         ; кінець програми
```

EQU - визначити асемблерну константу

Синтаксис:

<label> equ <expr>

Тут <expr> - цей правильний MPASM вираз. Значення виразу привласнюється мітці <label>.

Приклад:

```
four equ 4   ; привласнює чисельне значення мітці four
```

INCLUDE - включити додатковий файл джерела

Синтаксис:

```
include <<include_file>>
include "<include_file>"
```

Визначуваний файл прочитується як джерело коду. Після закінчення файлу, що включається, продовжуватиметься асемблювання початкового коду програми. Допускається до шести рівнів вкладеності. Файл <include_file> може бути укладений в лапки або кутові дужки. Якщо вказаний повний шлях до файлу, то пошук відбуватиметься тільки по цьому шляху. Інакше порядок пошуку наступний: поточний робочий каталог, каталог, в якому знаходиться вихідний код програми, каталог MPASM.

Приклад:

```
include "з:\sys\sysdefs.inc" ; system defs
include <addmain.asm> ; register defs
```

LIST - встановити параметри лістингу

Синтаксис:

```
list [<list_option>, <list_option>]
```

Директива <list> дозволяє вивід лістингу, якщо він до цього був заборонений. Крім того, один з параметрів лістингу може бути змінений для управління процесом асемблювання відповідно до табл. 5.4.

Таблиця 5.4

Параметри, що використовуються директивою list

Параметр	Значення за замовчуванням	Опис
1	2	3
C=nnn	80	Кількість символів в рядку
n=nnn	59	Кількість рядків на сторінці
t=ON|OFF	OFF	Укорочувати рядки лістингу
p=<type>	None	Встановити тип процесора: PIC16C54, PIC16C84, PIC16F84, PIC17C42 і ін.
r=<radix>	HEX	Встановити систему числення за умовчанням: hex, dec, oct.

Продовження табл. 5.4

1	2	3
w=<level>	0	Встановити рівень повідомлень діагностики у файлі лістингу: 0 - виводити всі повідомлення; 1 - виводити попередження і помилки; 2 - виводити тільки помилки.
x=ON|OFF	OFF	Ввімкнути або вимкнути макророзширення.

NOLIST - вимкнути вихід лістингу

Синтаксис:

NOLIST

ORG - встановити початкову адресу програми

Синтаксис:

<label> org <expr>

Встановлює початкову адресу програми для подальшого коду відповідно до адреси в <expr>. MPASM виводить переміщуваний об'єктний код, а MPLINK розмістить код за певною адресою. Якщо мітка <label> визначена, то їй буде привласнена величина <expr>. За замовченням початкова адреса має нульове значення. Директива може не використовуватися, якщо створюється об'єктний модуль.

Приклад:

```
int_1 org 0x20; Перехід по вектору 20
int_2 org int_1+0x10; Перехід по вектору 30
```

5.4. Приклади програм

5.4.1. Правила складання програм

Для більш зручного читання та налаштування тексту програм при написанні програм доцільно застосовувати шаблон для оформлення тексту програми, який наведений на рис.5.15.

Мітка	Мнемокод	Операнд	Коментар
tempa	EQU	H’020’	;ініціалізація змінних

Рис. 5.15. Шаблон оформлення тексту програми

Приклад складання тексту програми

Початковий текст програми на асемблері повинен починатися з директив асемблера, які здійснюють вибір типу контролера LIST та підключення бібліотеки програм #INCLUDE.

```
list      p=16f877A              ;директива асемблера для выбору типу контролера
#include <P16F877A.INC>         ;директива асемблера для підключення библиотеки
                                 програм
```

Перший рядок програми завжди повинен здійснювати вибір типу контролера, оскільки він є вказівкою асемблеру, які програмні модулі підключати для моделювання роботи контролера. Типів контролерів у списку багато, вони відрізняються апаратним та програмним забезпеченням і тому потрібно вибрати конкретний тип контролера, який буде використовуватися в системі, що розробляється, або в пристрої.

Другий рядок програми здійснює підключення бібліотеки програм для обраного контролера.

Далі потрібно виконати ініціалізацію змінних, які будуть застосовуватися в програмі. Для цього необхідно вказати назву змінної, яка повинна складатися з букв латиниці та цифр, потім директиву асемблера присвоєння EQU, потім адресу змінної в пам'яті даних в шістнадцятковій системи числення, де буде зберигатися значення змінної. Адреса змінної складається з позначення шістнадцяткової системи числення h, одинарних лапок, між якими вказується адреса комірки пам'яті даних у шістнадцятковій системі числення

```
tempa  EQU   H'0020'     ; ініціалізація змінних
tempb  EQU   H'0021'
```

Далі потрібно вказати вектор скидання (директива асемблера ORG), застосувати команду безумовного переходу за міткою на початок програми та вказати адресу розташування програми в пам'яті програм.

```
        org     0x000     ; вектор скидання
        goto    ini       ; перехід за міткою
        org     0x20      ; вказівка адреси розміщення програми в пам’яти програм
ini     clrf    tempa     ; очищення змінної tempa
        …                 ; текст програми
        end               ; директива асемблера закінчення текста програми
```

Потім пишется текст програми та в кінці директива асемблера END закінчення програми.

Приклади привласнення числового значення константи змінній при програмуванні.

В системі команд контролера є три команди пересилань:

1) команда **movwf f** – переслати зміст акумулятора W в регістр f;
2) команда **movfw f,d** – переслати зміст регістра f в акумулятор W (Якщо d=0, значення регістра f пересилается в W. Якщо d=1, значення регістра f пересилается в f, при цьому біт Z регістра STATUS приймає значення 1, якщо f=0. У такому випадку ця команда застосовується для перевірки змісту регістра f на нуль);
3) команда **movlw b'00001101'** – переслати константу в акумулятор W.

Для привласнення змінній числового значення необхідно застосувати дві команди пересилань:

```
movlw  d'10'        ; переслати десяткове число 10 в акумулятор.
movwf  tempa        ; переслати десяткове число 10 з акумулятора в регістр  tempa, в
                      акумуляторі залишається десяткове число 10.
```

Приклад переходу з одного банку до іншого банку.

При написанні програм часто потрібно переходити з одного банку до іншого банку. Це викликано тим, що регістри спеціального призначення та загального призначення перебувають у різних банках. Щоб отримати доступ до регістру, необхідно перейти до банку або відкрити банк, в якому знаходиться цей регістр. Регістри, що найчастіше використовуються, наприклад, STATUS знаходяться у всіх банках.

Перехід з банку до банку здійснюється за допомогою регістру спеціального призначення STATUS. Для цього треба використовувати біти RP0, RP1 регістру STATUS при безпосередньої адресації банків, та біт IRP – при непрямої адресації. Вибір банку здійснюється записом певної двійкової комбінації в біти RP0, RP1, яка наведена у табл. 5.5.

Таблиця 5.5

Стани бітів RP1,RP0 регістра STATUS для вибору банків пам'яті

RP1	RP0	Банк
0	0	Банк 0
0	1	Банк 1
1	0	Банк 2
1	1	Банк 3

Встановлення бітів RP0, RP1 в регістрі STATUS здійснюється за допомогою команд в тексті програми. Для цього застосовують бітові команди bcf f,b і bsf f,b. Команда bcf f,b очищує біт b в регістрі f (b=0), команда bsf f,b встановлює біт b в регістрі f (b=1).

Для вибору банку 0 необходно написати наступні команди:

```
bcf   STATUS,RP0      ;перехід до банку 0
bcf   STATUS,RP1
```

Вибір банку 1 здійснюється наступними командами:

```
bsf   STATUS,RP0      ;перехід до банку 1
bcf   STATUS,RP1
```

При цьому необхідно мати на увазі, що після скидання контролера при запуску програми по замовчування встанавлюється банк 0. При написанні команд в тексті програми регістри спеціального призначення та біти регістрів пишуться великими литерами. Замість назви біта може бути вказаний порядковий номер біта в регістрі.

Застосування ознак виконаних арифметичних або логічних операцій.

Три перших біта регістра STATUS – C, DC, Z – це біти ознак або прапорів результату виконаної операції, які в залежності від результату виконаної операції приймають значення «0» або «1». Біт C – ознака переносу/позики, біт DC – ознака додаткового переносу/позики, біт Z – ознака нульового результату.

Перевірка стану біта виконується при необхідності перевірки логічної умови, вона виконується за допомогою бітових команд btfsc f,b та btfss f,b.

Ці команди перевіряють стан біта b в регістрі f та пропускають наступну команду, якщо виконується логічна умова. Команда btfsc f,b перевіряє стан біта b, і якщо він дорівнює 0, то пропускається виконання наступної команди. Команда btfss f,b перевіряє стан біта b, і якщо він дорівнює 1, то пропускається виконання наступної команди. Приклад застосування цієї команди в програмі:

```
Мітка    Мнемокод і операнди                Коментар
         btfss  STATUS,C      ;перевіріти біт C переносу / позики, якщо він дорівнює 1,
                              тобто ;відбувся перенос з 7-го розряду до 8-го розряду, то
                              наступна ;команда goto m1 пропускається та виконується
                              команда incf ;tempa, якщо біт C дорівнює 0, то виконується
                              команда goto m1 і ;відбувається перехід до мітки m1.
         goto   m1;
         incf   tempa;
m1       nop;
```

Створення циклів.

Ще однією програмною конструкцією є цикли, які часто використовуються в програмах для реалізації багаторазового виконання будь-якої послідовності команд. Цикл має тіло циклу, лічильник циклу та команду виходу з циклу. Цикл може бути реалізований за допомогою команд умовних переходів btfsc f,b та btfss f,b, decfsz f,d, incfsz f,d.

Приклад циклу:

```
Мітка   Мнемокод і операнди        Коментар
m1                                  ;мітка, до якої здійснюється перехід
        addwf   tempb,0             ;додати W и tempb
        decf    tempi               ;відняти 1 з tempi
        btfss   STATUS,Z            ;перевірити бит Z в регістрі STATUS. Якщо результат
                                    віднімання в попередній команді дорівнює 0, то біт
                                    Z=1,  наступна команда goto m1 пропускається та
                                    виконується команда movwf tempc. Якщо результат
                                    віднімання не дорівнює 0, то біт Z=0,  виконується
                                    команда goto m1, за якою відбувається перехід до мітки
                                    m1 на початок циклу.
        goto    m1
        movwf   tempc
```

5.4.2. Програми арифметичних операцій

5.4.2.1. Програми додавання

Програма додавання однобайтних чисел

```
Мітка       Мнемокод і операнди                 Коментар
            list        p=16f877a               ;вибір типу мікроконтролера
            #include    <p16f877a.inc>          ;підключення бібліотеки програм
tempa       EQU         h'020'                  ;ініціалізація змінних
tempb       EQU         h'021'
tempc       EQU         h'022'
            org         0x00                    ;вектор скідання
            goto        begin                   ;перехід до основної програми
            org         0x100                   ;адреса розміщення програми
begin       clrw                                ;очищення акумулятора
            clrf        tempa                   ;очищення змінних
            clrf        tempb
            clrf        tempc
            movlw       d'20'                   ;запис числа 20 в десятковій системі числення до
                                                ;акумулятора
            movwf       tempa                   ;пересилка 20 із акумулятора в регістр tempa
            movlw       d'10'
            movwf       tempb
            addwf       tempa,0                 ;додавання змінної tempa до акумулятора, сбереження
                                                ;результату додавання відбувається в акумуляторі
            movwf       tempc                   ;пересилання результату додавання з акумулятора в tempc
            nop                                 ;немає операції
            end                                 ;кінець програми
```

Програма додавання однобайтних чисел з отриманням двобайтного результату.

Якщо результат додавання двох однобайтних чисел буде перевищувати 255_{10}, результат додавання буде двобайтним.

У випадку отримання двобайтного результату додавання треба резервувати два регістри для результату додавання – *tempcl*, *tempch* відповідно молодший та старший байти. При цьому у регістрі *tempcl* буде розташовуватись результат виконання команди *addwf tempa,0*, який буде пересилаться до регістру *tempcl* з акумулятора. А в регістр *tempch* треба записати «1», якщо в результаті додавання біт переносу C регістра STATUS отримає значення «1», тобто відбувся переніс з 7-го розряду до 8-го розряду і результат додавання буде двобайтним. Запис виконується командою *incf tempch*, тобто додаємо «1» до *tempch* тоді, коли біт C набуває значення «1». Це виконується командою умовного переходу *btfsc STATUS,C*, яка перевіряє стан біта C та, якщо він дорівнює «1», виконується наступна команда *incf tempch*. Якщо біт C набуває значення «0», тобто результат додавання однобайтний, наступна команда пропускається та виконується команда *nop*.

```
Мітка     Мнемокод і операнди          Коментар
          list      p=16f877a          ;вибір типу мікроконтролера
          #include  <p16f877a.inc>     ;підключення бібліотеки програм
tempa     EQU       h'020'             ;ініціалізація змінних
tempb     EQU       h'021'
tempcl    EQU       h'022'
tempch    EQU       h'023'
          org       0x00               ;вектор скідання
          goto      begin              ;перехід до основної програми
          org       0x100              ;адреса розміщення програми
begin     clrw                         ;очищення акумулятора
          clrf      tempa              ;очищення змінних
          clrf      tempb
          clrf      tempcl
          clrf      tempch
          movlw     d'200'             ;запис числа 200 в десятковій системі числення до
                                       ;акумулятора
          movwf     tempa              ;пересилка 200 із акумулятора в регістр tempa
          movlw     d'100'
          movwf     tempb
          addwf     tempa,0            ;додавання змінної tempa до акумулятора, збереження
                                       ;результату додавання в  акумуляторі
          movwf     tempcl             ;пересилання результату додавання з акумулятора до
                                       ;tempcl, отримання молодшого байту результату
                                       ;додавання
          btfsc     STATUS,C           ;перевірка  біта  C, який є ознакою перенесення
          incf      tempch             ;додавання 1 до tempch, отримання старшого байти
                                       ;результату додавання
          nop
          end
```

5.4.2.2. Програма віднімання чисел

```
Мітка    Мнемокод і операнди          Коментар
         list     p=16f877a           ;вибір типу мікроконтролера
         #include <p16f877a.inc>      ;підключення бібліотеки програм
tempa    EQU      h'020'              ;ініціалізація змінних
tempb    EQU      h'021'
tempc    EQU      h'022'
         org      0x00                ;вектор скідання
         goto     begin               ;перехід до основної програми
         org      0x100               ;адреса розміщення програми
begin    clrw                         ;очищення акумулятора
         clrf     tempa               ;очищення змінних
         clrf     tempb
         clrf     tempc
         movlw    d'15'               ;запис числа 15 в десятковій системі числення до
                                      ;акумулятора
         movwf    tempa               ;пересилка 15 із акумулятора в регістр tempa
         movlw    d'2'
         movwf    tempb
         subwf    tempa,0             ;віднімання акумулятора із змінної tempa, сбереження
                                      ;результату віднімання в акумуляторі
         movwf    tempc               ;пересилка резудьтату віднімання в tempc
         nop                          ;немає операції
         end                          ;кінець програми
```

5.4.2.3. Програма множення однобайтних чисел

Команда множення відсутня в системі команд мікроконтролера, тому множення чисел виконується за допомогою програми множення. Процедура множення полягає в заміні операції множення операцією багатократного додавання одного з чисел до самого себе, а інше число після кожного додавання зменшується на одиницю.

Багатократне додавання реалізується за допомогою циклу, в якому одне число додається до самого себе, а з іншого числа віднімається одиниця і це число називається лічильником циклу. Після кожного віднімання виконується перевірка результату віднімання і як тільки він буде дорівнювати 0, виконується вихід з циклу, отримана в результаті додавання сума буде результатом множення.

```
Мітка    Мнемокод і операнди          Коментар
         list     p=16f877a           ;вибір типу мікроконтролера
         #include <p16f877a.inc>      ;підключення бібліотеки програм
tempa    EQU      h'020'              ;ініціалізація змінних
tempb    EQU      h'021'
tempc    EQU      h'022'
         org      0x00                ;вектор скідання
         goto     begin               ;перехід до основної програми
         org      0x100               ;адреса розміщення програми
begin    clrw                         ;очищення акумулятора
```

```
        clrf    tempa           ;очищення змінних
        clrf    tempb
        clrf    tempc
        movlw   d'7'            ;запис числа 7 в десятковій системі числення до
                                ;акумулятора
        movwf   tempa           ;пересилка 7 із акумулятора в регістр tempa
        movlw   d'8'            ;запис числа 8 в десятковій системі числення до
                                ;акумулятора
        movwf   tempb           ;пересилка 8 із акумулятора в регістр tempb
        movfw   tempa           ;пересилка tempa в акумулятор
m1      addwf   tempc,1         ;додавання акумулятора до змінної tempc, сбереження
                                ;результату додавання  в  акумуляторі
        decf    tempb           ;віднімання 1 з tempb
        btfss   STATUS,Z        ;перевірити бит Z в регістрі STATUS.
                                ;Якщо результат віднімання в попередній команді
                                ;дорівнює 0, то біт Z=1,  наступна команда goto m1
                                ;пропускається та виконується команда movwf tempc.
                                ;Якщо результат віднімання не дорівнює 0, то біт Z=0,
                                ;виконується команда goto m1, за якою відбувається
                                ;перехід до мітки m1 на початок циклу.
        goto    m1              ;перехід до мітки m1
        nop                     ;немає операції
        end                     ;кінець програми
```

5.4.2.4. Програма ділення однобайтних чисел

Команда ділення відсутня в системі команд мікроконтролера, тому ділення чисел виконується за допомогою програми ділення. Процедура ділення полягає в багатократному відніманні дільника з діленого, часткою ділення буде кількість віднімань.

Багатократне віднімання реалізується за допомогою циклу, в якому дільник віднімається з діленого і рахується кількість віднімань. Після кожного віднімання виконується перевірка результату віднімання і як тільки він буде дорівнювати 0, виконується вихід з циклу, отримана в результаті рахування кількість віднімань буде часткою ділення.

Якщо результат останнього віднімання буде дорівнювати не 0, а буде від’ємним, тобто ділення не ціле, а з остачею, то треба ще перевіряти результат віднімання на позику.

```
Мітка   Мнемокод і операнди    Коментар
        list    p=16f877a       ;вибір типу мікроконтролера
        #include <p16f877a.inc> ;підключення бібліотеки програм
tempa   EQU     h'020'          ;ініціалізація змінних
tempb   EQU     h'021'
tempc   EQU     h'022'
        org     0x00            ;вектор скідання
        goto    begin           ;перехід до основної програми на мітку begin
        org     0x100           ;адреса розміщення програми
begin   clrw                    ;очищення акумулятора
```

```
        clrf    tempa           ;очищення змінних
        clrf    tempb
        clrf    tempc
        movlw   d'16'           ;запис числа 16 в десятковій системі числення до
                                ;акумулятора
        movwf   tempa           ;пересилка 16 із акумулятора в регістр tempa
        movlw   d'3'            ;запис числа 3 в десятковій системі числення до
                                акумулятора
        movwf   tempb           ;пересилка 3 із акумулятора в регістр tempb
m1      incf    tempc           ;додавання 1 до tempc – результат ділення
        subwf   tempa,1         ;віднімання акумулятора зі змінної tempa, збереження
                                результату віднімання  в  tempa
        btfss   STATUS,Z        ;перевірити бит Z в регістрі STATUS.
                                ;Якщо результат віднімання в попередній команді
                                ;дорівнює 0, то біт Z=1,  наступна команда goto m3
                                ;пропускається і виконується команда goto m4. Якщо
                                ;результат віднімання не дорівнює 0, то біт Z=0,
                                ;виконується наступна команда goto m3.
        goto    m3
        goto    m4
m3      btfss   STATUS,C        ;перевірити бит C в регістрі STATUS.
                                ;Якщо результат віднімання в попередній команді
                                ;додатний, то біт C=1 і  виконується команда goto m1
                                ;повернення на початок циклу. Якщо результат віднімання
                                ;від'ємний, то виконується  наступна команда goto m2,
                                ;перехід до команди decf tempc
        goto    m2              ;перехід до мітки m2
        goto    m1              ;перехід на початок циклу
m2      decf    tempc           ;відняти 1 з tempc
m4      nop                     ;немає операції
        end                     ;кінець програми
```

5.4.3. Програми перетворення кодів

5.4.3.1. Програма перетворення двійково-десяткового числа $10000101_{2\text{-}10}$ (85_{10}) на двійкове число

Двійково-десяткове число складається з двох тетрад: старша тетрада відповідає десятковому числу 8; молодша тетрада – десятковому числу 5. Старша тетрада відповідає кількості десятків в числі, молодша терада – кількості одиниць.

Десяткове число можна представити як суму десяткового розяда і одиничного розряду

$$85 = 80 + 5 = 8\times10 + 5\times1.$$

Тому для того, щоб перетворити двійково-десяткове число в двійкове необхідне старшу тетраду, яка відповідає кількості десятків в числі, помножити на 10 і потім додати до отриманого результату модлодшу тетраду, яка відповідає кількості одиниць.

```
Мітка   Мнемокод і операнди        Коментар
        list    p=16f877a           ;вибір типу мікроконтролера
        #include <p16f877a.inc>     ;підключення бібліотеки програм
```

```
tempa   EQU     h'020'          ;ініціалізація змінних
tempah  EQU     h'021'
tempal  EQU     h'022'
tempb   EQU     h'023'
        org     0x00            ;вектор скідання
        goto    begin           ;перехід до основної програми
        org     0x20            ;адреса розміщення програми
begin   clrw                    ;очищення акумулятора
        clrf    tempa           ;очищення змінних
        clrf    tempah
        clrf    tempal
        clrf    tempb
        movlw   b'10000101'     ; запис десяткового числа 85 в двійково-десятковому коді
                                ;10000101 2-10  до акумулятора
        movwf   tempah          ;пересилка  із акумулятора в регістр tempah
        movwf   tempal          ;пересилка  із акумулятора в регістр tempal
        bcf     tempah,0        ;очищення бітів 0-3 tempah
        bcf     tempah,1
        bcf     tempah,2
        bcf     tempah,3        ;після очищення бітів 0-3 tempah отримаємо число 80 2-10
        bcf     tempal,4        ;очищення бітів 4-7 tempal
        bcf     tempal,5
        bcf     tempal,6
        bcf     tempal,7        ;після очищення бітів 4-7 tempal отримаємо число 05 2-10 –
                                ;кількість одиниць
        swapf   tempah,1        ;обмін півбайтами, отримаємо число 08 – кількість
                                ;десятків
        movlw   d'10'
        movwf   tempb           ;запис 10 в tempb
        movfw   tempah          ;запис tempah= 08 в акумулятор
        clrf    tempah          ;очищення tempah
                                ;множення tempah на 10
m1      addwf   tempah,1        ;додати tempah до акумулятора, зберегти в tempah
        decf    tempb           ;зменшити tempb на одиницу
        btfss   STATUS,Z        ;перевірити результат віднімання, якщо не дорівнює 0, то
                                ;перейти на початок циклу, якщо 0, то перейти на команду;
                                movfw tempah
        goto    m1
        movfw   tempah          ;записати tempah в акумулятор
        addwf   tempal,0        ;додати до акумулятору tempal
        movwf   tempa           ;записати результат додавання в tempa – це буде двійкове
                                число 85
        nop                     ;немає операції
        end                     ;кінець програми
```

5.4.3.2. Програма перетворення двійкового числа 00110011_2 (51_{10}) на двійково-десяткове число

Для перетворення двійкового числа в двійково-десяткове необхідно визначити скількі десятків містить двійкове число, для цього потрібно поділити двійкове число на 10 і отримати частку, яка буде кількісттю десятків і буде відповідати старшої тетраді двійково-десяткового числа. Остача буде відповідати кількості одиниць, тобто молодший тетраді двійково-десяткового числа.

```
Мітка   Мнемокод і операнди          Коментар
        list    p=16f877a            ;вибір типу мікроконтролера
        #include <p16f877a.inc>      ;підключення бібліотеки програм
lsd     EQU     h'020'               ;ініціалізація змінних, які призначні для зберігання
msd     EQU     h'021'               ;двійково-десяткового числа
        org     0x00                 ;вектор скідання
        goto    begin                ;перехід до основної програми на мітку begin
        org     0x100                ;адреса розміщення програми
begin   clrw                         ;очищення акумулятора
        clrf    lsd                  ;очищення змінних
        clrf    msd
        movlw   d'51'                ;запис числа 51 в десятковій системі числення до
                                     ;акумулятора
        movwf   lsd                  ;пересилка 51 із акумулятора в регістр lsd
gtenth  movlw   d'10'                ;запис числа 10 в десятковій системі числення до
                                     ;акумулятора
        subwf   lsd,0                ;віднімання  числа 10 зі змінної lsd
        btfss   STATUS,C             ;перевірити бит C в регістрі STATUS.
                                     ;Якщо результат віднімання в попередній команді
                                     ;від'ємний, то біт C=0,  виконується наступна  команда
                                     ;goto over. Якщо результат віднімання додатний, то біт
                                     ;C=1, то пропускається  наступна команда і виконується
                                     ;команда movwf lsd .
        goto    over
        movwf   lsd                  ;різниця після віднімання, кількість одиниць
        incf    msd                  ;збільшення старшої тетради на одиницю, кількість
                                     ;десятків
        goto    gtenth
over    nop
        swapf   msd                  ;переставлення  місцями тетрад
        movfw   msd                  ;запис змінної msd в акумулятор
        addwf   lsd,1                ;додавання до старшої тетради  молодшої, кількість
                                     ;десятків додається до кількості одиниць  і в lsd буде
                                     ;записано двійково-десяткове число
        nop                          ;немає операції
        end                          ;кінець програми
```

6. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ. ТАЙМЕРИ ТА ПОРТИ

6.1. Таймери

6.1.1. Модуль таймера TMR0

Модуль TMR0 таймера/лічильника має такі особливості:

- 8-розрядний таймер/лічильник;
- можливість читання та запису поточного значення лічильника;
- 8-розрядний програмований дільник;
- внутрішнє або зовнішнє джерело тактового сигналу;
- вибір активного фронту зовнішнього тактового сигналу;
- переривання при переповненні (перехід від FFh до 00h).

Блок-схема модуля TMR0 та загального з WDT попередника наведена на рис. 6.1.

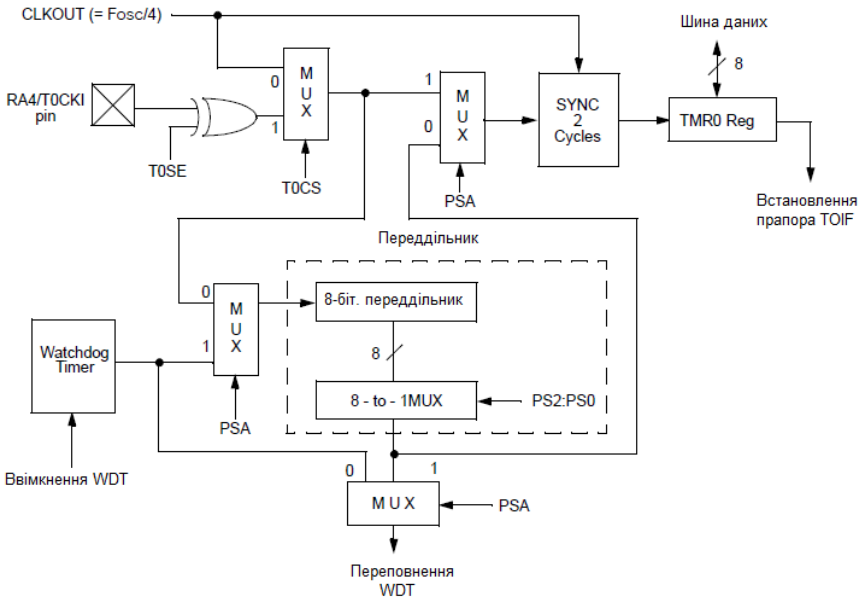


Рис. 6.1. Блок-схема таймера TMRO [4]:
біти управління TOCs, TOSE, PS2,PS1,PS0, PSA розташовані в регістрі OPTION_REG

Якщо біт TOCS скинутий на «0» (OPTION_REG<5>), TMR0 працює від тактового внутрішнього сигналу. Приріст лічильника TMR0 відбувається в кожному машинному циклі (якщо вимкнено переддільник). Після запису до TMR0 збільшення лічильника заборонено впродовж двох наступних циклів.

Користувач повинен скоригувати цю затримку перед записом нового значення TMR0.

Якщо біт TOCS встановлено в «1» (OPTION_REG<5>), TMR0 працює від зовнішнього джерела тактового сигналу з входу RA4/TOCKI. Активний фронт зовнішнього тактового сигналу вибирається бітом TOSE у регістрі OPTION_REG<4> (TOSE=0 – активним є фронт сигналу).

Переддільник може бути увімкнений перед WDT або TMR0, залежно від стану біта PSA (OPTION_REG<3>). Не можна прочитати або записати нове значення у переддільник.

Використання зовнішнього джерела тактового сигналу для TMR0. Якщо дільник не використовується, зовнішній тактовий сигнал надходить безпосередньо до синхронізатора. Синхронізація TOCKI з тактовим сигналом мікроконтролера ускладнюється через опитування виходу синхронізатора у машинні цикли *Q2* та *Q4*. Тому тривалість високого або низького логічного рівня зовнішнього сигналу повинна бути не меншою за $2T_{osc}$ (плюс невелика затримка внутрішнього RC- кола на 20 нс).

Переддільник. 8-розрядний лічильник може працювати як переддільник TMR0 або вихідний дільник WDT. Для простоти опису цей лічильник називається переддільник. Існує лише один переддільник, який може бути включений перед TMR0 або WDT. Використання переддільника перед TMR0 означає, що WDT працює без переддільника, і навпаки.

Коефіцієнт поділу переддільника визначається бітами PSA і PS2:PS0 у регістрі OPTION_REG<3:0> (див. табл. 5.2).

Якщо переддільник включений перед TMR0, будь-які команди запису TMR0 (наприклад, CLRF 1, MOVWF 1, BSF 1,x та т. п.) скидають переддільник.

Якщо переддільник підключено до WDT, команда CLR WDT скине переддільник разом з WDT.

Переддільник також очищається при скиданні мікроконтролера.

Переддільник недоступний для читання/запису.

Запис до регістру TMR0 скине переддільник, якщо він підключений до TMR0, але не змінить його режиму роботи.

Приклад програми переривань таймером TMR0 наведений нижче. Для того, щоб застосувати переривання від таймера TMR0 необхідно в програмі після вектора скидання вказати вектор переривань org 0x04 і потім команду перехода до підпрограми.

На рис. 6.2 наведений приклад програми застосування таймера TMR0.

```
Мітка   Мнемокод і операнди          Коментар
        list    p=16f877a            ;вибір типу мікроконтролера
        #include <p16f877a.inc>      ;підключення бібліотеки програм
tempa   EQU     h'020'               ;ініціалізація змінних
tempb   EQU     h'021'
        org     0x00                 ;вектор скідання
        goto    begin                ;перехід до основної програми на мітку begin
        org     0x04                 ;вектор переривання
        goto    prer                 ;перехід до підпрограми переривань
        org     0x100                ;адреса розміщення програми
begin   clrw                         ;очищення акумулятора
        clrf    tempa                ;очищення змінних
        clrf    tempb
        bsf     STATUS,RP0           ;перехід до банку 1
        movlw   b'10100000'          ;налаштування таймера TMR0
        movwf   OPTI ON_REG
        movlw   b'10100000'          ;налаштування переривань
        movwf   INTCON
        bcf     STATUS,RP0           ;перехід до банку 0
        clrf    TMR0
        movlw   d'240'               ;завдання початкового значення лічби
        movwf   TMR0;
m1      incf    tempb                ;цикл для формування машиних циклів для роботи
                                     ;таймера
        btfss   tempb,7
        goto    m1
prer    movlw   d'1'                 ;підпрограма переривань
        movwf   tempa
        bcf     INTCON,2             ;скидання прапора переривання від TMR0
        movlw   d'240';              ;завдання початкового значення лічби
        movwf   TMR0;
        bsf     INTCON,7             ;дозвіл переривань
        return                       ;повернення з підпрограми переривань
        nop                          ;немає операції
        end                          ;кінець програми
```

Рис. 6.2. Приклад програми застосування таймера TMR0

6.1.2. Модуль таймера TMR1

TMR1 – 16-розрядний таймер/лічильник, що складається з двох 8-розрядних регістрів (TMR1H та TMR1L), доступних для читання та запису. Підрахунок виконується у спарених регістрах (TMR1H:TMR1L), інкрементуючи їх значення від 0000h до FFFFh, далі рахує з 0000h. При переповненні лічильника встановлюється у «1» прапор переривання TMR1IF в регістрі PIR1<0>. Власне переривання можна дозволити або заборонити установкою/скиданням біта TMR1IE в регістрі PIE1<0>.

Модуль TMR1 може працювати у двох режимах:

- режим таймера;

- режим лічильника.

Вмикання модуля TMR1 здійснюється установкою біта TMR1ON у «1» (T1CON<0>).

Бітом TMR1CS (T1CON<1>) вибирається джерело тактових імпульсів. У режимі таймера TMR1 інкрементується на кожному машинному циклі. Якщо TMR1 працює із зовнішнім джерелом тактового сигналу, то збільшення відбувається за кожним фронтом сигналу.

Модуль TMR1 має внутрішній вхід скидання від модуля CPP.

Якщо увімкнено генератор тактових імпульсів (T1OSCEN=1), виводи RC1/T1OSI/CCP2 та RC0/T1OSO/T1CKI налаштовані як входи. Значення бітів TRISC<1:0> ігноруються, а зчитування даних з цих виводів дає результат «0».

Керуючі біти модуля TMR1 знаходяться у регістрі T1CON за адресою 10h (табл. 6.1).

Таблиця 6.1

Регістр T1CON

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	-T1SYNC	TMR1CS	TMR1ON
Біт 7-6	Не реалізовані: читаються як «0»						
Біт 5-4	T1CKPS1: T1CKPS0: Вибір коефіцієнта ділення передділення TMR1 11 - 1:8 10 - 1:4 01 - 1:2 00 - 1:1						
Біт 3	T1OSCEN: Увімкнення тактового генератора TMR1 1 - Генератор увімкнений 0 - Генератор вимкнений (інвертуючій елемент і резистивний зворотній зв’язок вимкнений для зменшення струму споживання)						
Біт 2	-T1SYNC: Синхронізація зовнішнього тактового сигналу TMR1CS = 1 1 - не синхронізувати зовнішній тактовий сигнал 0 - синхронізувати зовнішній тактовий сигнал TMR1CS = 0 Значення біта ігнорується						
Біт 1	TMR1CS: Вибір джерела тактового сигналу 1 - зовнішнє джерело з виводу RC0/T1OSO/T1CKI (активним є фронт сигналу) 0 - внутрішнє джерело $F_{OSC}/4$						
Біт 0	TMR1ON: Включення модуля TMR1 1 = увімкнений 0 = вимкнений						

Робота модуля TMR1 у режимі таймера. Приріст таймера відбувається від внутрішнього сигналу $F_{osc}/4$, коли біт TMR1CS(T1CON<1>) скинутий на «0». У цьому режимі біт синхронізації T1SYNC(T1CON<2>) ігнорується, тому що внутрішній тактовий сигнал завжди синхронізований.

Робота TMR1 у режимі лічильника. Модуль TMR1 може працювати у синхронному або асинхронному режимах залежно стану біта TMR1CS. Коли TMR1 використовує зовнішній тактовий сигнал, збільшення таймера відбувається за фронтом. Увімкнувши TMR1 в режим зовнішнього тактового сигналу, підрахування почнеться тільки після появи зрізу сигналу (рис. 6.3).

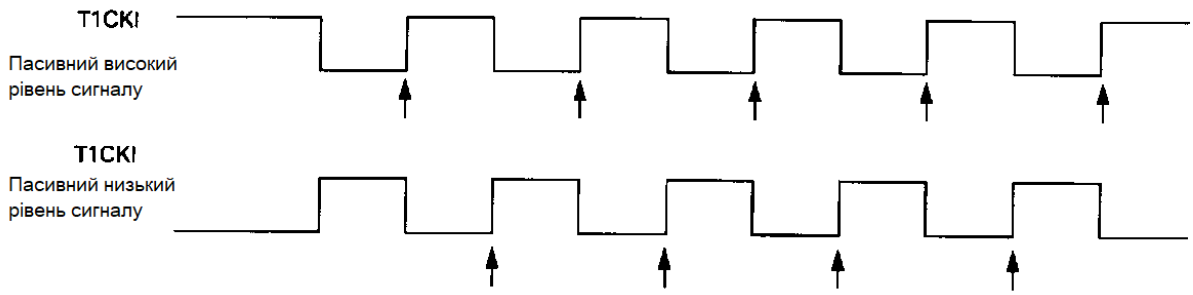


Рис. 6.3. Схема приросту TMR1 [4]

Робота TMR1 у режимі синхронного лічильника. Робота TMR1 від зовнішнього джерела тактового сигналу вибирається встановленням біта TMR1CS у «1» (рис. 6.4). У цьому режимі збільшення таймера відбувається за кожним фронтом сигналу на виводі RC1/T1OS/CCP2 (якщо T1OSCEN=1) або RC0/T1OSO/T1CKI (якщо T1OSCEN=0).

Якщо T1SYNC=0, активний фронт зовнішнього тактового сигналу синхронізується з внутрішнім тактовим сигналом на виході асинхронного переддільника.

Якщо T1OSCEN=0, інвертувальний елемент та резистивний зворотний зв’язок вимкнені для зменшення струму споживання.У режимі SLEEP мікроконтролера лічильник не буде інкрементуватися за наявності тактового сигналу через вимкнений синхронізатор (переддільник продовжує рахування від тактових імпульсів).

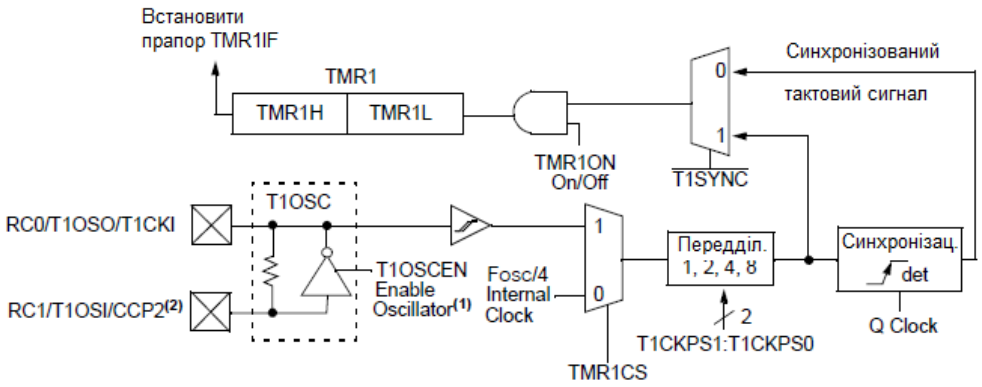


Рис. 6.4. Блок-схема таймера TMR1 [4]

Робота TMR1 у режимі асинхронного лічильника. Якщо біт – T1SYNC (T1CON<2>) встановлений у «1», зовнішній тактовий сигнал TMR1 не синхронізуватиметься з внутрішнім тактовим сигналом мікроконтролера, таймер продовжує працювати в режимі SLEEP мікроконтролера. Переповнення таймера викликає «пробудження» мікроконтролера, якщо дозволено переривання від TMR1. Однак потрібна обережність під час запису/читання TMR1.

У цьому режимі TMR1 не може використовуватися для захоплення/порівняння даних модуля CCP.

Читання/запис TMR1 в асинхронному режимі.

Читання TMR1H або TMR1L під час рахування в асинхронному режимі гарантує отримання поточного значення лічильника (реалізовано апаратно). Однак користувач повинен мати на увазі, що читання 16-розрядного значення виконується побайтно. Це накладає деякі обмеження, оскільки таймер може переповнитися між читаннями байт.

Запис TMR1 рекомендується виконувати після зупинки таймера. Запис у регістри TMR1 під час збільшення таймера може призвести до непередбачуваного значення регістра.

Робота TMR1 у режимі генератора. Резонатор підключається до виводів T1OSI (вхід) та T1OSO (вихід підсилювача). Максимальна частота резонатора – 200 КГц. Тактовий генератор TMR1 (ідентичний LP генератору) переважно призначений для кварцового резонатора 32 КГц. Увімкнення генератора здійснюється встановленням біта T1OSEN у регістрі T1CON<3>, що дозволяє TMR1 працювати в режимі SLEEP мікроконтролера (табл. 6.2).

Користувач повинен забезпечити програмне затримування, щоб гарантувати належний запуск генератора.

Таблиця 6.2

Вибір конденсаторів для генератора TMR1

Тип генератора	Частота	C1	C2
LP	32 кГц	33 пФ	33 пФ
	100 кГц	15 пФ	15 пФ
	200 кГц	15 пФ	15 пФ
Орієнтовані значення			
Протестовані резонатори:			
32.768 кГц	Epson C-001 R32.768K-A		±20 PPM
100 кГц	Epson C-2 100.00 KC-P		±20 PPM
200 кГц	STD XTL 200.000 кГц		±20 PPM

Примітка: Велика ємність збільшує стабільність генератора, але також збільшує час запуску.

Скидання модуля TMR1 тригером модуля CCP.

Якщо модуль CCP1 або CCP2 працює в режимі порівняння з тригером спеціальних функцій (CCP1M3:CCP1M0=1011), сигнал тригера скине модуль TMR1.

Сигнал з тригера спеціальних функцій модуля CCP1 не буде встановлювати прапор TMRIF (PIR<0>) у стан «1».

Модуль TMR1 має працювати у режимі синхронізованого зовнішнього тактового сигналу або внутрішнього тактового сигналу. Ця функція не працює в асинхронному режимі.

Якщо запис TMR1 збігається з сигналом скидання від тригера спеціальних подій, пріоритет віддається запису TMR1.

У цьому режимі модуля CCP період скидання TMR1 зберігається в регістрах CCPR×H:CCPR×L.

Скидання регістрів TMR1 (TMR1H, TMR1L). Регістри TMR1H, TMR1L не скидаються в 00h при скиданні за вмиканням живлення POR та інших видах скидання, за виключенням скидання за сигналом тригера спеціальних подій модуля CCP1 чи CCP2.

Регістр T1CON скидається в 00h при скиданні POR та BOR (TMR1 вимикається, коефіцієнт переддільника дорівнює 1:1). За інших видів скидання значення регістру T1CON не зміниться.

Переддільник TMR1. Переддільник TMR1 очищається при записі до регістрів TMR1H та TMR1L.

Регістри та біти, що пов’язані з роботою TMR1, наведені у табл. 6.3, а приклад програми застосування TMR1 – на рис. 6.5.

Таблиця 6.3

Регістри та біти, що пов'язані з роботою TMR1

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh 8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Молодший байт 16-розрядного таймера 1								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Старший байт 16-розрядного таймера 1								xxxx xxxx	uuuu uuuu
10h	T1CON	-	-	T1CKPS1	T1CKPS0	T1OSCEN	-T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Позначення: - – не застосовується, читається як 0; u – не змінюється; x – не відомо; q – залежить від умов

```
Мітка   Мнемокод і операнди        Коментар
        list    p=16f877a          ;вибір типу мікроконтролера
        #include <p16f877a.inc>    ;підключення бібліотеки програм
tempa   EQU     h'020'             ;ініціалізація змінних
tempb   EQU     h'021'
        org     0x00               ;вектор скідання
        goto    begin              ;перехід до основної програми на мітку begin
        org     0x04               ;вектор переривання
        goto    prer               ;перехід до підпрограми переривань
        org     0x100              ;адреса розміщення програми
begin   clrw                       ;очищення акумулятора
        clrf    tempa              ;очищення змінних
        clrf    tempb
        movlw   b'00100101'        ;налаштування таймера TMR1
        movwf   T1CON
        movlw   b'11000000'        ;налаштування переривань периферійних модулів
        movwf   INTCON
        bsf     STATUS,RP0         ;перехід до банку 1
        movlw   b'00000001'        ;налаштування переривань таймера TMR1
        movwf   PIE1
        bcf     STATUS,RP0         ;перехід до банку 0
        movlw   d'240'             ;завдання початкового значення лічби
        movwf   TMR1L              ;молодший байт
        movlw   d'255'
        movwf   TMR1H              ;старший байт
m1      goto    m2                 ;цикл для формування машиних циклів для роботи таймера
m2      goto    m1
        nop
prer    movlw   d'1'               ;підпрограма переривань
        movwf   tempa
        bcf     PIR1,0             ;скидання прапора переривання від TMR0
        movlw   d'240'             ;завдання початкового значення лічби
        movwf   TMR1L
        movlw   d'255'
        movwf   TMR1H
        retfie                     ;повернення з підпрограми переривань з дозволом
                                   переривань
        nop                        ;немає операції
        end                        ;кінець програми
```

Рис. 6.5. Приклад програми застосування таймера TMR1

6.1.3. Модуль таймера TMR2

Модуль TMR2 – 8-розрядний таймер з програмованим розподільником та вихідним дільником, 8-розрядним регістром періоду PR2. TMR2 може бути опорним таймером для CCP модуля в ШІМ режимі. Регістри TMR2 доступні для запису/читання та очищаються за будь-якого виду скидання. Блок-схема таймера TMR2 наведена на рис.6.6.

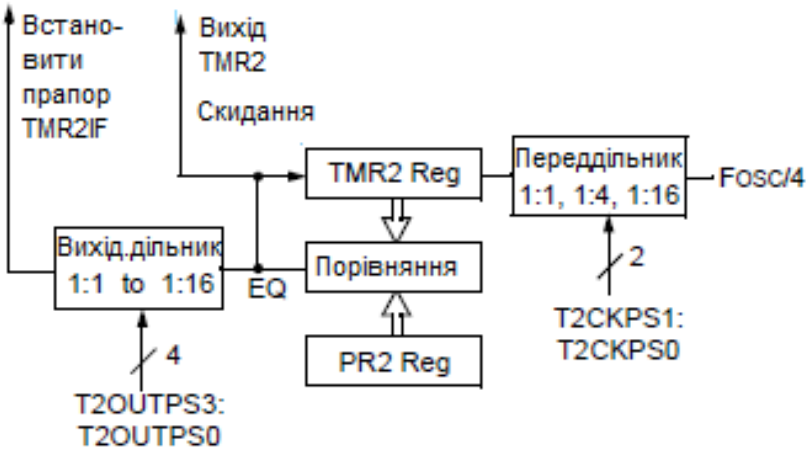


Рис. 6.6. Блок-схема таймера TMR2 [4]

У табл. 6.4 наведене призначення бітів регістра T2CON.

Таблиця 6.4

Регістр T2CON

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
Біт 7	Не реалізовані: читаються як «0»						
Біт 6-3	TOUTPS3: TOUTPS0: Вибір коефіцієнта вихідного дільника TMR2 0000 - 1:1 0001 - 1:2 : 1111 - 1:16						
Біт 2	TMR2ON: Увімкнення модуля TMR2 1 - увімкнений 0 - вимкнений						
Біт 1-0	T2CKPS1:T2CKPS0: Вибір коефіцієнта ділення переддільника TMR2 00 - 1:1 01 - 1:4 1x - 1:16						

Вхідний тактовий сигнал (Fosc/4) надходить через переддільник з програмованим коефіцієнтом ділення (1:1, 1:4 або 1:16), який визначається бітами T2CKPS1:T2CKPS0 (T2CON<1:0>).

TMR2 рахує, інкрементуючи від 00h до значення в регістрі PR2, потім скидається в 00h на наступному машинному циклі. Регістр PR2 доступний для запису та читання. Після скидання значення регістра PR2 дорівнює FFh.

Сигнал переповнення TMR2 проходить через вихідний 4-розрядний дільник з програмованим коефіцієнтом ділення (від 1:1 до 1:16 включно) для встановлення прапора TMR2IF у регістрі PIR1<1>.

Для зменшення енергоспоживання таймер TMR2 може бути вимкнений скиданням біта TMR2ON (T2CON<2>) у «0».

Модуль TMR2 може бути застосований для програмного вибору швидкості обміну даними модуля SSP.

Переддільник та вихідний дільник TMR2. Лічильник переддільника та вихідного дільника скидаються у разі:

- запису до регістру TMR2;
- запису до регістру T2CON;
- будь-якого виду скидання мікроконтролера (POR, BOR, скидання WDT або активний сигнал –MCLR).

Реєстр TMR2 не очищується під час запису T2CON.

Сигнал TMR2. Сигнал переповнення TMR2 (до вихідного переддільника) надходить до модуля SSP для управління швидкістю передавання.

У табл. 6.5 наведені регістри, які пов’язані з роботою таймера **TMR2**, а на рис. 6.7 – приклад програми застосування таймера.

Таблиця 6.5

Регістри і біти, які пов’язані з роботою TMR2

Адреса	Ім’я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh 8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR2	Регістр таймера 2								0000 0000	0000 0000
10h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	--00 0000	--uu uuuu
92h	PR2	Регістр періоду таймера 2								1111 1111	1111 1111

Позначення: - – не застосовується, читається як 0; u – не змінюється; x – не відомо; q – залежить від умов

```
Мітка    Мнемокод і операнди          Коментар
         list     p=16f877a           ;вибір типу мікроконтролера
         #include <p16f877a.inc>      ;підключення бібліотеки програм
tempa    EQU      h'020'              ;ініціалізація змінних
tempb    EQU      h'021'
         org      0x00                ;вектор скідання
         goto     begin               ;перехід до основної програми на мітку begin
         org      0x04                ;вектор переривання
         goto     prer                ;перехід до підпрограми переривань
         org      0x100               ;адреса розміщення програми
begin    clrw                         ;очищення акумулятора
         clrf     tempa               ;очищення змінних
         clrf     tempb
         movlw    b'00000100'         ;налаштування таймера TMR2
         movwf    T2CON
         movlw    b'11000000'         ;налаштування переривань периферійних модулів
         movwf    INTCON
         bsf      STATUS,RP0          ;перехід до банку 1
         movlw    b'00000010'         ;налаштування перериваньтаймера TMR2
         movwf    PIE1
         movlw    d'25'               ;завдання кінцевого значення лічби
         movwf    PR2
         bcf      STATUS,RP0          ;перехід до банку 0
         movlw    d'10'               ;завдання початкового значення лічби
         movwf    TMR2;               ;молодший байт
         movlw    d'255'              ;завдання початкового значення циклу
         movwf    tempb;
m1       decf     tempb               ;цикл для формування машиних циклів для роботи таймера
         btfss    STATUS,Z
         goto     m1
         nop
prer     movlw    d'1'                ;підпрограма переривань
         movwf    tempa
         bcf      PIR1,1              ;скидання прапора переривання від TMR0
         movlw    d'10';              ;завдання початкового значення лічби
         movwf    TMR2;
         retfie                       ;повернення з підпрограми переривань з дозволом
                                      ;переривань
         nop                          ;немає операції
         end                          ;кінець програми
```

Рис. 6.7. Приклад програми застосування таймера TMR2

Контрольні запитання та завдання

1. Для чого призначені таймери у складі мікроконтролера?
2. Які особливості таймерів мікроконтролера?
3. Яким чином можна налаштувати таймер TMR0?
4. Яким чином можна змінювати час затримки таймера?
5. Яким чином дізнаються, що час затримки таймера добіг до кінця?

6. Що треба зробити, щоб таймер підраховував не часові інтервали, а події на зовнішніх пристроях?

7. Яку розрядність мають таймери TMR0, TMR1, TMR2?

8. Навіщо потрібен переддільник таймера?

9. Які регістри застосовуються для налаштування таймера TMR2?

10. Для чого призначається регістр PR2 таймера TMR2?

6.2. Порти введення/виведення

6.2.1. Регістри PORTA та TRISA

Порт А – 6-розрядний порт введення/виведення. Усі канали PORTA мають відповідні біти напрямку до регістрів TRISA, що дозволяють налаштовувати канал як вхід або вихід. Запис «1» до TRISA переводить відповідний вихідний буфер у 3-й стан, як вхід. Запис «0» до регістру TRISA визначає відповідний канал як вихід, вміст регістра (заскочки) PORTA передається на вивід мікроконтролера (якщо вихідна заскочка підключена до виводу мікроконтролера). Блок-схеми виводів порту А наведені на рис.6.8.

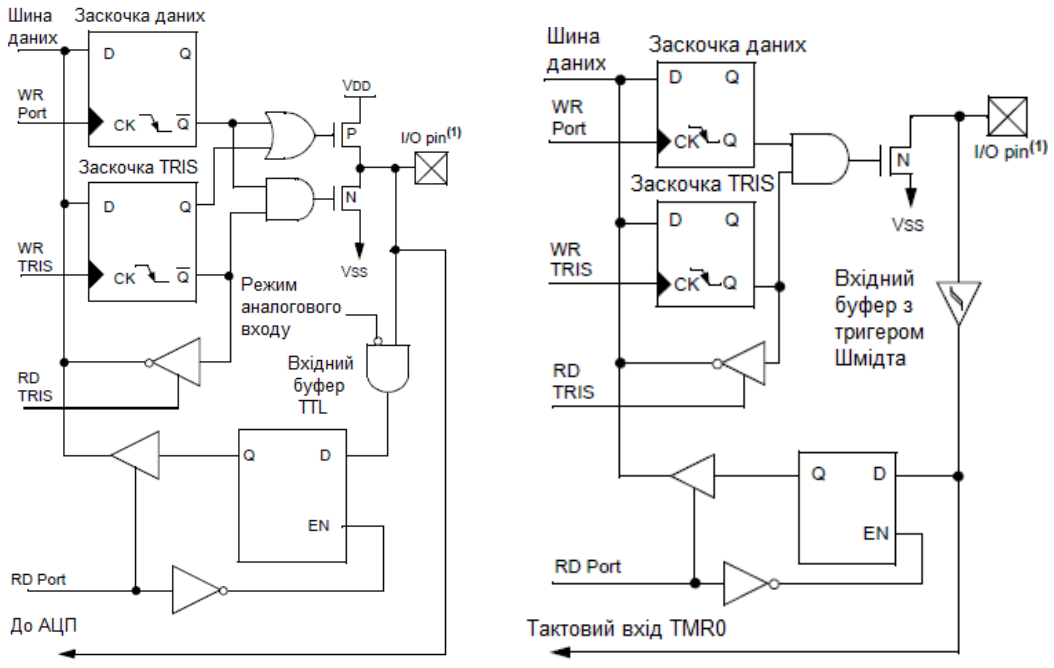


Рис. 6.8. Блок схеми виводів порту А [4]

Читання регістру PORTA повертає стан на виводах порту, а запис здійснюється у заскочку PORTA. Усі операції запису до порту виконуються за принципом "читання-модифікація-запис", тобто спочатку здійснюється читання стану виводів порту, а потім зміна та запис до заскочки.

RA4 – має тригер Шмідта на вході та відкритий стік на виході, мультиплікований із тактовим входом TOCKI. Всі інші канали порту А мають TTL буфер на вході та повнофункціональні вихідні КМОП буфери.

Канали порту А мультиплікові з аналоговими входами АЦП та аналоговим входом джерела опорної напруги U_{REF}. Біти керування режимами роботи каналів порту введення/виведення PORTA знаходяться в регістрі ADCON1. Функціональне призначення виводів порту А наведене у табл. 6.6.

Таблиця 6.6

Функціональне призначення виводів порту А

Позначення вивода	№ біта	Тип буфера	Опис
RA0/AN0	Біт 0	TTL	Двонаправлений порт введення/виведення або аналоговий вхід
RA1/AN1	Біт 1	TTL	Двонаправлений порт введення/виведення або аналоговий вхід
RA2/AN2	Біт 2	TTL	Двонаправлений порт введення/виведення або аналоговий вхід
RA3/AN3	Біт 3	TTL	Двонаправлений порт введення/виведення або аналоговий вхід
RA4/TOCKI	Біт 4	ST	Двонаправлений порт введення/виведення, може застосовуватися як TOCI, вихід з відкритим стоком
RA5/-SS/AN4	Біт 5	TTL	Двонаправлений порт введення/виведення або вхід вибору синхронного послідовного порту або аналоговий вхід

Після скидання по включенню живлення виводи налаштовуються як аналогові входи, а читання дає результат "0".

Біти регістру TRISA керують напрямком каналів PORTA, навіть коли вони використовуються як аналогові входи. Користувач повинен переконатися, що відповідні канали порту А налаштовані на вхід під час використання їх як аналогові входи.

Регістри та біти, що пов’язані з роботою PORTA, наведені в таблиці 6.7, а приклад програми ініціалізації порту А – на рис. 6.9.

Таблиця 6.7

Регістри та біти, що пов'язані з роботою порта А

Ад-реса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
05h	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	-	-	Регістр напрямку даних PORTA						--11 1111	--11 1111
9Fh	ADCON1		-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	0--- 0000

```
Мітка   Мнемокод   Операнди        Коментар
        bcf        STATUS,RP1
        bcf        STATUS,RP0      ;перехід до банку 0
        clrf       PORTA           ;ініціалізація заскочок PORTA
        bsf        STATUS,RP0      ;перехід до банку 1
        movlw      0x06
        movwf      ADCON1          ;призначення каналів порта А дискретними
        movlw      0xCF            ;встановлення напрямку виводів порта А
        movwf      TRISA           ;налаштування виводів RA<3:0> як входів
                                   ;налаштування виводів RA<5:4> як виходів
                                   ;біти TRISA <7:6> завжди читаються як «0»
```

Рис. 6.9. Приклад програми ініціалізації порту А

6.2.2. Регістри PORTB та TRISB

Порт В – 8-розрядний двонаправлений порт введення/виведення. Блок схеми виводів порту В наведені на рис.6.10. Функціональне призначення виводів порту В наведене в табл. 6.8.

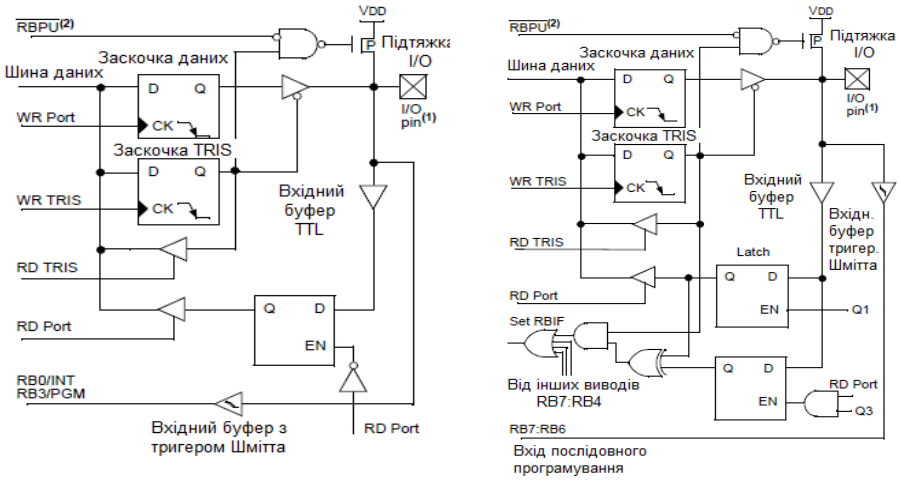


Рис. 6.10. Блок схеми виводів порту В [4]

Біти регістра TRISB визначають напрямок каналів порту. Установка бітів регістра TRISB в «1» переводить вихідний буфер у 3-й стан і відповідні виводи порта будуть входами. Запис «0» в регістр TRISB

налаштовує відповідний канал як вихід, вміст заскочки PORTB передається на вивід мікроконтролера (якщо вихідна заскочка підключена до виводу мікроконтролера).

Таблиця 6.8

Функціональне призначення виводів порту B

Позначення вивода	№ біта	Тип буфера	Опис
RB0/INT	Біт 0	TTL/ST	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора, вхід зовнішнього переривання
RB1	Біт 1	TTL	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора
RB2	Біт 2	TTL	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора
RB3/PGM	Біт 3	TTL	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора або вхід програмування в режимі LVP
RB4	Біт 4	TTL	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора та перериванням за зміненням вхідного сигналу
RB5	Біт 5	TTL	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора та перериванням за зміненням вхідного сигналу
RB6/PGC	Біт 6	TTL/ ST	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора та перериванням за зміненням вхідного сигналу. Тактовий вхід у режимі програмування
RB7/PGD	Біт 7	TTL/ ST	Двонаправлений порт введення/виведення з програмним включенням підтягуючого резистора та перериванням за зміненням вхідного сигналу. Виведення даних у режимі програмування

Позначення: ST – вхід з тригером Шмідта; TTL – вхідний буфер TTL

Примітки: 1. Вхідний буфер з тригером Шмідта при застосуванні зовнішніх переривань.

2. Вхідний буфер з тригером Шмідта при роботі в режимі послідовного програмування.
3. Низьковольтне програмування (LVP) ICSP дозволено за замовчуванням, що вимикає функцію цифрового порту введення/виведення RB3. Для застосування RB3 як цифрового порту введення/виведення необхідно вимкнути режим низьковольтного програмування.

Три виводи порту B мультиплікові зі схемою низьковольтного програмування: RB3/PGM, RB6/PGC, RB7/PGD.

До кожного виводу порту B підключений внутрішній підтягуючий резистор. Біт –RBPU (OPTION_REG <7>) визначає підключені (-RBPU=0) чи ні (-RBPU=1) підтягуючі резистори. Підтягуючі резистори автоматично відключаються після скидання по ввімкненню живлення POR і коли канали порту налаштовуються на вихід.

Чотири канали порту B RB7:RB4, які налаштовані на вхід, можуть генерувати переривання зміни логічного рівня сигналу на вході. Якщо один із каналів RB7:RB4 налаштований на вихід, то він не може бути джерелом переривань. Сигнал на виводах RB7:RB4 порівнюється зі значенням, збереженим під час останнього читання регістра PORTB. У разі розбіжності одного зі значень встановлюється прапор RBIF (INTCON<0>) та, якщо дозволено, генерується переривання.

Це переривання може вивести мікроконтролер з режиму SLEEP.

У підпрограмі обробки переривань необхідно здійснити такі дії:

- читання або запис до регістру PORTB, виключивши невідповідність збереженного значення на виводах RB7:RB4 поточному значенню;

- скидання прапора RBIF «0».

Невідповідність збереженого значення сигналу на вході порту B завжди встановлює біт RBIF «1». Читання з регістру PORTB перерве умову невідповідності та дозволить скинути прапор RBIF у «0».

Переривання через зміну сигналів на входах рекомендується використовувати для визначення натискання клавіш, коли порт B повністю задіяний для реалізації клавіатури. Не рекомендується опитувати регістр PORTB під час використання переривань через зміни вхідного сигналу.

Переривання зміни сигналу на входах порту B та програма перемикання конфігурації цих каналів дозволяє реалізувати простий інтерфейс обслуговування клавіатури з виходом з режиму SLEEP шляхом натисканням на клавіші.

RB0/INT – вхід зовнішнього джерела переривань, що налаштовується бітом INTEDG (OPTION_REG<6>).

Регістри та біти, що пов’язані з роботою порту B, наведені у табл. 6.9.

Таблиця 6.9

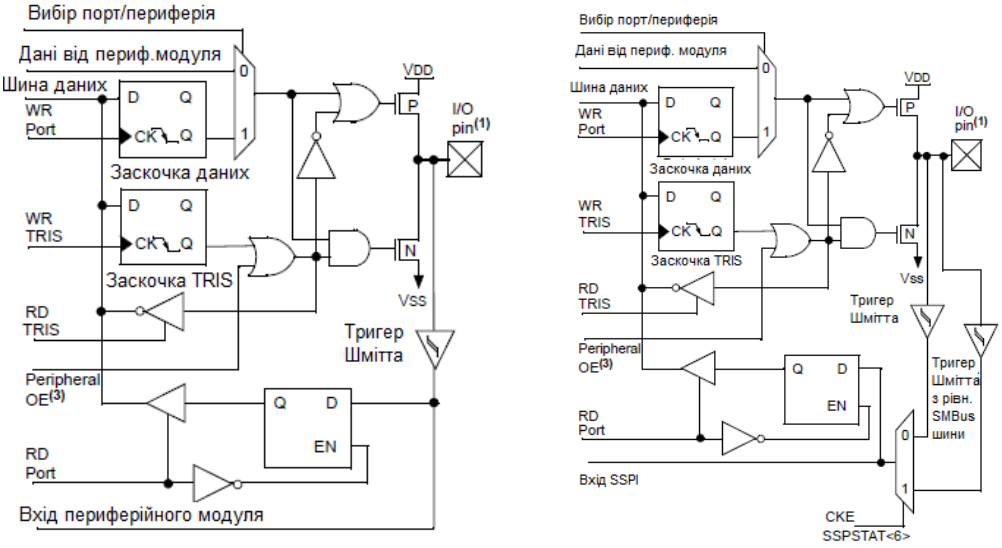
Регістри та біти, що пов'язані з роботою порту B

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
06h,106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h,186h	TRISB	Регістр напрямку даних PORTB								1111 1111	1111 1111
81h,181h	OPTION _REG	-RBPU	INTEDG	TOCS	TOSE	PSA	PC2	PC1	PC0	1111 1111	1111 1111

6.2.3. Регістри PORTC та TRISC

Порт C – 8-розрядний двонаправлений порт вводу/виводу. Біти регістру TRISC визначають напрямок каналів порту. Установка біта в «1» регістру TRISC переводить вихідний буфер у 3-й стан. Запис «0» в регістр TRISC налаштовує відповідний канал як вихід, вміст заскочки PORTC передається на вихід мікроконтролера (якщо вихідна заскочка підключена до виводу мікроконтролера).

Блок схеми виводів порту C наведені на рис. 6.11.



Примітки: 1. Виводи портів мають захисні діоди, які підключені до V_{DD} і V_{SS}

2. Сигнал режиму каналів – вивід застосовується периферійним модулем або як цифровий порт вводу/виводу.

3. Сигнал дозволу (OE) від периферійного модуля, налаштовувати канал як вихід

Рис. 6.11. Блок схеми виводів порту C [4]

Функціональне призначення виводів порту B наведене в табл. 6.10.

Таблиця 6.10

Функціональне призначення виводів PORTC

Позначення вивода	№ біта	Тип буфера	Опис
RC0/T1OSO/ T1CKI	Біт 0	ST	Двонаправлений порт введення/виведення або вихід генератора TMR1/ вхід тактового генератора
RC1/T1OSI/ CCP2	Біт 1	ST	Двонаправлений порт введення/виведення або вхід генератора TMR1 або вхід захоплення 2/вихід порівняння 2/вихід ШИМ 2.
RC2/CCP1	Біт 2	ST	Двонаправлений порт введення/виведення або вхід захоплення 1/вихід порівняння 1/вихід ШИМ 1.
RC3/SCK/SCL	Біт 3	ST	Двонаправлений порт введення/виведення або вхід/вихід тактового сигналу модуля MSSP в SPI, I^2C режимі.
RC4/SD/SDA	Біт 4	ST	Двонаправлений порт введення/виведення або вхід даних в режимі SPI або вхід/вихід даних у режимі I^2C.
RC5/SDO	Біт 5	ST	Двонаправлений порт введення/виведення або вихід даних у режимі SPI.
RC6/TX/CK	Біт 6	ST	Двонаправлений порт введення/виведення або вихід передавача USART в асинхронному режимі або лінія тактового сигналу USART у синхронному режимі.
RC7/RX/DT	Біт 7	ST	Двонаправлений порт введення/виведення або вхід приймача USART в асинхронному режимі або лінія даних USART в синхронному режимі.

Позначення: ST – вхід з тригером Шмідта.

Виводи порту С мультиплексовані з кількома периферійними модулями. На каналах порту С присутній вхідний буфер із тригером Шмідта.

Коли модуль MSSP включений у режимі I2C, виводи порту С <4:3> можуть підтримувати рівні вихідних сигналів за специфікацією I2C чи SMBus залежно стану біта CKE (SSPSTAT<6>).

При використанні периферійних модулів необхідно відповідним чином налаштовувати біти регістра TRIS для кожного вивода порту С. Деякі периферійні модулі скасовують дію бітів TRISC примусово налаштовуючи вивід на вхід або вихід. У зв'язку з чим не рекомендується використовувати команди читання-модифікація-запис з регістром TRISC.

Регістри та біти, що пов’язані з роботою порту C, наведені в таблиці 6.11.

Таблиця 6.11

Регістри та біти, що пов’язані з роботою PORTC

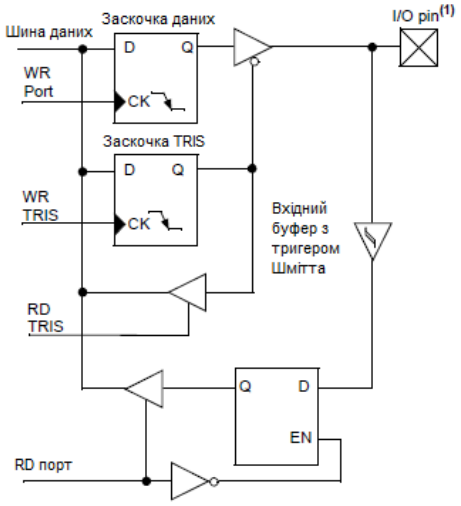
Адреса	Ім’я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	Регістр напрямку даних PORTC								1111 1111	1111 1111

6.2.4. Регістри PORTD та TRISD

Порт D – 8-розрядний двонаправлений порт введення/виведення.

Блок схеми виводів порту D наведені на рис. 6.12. Функціональне призначення виводів порту D наведене в табл. 6.12. Біти регістра TRISD визначають напрямок каналів порту.

Порт D може працювати як 8-розрядний мікроконтролерний порт (відомий паралельний порт), якщо біт PSPMODE (TRISE<4>) встановлений у «1». У режимі керованого паралельного порту до входів підключені буфери TTL.



Примітка. Виводи портів мають захисні діоди, які підключені до V_{DD} і V_{SS}

Рис. 6.12. Блок схеми виводів порту D [4]

Таблиця 6.12

Функціональне призначення виводів PORTD

Позначення вивода	№ біта	Тип буфера	Опис
RD0/PSP0	Біт 0	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 0
RD1/ PSP1	Біт 1	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 1
RD2/ PSP2	Біт 2	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 2
RD3/ PSP3	Біт 3	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 3
RD4/ PSP4	Біт 4	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 4
RD5/ PSP5	Біт 5	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 5
RD6/ PSP	Біт 6	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 6
RD7/ PSP7	Біт 7	ST/TTL	Двонаправлений порт введення/виведення або вивід веденого паралельного порту біт 7

Регістри та біти, що пов’язані з роботою порту D, наведені у табл. 6.13.

Таблиця 6.13

Регістри та біти, що пов’язані з роботою PORTD

Адреса	Ім’я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	Регістр напрямку даних PORTD								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	-	Рег.напр.даних PORTE			0000 -111	0000 -111

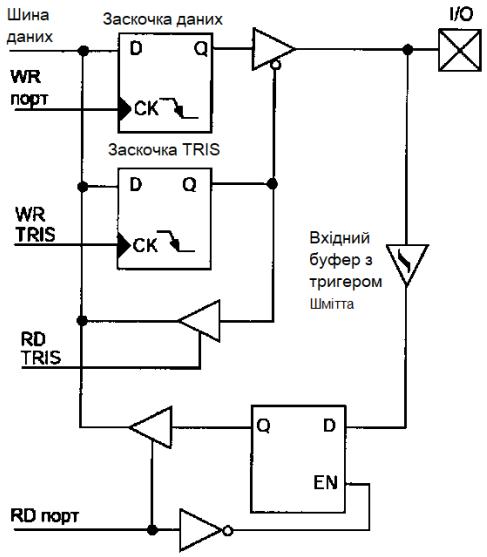
6.2.5. Регістри PORTE та TRISE

Порт Е має три виводи (RE0/-RD/AN5, RE1/-WR/AN6, RE2/-CS/AN7), що індивідуально налаштовуються на вхід або вихід. Виводи порту Е мають вхідний буфер Шмідта.

Блок схеми виводів порту Е наведені на рис.6.13. Функціональне призначення виводів порту D наведене у табл. 6.14.

Канали порту Е стануть керуючими виводами веденого паралельного порту, коли біт PSPMODE(TRISE<4>) встановлений в «1». У цьому режимі

біти TRISE<2:0> мають бути встановлені в «1». У регістрі ADCON1 необхідно також налаштувати виводи порту E як цифрові канали вводу/виводу. У режимі керованого паралельного порту до виводів порту E підключені вхідні буфери TTL.



Примітка. Виводи портів мають захисні діоди, які підключені до V_{DD} та V_{SS}

Рис 6.13. Блок схеми виводів порту E [4]

Таблиця 6.14

Функціональне призначення виводів порту E

Позначення виводу	№ біта	Тип буфера	Опис
RE0/-RD/AN5	Біт 0	ST/TTL	Двонаправлений порт вводу/виводу або вхід управління читанням веденого паралельного порту або аналоговий вхід: -RD 1 - очікування 0 - операція читання. Заскочка PORTE підключена до виводів PORTE (якщо –CS = 0)
RE1/ -WR/AN6	Біт 1	ST/TTL	Двонаправлений порт вводу/виводу або вхід управління записом веденого паралельного порту або аналоговий вхід: -WR 1 - очікування 0 - операція запису. Дані з виводів PORTE зберігаються у внутрішній заскочці PORTE (якщо –CS - 0)
RE2/ -CS/AN7	Біт 2	ST/TTL	Двонаправлений порт вводу/виводу або вхід вибору мікросхеми веденого паралельного порту або аналоговий вихід: -CS 1 - мікросхема не обрана 0 - мікросхема обрана

Позначення: ST – вхід з тригером Шмідта; TTL – вхідний буфер TTL

Примітка: В режимі цифрового введення/виведення підключений буфер з тригером Шмідта, а в режимі паралельного порту підключений вхідний буфер TTL

Виводи порту E мультиплiковані з аналоговими входами. Коли канали порту E налаштовані як аналогові входи, біти регістру TRISE управляють напрямом даних порту E (читання даватиме результат «0»).

Після скидання, увімкнення живлення виводи налаштовуються як аналогові входи, а читання дає результат «0».

Функціональне призначення бітів регістру TRISE наведене у табл. 6.15. Регістри та біти, що пов'язані з роботою порту E, наведені у табл. 6.16.

Таблиця 6.15

Регістр TRISE

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
IBF	OBF	IBOV	PSPMODE		BIT2	BIT1	BIT0

Біти управління і статусу веденого паралельного порту	
Біт 7	IBF: Біт статусу прийомного буфера 1 - байт даних отриманий 0 - байт даних не був отриманий
Біт 6	OBF: Біт статусу передаючого буфера 1 - попередньо записаний байт ще не прочитаний 0 - вихідний буфер був прочитаний
Біт 5	IBOV: прапор переповнення прийомного буфера 1 - відбувся новий запис, а попередній байт не був прочитаний (скидається програмно) 0 - переповнення не було
Біт 4	PSPMODE: Режим роботи PORTD 1 - PORTD працює як відомий паралельний порт 0 - PORTD працює в режимі цифрових каналів вводу/виводу
Біт 3	Не реалізований: читається як «0»
Біт 2	BIT2: Напрямок вивода RE2/-CS/AN7 1 - вхід 0 - вихід
Біт 1	BIT1: Напрямок вивода RE1/-WR/AN6 1 - вхід 0 - вихід
Біт 0	BIT0: Напрямок вивода RE0/-RD/AN5 1 - вхід 0 - вихід

Таблиця 6.16

Регістри та біти, що пов'язані з роботою PORTE

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
09h	PORTE	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
9Fh	TRISE	IBF	OBF	IBOV	PSPMODE	-	Рег.напр.даних PORTE			0000 -111	0000 -111
9F	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	0--- 0000

6.2.6. Програмування портів

Мікроконтролери P16F877a для взаємодії із зовнішними пристроями мають 5 портів: A, B, C, D, E. Виводи мікроконтролера – це виводи портів для підключення зовнішних пристроїв, за виключенням виводів для підключення живлення і скидання. Порти можуть приймати дискретні вхідні сигнали та формувати дискретні вихідні сигнали ТТЛ рівня. Порт А та порт Е можуть приймати аналогові сигнали в діапазоні 0-5 В. Формувати аналогові вихідні сигнали ці мікроконтролери не можуть.

Порти двонаправлені,один і той вивід мікроконтролера може бути призначений як для вводу сигналу, так і для виводу. Для налаштування та роботи з портами мікроконтролери мають регістри спеціального призначення, які називаються PORTx та TRISx. Регістри PORTx призначені для розміщення даних, які передаються або приймаються. Регістри TRISx призначені для вибору напрямку порту. Якщо в відповідний біт регістра TRISx буде записана «1», то цей вивід мікроконтролера і порта буде входом, якщо – «0», то це буде вихід.

Робота з портами полягає в попередньому налаштуванні порта, а потім опитуванні порта. Налаштування порта полягає в становленні бітів регістра TRISx в відповідні значення, або «0» або «1». Якщо «0», то цей вивід порта буде виходом, якщо «1», то цей вивід порта буде входом.

Нижче наведені стани бітів регістрів портів А та D (табл.6.17-6.20) відповідно схеми підключення сигналів до мікроконтролера (рис.6.14).

До порта А на вивід RA0 підключений аналоговий сигнал з потенціометра R1, до виводів RA1, RA2, RA3 підключені дискретні сигнали з кнопок S1, S2, S3, тому виводи RA0, RA1, RA2, RA3 будуть входами. Якщо кнопка S1 не натиснута, то через дільник R6 та кнопку S1 на вхід RA1 поступає сигнал напруги 5 В, що відповідає логічній «1» і в першому біті регістра PORTA з'явиться логічна «1». Якщо кнопка S1 натиснута, то через дільник R6 та кнопку S1 на вхід RA1 поступає сигнал напруги 0 В, тому що

кнопка підключає вхід RA1 до загальної точки блока живлення, що відповідає логічному «0» та в першому біті регістра PORTA з'явиться логічний «0».

До виводів RD0, RD1, RD2, RD3 підключені світлодіоди, тому ці виводи будуть виходами.

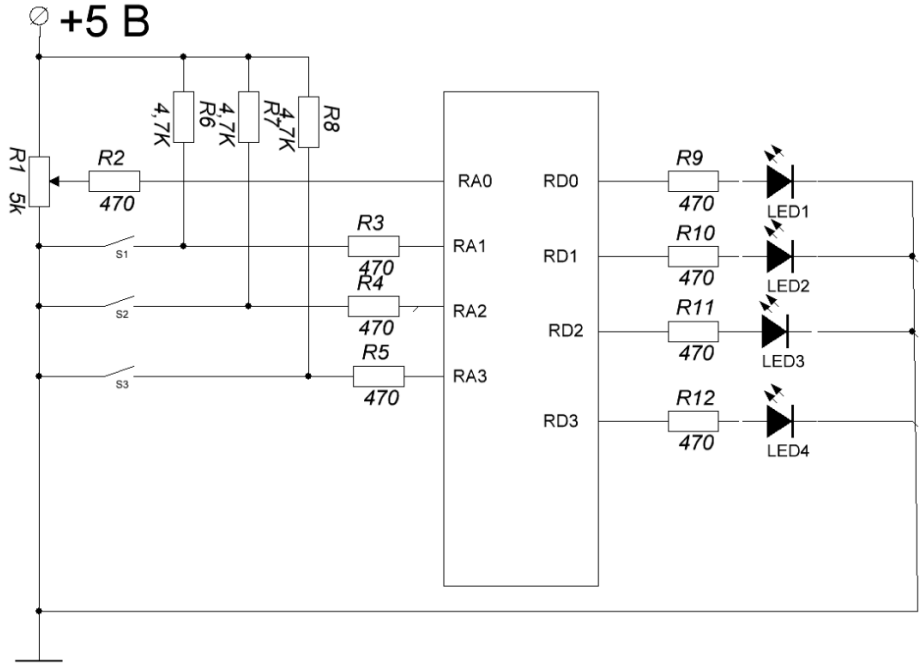


Рис 6.14. Схема підключення сигналів до виводів мікроконтролера

Регістри PORTA, PORTD знаходяться в банку «0», регістри TRISA, TRISD знаходяться в банку «1», тому для роботи з цими регістрами треба переходити з банку в банк, що відбувається за допомогою регістра STATUS та бітів регістра RP0, RP1. Крім того, для роботи з портом А треба обрати, які входи будуть аналогові, а які дискретні, потім відповідно таблиці регістра ADCON1 обрати двійкове число, яке відповідає необхідної комбінації аналогових і дискретних сигналів, та записати його в регістр ADCON1.

Якщо в нашому випадку вхід RA0 – аналоговий, а інши входи – дискретні, то в регістр ADCON1 треба записати двійкове число «00001110».

Завдання для програми буде полягати в наступному. Треба натиснути на кнопку S1 та запалити світлодіод LED1.

Для налаштування портів відповідно рис.6.14 треба записати в регістр TRISA двійкове число «00001111», тому що до виводи RA0-RA3 є входами, RA4-RA5 – виходами. В регістр TRISD треба записати двійкове число «00000000», тому що виводи RD0-RD7 є виходами.

Для опитивання кнопки треба застосувати команду перевірки стану біта 1 регістра PORTA - btfsc PORTA,1. Якщо кнопка не натиснута, цей біт має значення «1» , якщо натиснута – «0». Тому треба перевіряти стан біта 1 регістра PORTA на появу «0». Якщо цей біт дорівнює «0», то це означає, що кнопка S1 натиснута та треба запалювати світлодіод LED1.

Для запалювання свілодіода LED1, який підключений до виводу RD0, треба в біт 0 регістра PORTD записати «1», після цього на цьому виводі з’явиться напруга 5 В та світлодіод буде запалений.

Нижче наведена програма опитування кнопки S1 та запалювання світлодіода LED1.

Таблиця 6.17

Біти регістра PORTA

Номер виводу МК			7	6	5	4	3	2
Назва виводу МК			RA5	RA4	RA3	RA2	RA1	RA0
Номер біта регістра			5	4	3	2	1	0
Номер каналу АЦП			AN4		AN3	AN2	AN1	AN0

Таблиця 6.18

Біти регістра TRISA

Назва виводу МК			RA5	RA4	RA3	RA2	RA1	RA0
Номер біта регістра			5	4	3	2	1	0
Значення бітів			0	0	1	1	1	1
Призначення виводу			Вихід	Вихід	Вхід дискр.	Вхід дискр.	Вхід дискр.	Вхід аналог.

Таблиця 6.19

Біти регістра TRISD

Назва виводу МК	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
Номер біта регістра	7	6	5	4	3	2	1	0
Значення бітів	0	0	0	0	0	0	0	0
Призначення виводу	Вихід	Вихід	Вихід	Вихід	Вихід	Вихід	Вихід	Вихід

Таблиця 6.20

Біти регістр PORTD

Номер виводу МК	30	29	28	27	22	21	20	19
Назва виводу МК	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
Номер біта регістра	7	6	5	4	3	2	1	0
Значення бітів	0	0	0	0	0	0	0	1

Приклад програми застосування портів наведено на рис. 6.15.

```
Мітка   Мнемокод і операнди          Коментар
        list    p=16f877a            ;вибір типу мікроконтролера
        #include <p16f877a.inc>      ;підключення бібліотеки програм
        org     0x00                 ;вектор скідання
        goto    begin                ;перехід до основної програми на мітку begin
        org     0x100                ;адреса розміщення програми
begin   clrw                         ;очищення акумулятора
        clrf    PORTA                ;очищення портів
        clrf    PORTD
        bsf     STATUS,RP0           ;перехід до банку 1
        movlw   b'00011111'          ;налаштування виводів порта A, як входів
        movwf   TRISA
        Movlw   b'00001110'          ;виводи порта A: RA0 – аналоговий, RA1-RA3 -
                                     ;дискретні
        movwf   ADCON1
        movlw   b'00000000'          ;налаштування виводів порта D, як виходів
        movwf   TRISD
        bcf     STATUS,RP0           ;перехід до банку 0
m1      btfsc   PORTA,2              ;цикл для опитування 2-го біта порта A, до якого
                                     ;підключена кнопка. При натисненні на  кнопку біт 2
                                     ;приймає значення «0».
        goto    m1                   ;якщо біт 2 не дорівнює 0, то виконується команда
                                     goto ;m1 і здійснюється перехід на мітку m1, якщо
                                     дорівнює 0, ;то пропускається наступна команда goto
                                     m1 і виконується ;перехід  на команду bsf PORTD,1
        bsf     PORTD,1              ;запалення світлодіода LED1
        nop                          ;немає операції
        end                          ;кінець програми
```

Рис. 6.15. Програма застосування портів

Контрольні запитання та завдання

1. Навіщо потрібні порти в комп’ютерах?
2. Скільки портів має мікроконтролер PIC16F877a?
3. Які регістри спеціального призначення має кожен порт?
4. Наведіть призначення регістра TRIS.
5. Наведіть призначення регістра PORT.
6. Який порт може приймати аналогові сигнали?
7. В чому полягає налаштування порта?
8. В яких банках знаходяться регістри портів?
9. Що треба зробити щоб порт був вхідним?
10. Що треба зробити щоб порт став вихідним?

7. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ. СИСТЕМИ ВВЕДЕННЯ/ВИВЕДЕННЯ

При розробці програм для мікроконтролерів дуже часто виникає потреба у виведенні інформації на цифрові індикатори та уведенні даних з технологічної клавіатури.

7.1. Цифрові індикатори

Найбільш поширеним типом індикаторів є сьомисегментні індикатори, які формують зображення з сегментів і призначені для виведення цифрової інформації та деяких символів. Зображення створюється подачею напруги на відповідні сегменти, які починають світитися та показувати потрібну цифру. На вхід цих індикаторів подається сьомисегментний код, який может формуватися дискретними виходами мікроконтролера (табл. 7.1).

Таблиця 7.1

Таблиця сьомисегментних кодів

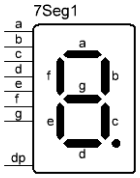
	dp	G	F	E	D	C	B	A	h
	7	6	5	4	3	2	1	0	
		0	0	0	0	0	0	0	
0		**0**	**1**	**1**	**1**	**1**	**1**	**1**	3F
1		**0**	**0**	**0**	**0**	**1**	**1**	**0**	06
2		**1**	**0**	**1**	**1**	**0**	**1**	**1**	5B
3		**1**	**0**	**0**	**1**	**1**	**1**	**1**	4F
4		**1**	**1**	**0**	**0**	**1**	**1**	**0**	66
5		**1**	**1**	**0**	**1**	**1**	**0**	**1**	6D
6		**1**	**1**	**1**	**1**	**1**	**0**	**1**	7D
7		**0**	**0**	**0**	**0**	**1**	**1**	**1**	07
8		**1**	**1**	**1**	**1**	**1**	**1**	**1**	7F
9		**1**	**1**	**0**	**1**	**1**	**1**	**1**	6F

Схема підключення цифрових індикаторів до мікроконтролера наведена на рис.7.1. Ця схема реалізує динамічний вивід інформації на індикатори. Сьомисегментний код з мікроконтролера подається на всі індикатори одночасно і для того, щоб вони не виводили однакову

інформацію на всі індикатори треба програмно синхронізувати вивід даних на кожний розряд індикатора. Це робиться, наприклад, наступним чином. Потрібно вивести значення сьомисегментного кода в порт D (рис.7.1) і зберегти в зовнішній 8-розрядній D-типу заскочці 74HC573. Потім треба обрати десятковий розряд, куди треба вивести значення сьомисегментного коду, що робиться теж через порт D і іншу заскочку 74HC573. Далі все це повторити для наступного десяткового розряду. Тобто ці індикатори засвічуються по черзі з високою частотою і це призводить до того, що зображення на індикаторах на всіх розрядах видається постійним. Це так званий динамічний вивід інформації, який дозволяє виводити інформацію з мікроконтролера семісегментним кодом і застосовувати мінімальну кількість портів мікроконтролера.

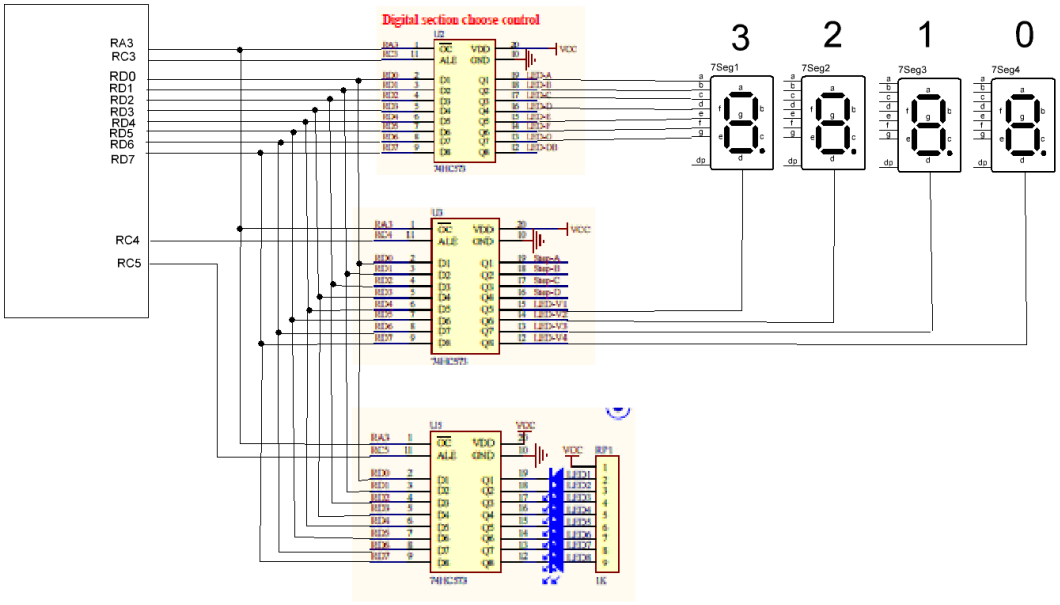


Рис.7.1. Схема підключення цифрових індикаторів до мікроконтролера

На рис.7.2 наведене статичне виведення даних на цифрові індикатори, коли кожному розряду індикатора відповідає один напівбайт порту. Інформація виводиться 4-х розрядним паралельним двійковим кодом, який через дешифратори перетворюється в сьомисегментний код і подається на відповідний десятковий розряд індикатора.

Нижче наведена програма виведення інформації на цифрові індикатори згідно з рис.7.2. У програмі застосований нульовий розряд індикатора, який підключений до виводів RB0 – RB3 порту B. На цифрові індикатори будуть виводитися цифри. В програмі задіяні кнопки S1, S2 та S3, які підключені до виводів RA1, RA2 та RA3 (рис.6.14).

При натисканні на кнопок RA1 та RA2 на цифровий індикатор виводяться цифри «0» або «1» залежно від того, яка кнопка натиснута.

Якщо натиснути на кнопку RA3, програма перейде до послідовного виведення на індикатор цифр від «2» до «9», а далі знов повернеться на опитування кнопок.

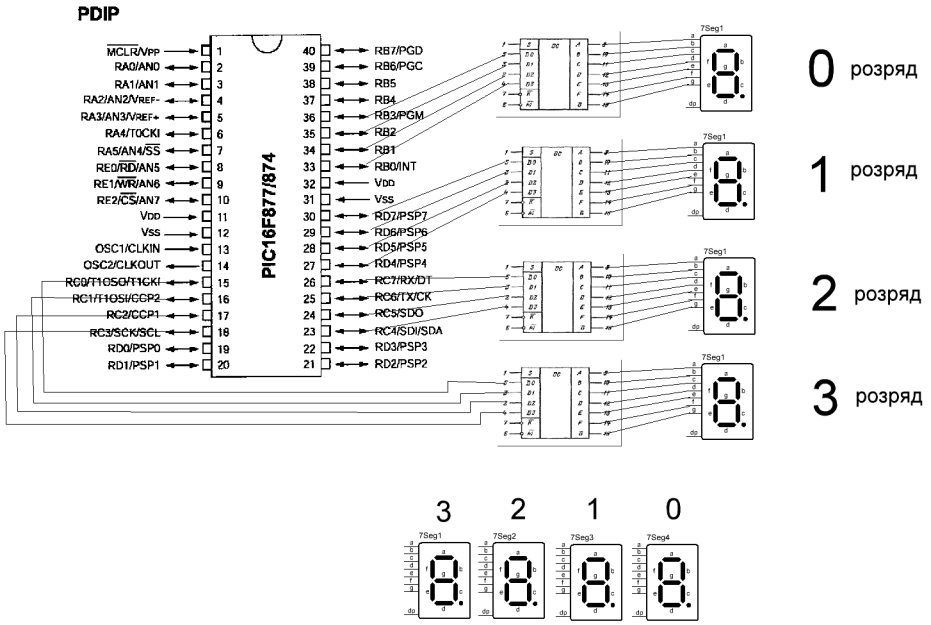


Рис.7.2.Схема підключення цифрових індикаторів до мікроконтролера

Приклад програми статичного виведення на цифровий індикатор наведений на рис. 7.3.

Мітка	Мнемокод і операнди		Коментар
	list	p=16f877a	;вибір типу мікроконтролера
	#include	<p16f877a.inc>	;підключення бібліотеки програм
temp1	equ	h’020’	
	org	0x00	;вектор скідання
	goto	begin	;перехід до основної програми на мітку begin
	org	0x100	;адреса розміщення програми
begin	clrw		;очищення акумулятора
	clrf	temp1	
	clrf	PORTB	;очищення портів
	clrf	PORTA	
	bsf	STATUS,RP0	;перехід до банку 1
	movlw	b’00001111’	;налаштування виводів порта E, як входів
	movwf	TRISA	
	Movlw	b’00001110’	;виводи порта E дискретні
	movwf	ADCON1	
	movlw	b'00000000'	;налаштування виводів порта B, як виходів
	movwf	TRISB	
	bcf	STATUS,RP0	;перехід до банку 0
m10	btfsc	PORTA,1	;опитування кнопки S1, вхід RA1

```
        call    m0              ;перехід до підпрограми вивода цифри «0»
        btfsc   PORTA,2         ;опитування кнопки S2, вхід RA2
        call    m1              ;перехід до підпрограми вивода цифри «1»
        btfsc   PORTA,3         ;опитування кнопки S3, вхід RA3
        goto    m11             ;перехід до викликів підпрограм, які послідовно виводять
                                ;цифри 3-9
        goto    m10             ;повернення на початок опитуввання кнопок
m11     call    m3              ;перехід до підпрограми вивода цифри «3»
        call    m4              ;перехід до підпрограми вивода цифри «4»
        call    m5              ;перехід до підпрограми вивода цифри «5»
        call    m6              ;перехід до підпрограми вивода цифри «6»
        call    m7              ;перехід до підпрограми вивода цифри «7»
        call    m8              ;перехід до підпрограми вивода цифри «8»
        call    m9              ;перехід до підпрограми вивода цифри «9»
        goto    m10
m0      bcf     PORTB,0         ;підпрограма виведення цифри «0»
        bcf     PORTB,1
        bcf      PORTB,2
        bcf     PORTB,3
        return
        nop
m1      bsf     PORTB,0         ;підпрограма виведення цифри «1»
        bcf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
        nop
m2      bcf     PORTB,0         ;підпрограма виведення цифри «2»
        bsf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
        nop
m3      bsf     PORTB,0         ;підпрограма виведення цифри «3»
        bsf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
        nop
m4      Bcf     PORTB,0         ;Підпрограма виведення цифри «4»
        Bcf     PORTB,1
        Bcf     PORTB,2
        Bcf     PORTB,3
        return
m5      bsf     PORTB,0         ;підпрограма виведення цифри «5»
        bcf     PORTB,1
        bsf     PORTB,2
        bcf     PORTB,3
        return
        nop
m6      bcf     PORTB,0         ;підпрограма виведення цифри «6»
        bsf     PORTB,1
        bsf     PORTB,2
```

```
        bcf     PORTB,3
        return
        nop
m7      bsf     PORTB,0         ;Підпрограма виведення цифри «7»
        bsf     PORTB,1
        bsf     PORTB,2
        bcf     PORTB,3
        return
        nop
m8      bcf     PORTB,0         ;підпрограма виведення цифри «8»
        bcf     PORTB,1
        bcf     PORTB,2
        bsf     PORTB,3
        return
        nop
m9      bsf     PORTB,0         ;підпрограма виведення цифри «9»
        bcf     PORTB,1
        bcf     PORTB,2
        bsf     PORTB,3
        return
        nop
        nop                     ;немає операції
        end                     ;кінець програми
```

Рис. 7.3. Програма статичного виведення на цифровий індикатор

7.2. Технологічні клавіатури

За способом підключення до мікроконтролера існують два основних типи технологічних клавіатур: клавіатури із загальною точкою та матричні клавіатури.

Клавіші клавіатури із загальною точкою підключаються до дискретних входів мікроконтролера: кожна клавіша до окремого дискретного входу і тому, якщо застосовується 16-ти клавішна клавіатура, для уведення даних треба задіяти 16 дискретних входів, тобто два порти. Це не завжди можливо через те, що в програмі потрібно виконувати ще й інші функції. Приклад клавіатури з загальною точкою наведений на рис. 6.14.

Клавіатури із загальною точкою мають простий спосіб опитування, який полягає в послідовному опитуванні кожного дискретного входу, поки не буде визначена клавіша, що натиснута, а потім виконується функція, яка закріплена за цією клавішею.

Простота опитування клавіші, а відповідно й простота програми опитування, є перевагою даної клавіатури. Проте велика кількість задіяних

клавіш та портів є недоліком. На рис. 7.5 наведений приклад програми опитування клавіатури із загальною точкою.

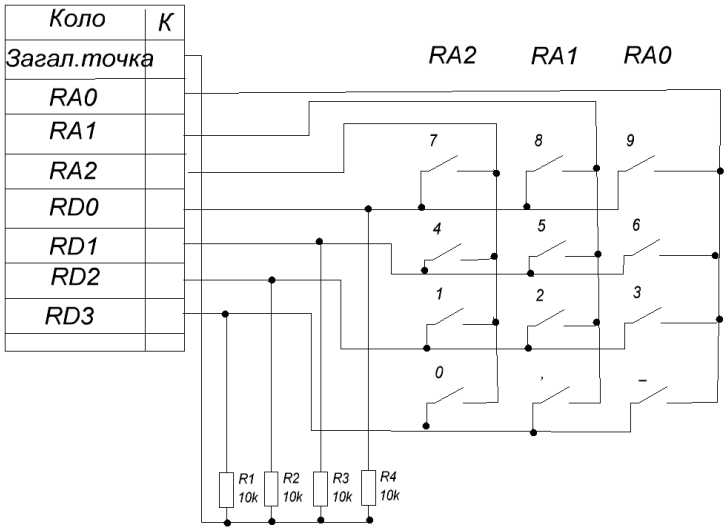


Рис. 7.4. Схема підключення матричної клавіатури

```
Мітка   Мнемокод і операнди        Коментар
        list    p=16f877a          ;вибір типу мікроконтролера
        #include <p16f877a.inc>    ;підключення бібліотеки програм
temp1   equ     h’020’
        org     0x00               ;вектор скідання
        goto    begin              ;перехід до основної програми на мітку begin
        org     0x100              ;адреса розміщення програми
begin   clrw                       ;очищення акумулятора
        clrf    temp1              ;очищення  temp1
        clrf    PORTB              ;очищення  портів
        clrf    PORTA              ;очищення  портів
        bsf     STATUS,RP0         ;перехід до банку 1
        movlw   b’00000111’        ;налаштування виводів порта А, як входів
        movwf   TRISA
        movlw   b’00000110’        ;виводи порта А дискретні
        movwf   ADCON1
        movlw   b'00000000'        ;налаштування виводів порта В, як виходів
        movwf   TRISB
        bcf     STATUS,RP0         ;перехід до банку 0
        btfsc   PORTA,1            ;опитування кнопки S1
        call    m0                 ;перехід до підпрограми m0
        btfsc   PORTA,2            ;опитування кнопки S2
        call    m1                 ;перехід до підпрограми m1
        btfsc   PORTA,3            ;опитування кнопки S3
        goto    m2
        goto    m10
        call    m3
        call    m4
        …
        nop
        end
```

Рис. 7.5. Програма опитування клавіатури із загальною точкою

Матрична клавіатура застосовує удвічі меншу кількість дискретних виводів мікроконтролера, але потребує більш складної програми. Підключення матричної клавіатури наведено на рис. 7.4.

Програма опитування матричної клавіатури та виведення значення цифри на цифровий індикатор реалізована відповідно рис. 7.6 та рис. 7.3.

Опитування матричної клавіатури, яка підключена до виводів порту А та порту D, здійснюється наступним чином.

Спочатку логічна «1» подається на вивід RA0 (виходи порту А, рис. 7.4), який підключений до стовпця з клавішами «9», «6», «3», «-», а потім послідовно опитуються виводи (входи порту D) RD0, RD1, RD2, RD3. Якщо якась клавіша натиснута, здійснюється перехід на підпрограму виведенння значення натиснутої клавіши на цифровий індикатор 0-го розряду (рис. 7.2).

Виведення значення цифри клавіши, що натиснута, здійснюється через виводи RB0, RB1, RB2, RB3 порту В у статичному режимі. 4-розрядний двійковий код з виводів RB0, RB1, RB2, RB3 порту В подається на дешифратор перетворення двійкового коду у сьомисегментний код і далі на нульовий розряд цифрового індикатору.

Далі логічна «1» виводиться на вивід RA1, що підключений до стовпця із клавішами «8», «5», «2», «.», а далі послідовно опитуються виводи RD0, RD1, RD2, RD3 (входи порту D).

Далі логічна «1» виводиться на вивід RA2, що підключений до стовпця із клавішами «7», «4», «1», «0», а далі послідовно опитуються виводи RD0, RD1, RD2, RD3 (входи порту D).

Після чого виконується перехід на початок програми й таким чином клавіатура постійно опитується.

Приклад програми опитування матричної клавіатури наведений на рис. 7.6.

```
Мітка    Мнемокод і операнди          Коментар
         list      p=16f877a          ;вибір типу мікроконтролера
         #include  <p16f877a.inc>     ;підключення бібліотеки програм
mex      equ       h'020'             ;ініціалізація змінних
men      equ       h'021'
temp0    equ       h'022'
temp00   equ       h'023'
         org       0x00               ;вектор скідання
         goto      begin              ;перехід до основної програми на мітку begin
         org       0x20               ;адреса розміщення програми
begin    clrw                         ;очищення акумулятора
         clrf      mex                ;очищення змінних
```

```
        clrf    men
        clrf    temp0
        clrf    temp00
;Налаштування портів
        clrf    PORTB       ;очищення  портів
        clrf    PORTA
        clrf    PORTD
        bsf     STATUS,RP0  ;перехід до банку 1
        movlw   b'00000000' ;налаштування виводів порта A, як виходи
        movwf   TRISA
        movwf   TRISB       ;налаштування виводів порта B, як виходів
        movlw   b'00001111' ;налаштування виводів порта D, як входи
        movwf   TRISD
        movlw   b'00000110' ;виводи порта A дискретні
        movwf   ADCON1
        bcf     STATUS,RP0  ;перехід до банку 0
;Опитування матричної клавіатури
pm0     bsf     PORTA,0     ;встановити 1 на вихід PORTA,0 для опитування
                            ;клавіатури
        btfsc   PORTD,0     ;опитати клавішу 9, якщо натиснута, то перехід до
                            ;підпрограми call m9, якщо ні – перехід до команди
                            ;btfsc PORTD,1
        call    m9          ;перехід до підпрограми виводу цифри 9
        btfsc   PORTD,1
        call    m6
        btfsc   PORTD,2
        call    m3
        btfsc   PORTD,3
        call    mex
        bcf     PORTA,0     ;скинути біт 0 порта A
        bsf     PORTA,1     ;встановити 1 на вихід PORTA,1 для опитування
                            ;клавіатури наступного стовпця
        btfsc   PORTA,0
        call    m8
        btfsc   PORTD,1
        call    m5
        btfsc   PORTD,2
        call    m2
        btfsc   PORTD,3
        call    men
        bcf     PORTA,1     ;скинути біт 1 порта A
        bsf     PORTA,2     ;встановити 1 на вихід PORTA,2 для опитування
                            ;клавіатури наступного стовпця
        btfsc   PORTD,0
        call    m7
        btfsc   PORTD,1
        call    m4
        btfsc   PORTD,2
        call    m1
        btfsc   PORTD,3
        call    m0
        bcf     PORTA,2
        goto    pm0         ;повернутися на початок програми опитування клавіш
```

```
;підпрограми виведення значень  натиснутих клавіш
m0      movlw   d’0’            ;встановити код натиснутої клавіши "0"
        movwf   temp0           ;temp0;привласнити його змінній temp0
        movlw   d’1’            ;встановити ознаку натиснутої клавіши
        movwf   temp00
        bcf     PORTB,0         ;вивести 4-х розрядний код цифри 0 на індикатор
        bcf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return                  ;повернутися до основний програми
m1      movlw   d’1’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bsf     PORTB,0
        bcf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
m2      movlw   d’2’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bcf     PORTB,0
        bsf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
m3      movlw   d’3’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bsf     PORTB,0
        bsf     PORTB,1
        bcf     PORTB,2
        bcf     PORTB,3
        return
m4      movlw   d’4’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bcf     PORTB,0
        bcf     PORTB,1
        bsf     PORTB,2
        bcf     PORTB,3
        return
m5      movlw   d’5’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bsf     PORTB,0
        bcf     PORTB,1
        bsf     PORTB,2
```

```
        bcf     PORTB,3
        return
m6      movlw   d’6’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bcf     PORTB,0
        bsf     PORTB,1
        bsf     PORTB,2
        bcf     PORTB,3
        return
m7      movlw   d’7’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bsf     PORTB,0
        bsf     PORTB,1
        bsf     PORTB,2
        bcf     PORTB,3
        return
m8      movlw   d’8’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bcf     PORTB,0
        bcf     PORTB,1
        bcf     PORTB,2
        bsf     PORTB,3
        return
m9      movlw   d’9’
        movwf   temp0
        movlw   d’1’
        movwf   temp00
        bsf     PORTB,0
        bcf     PORTB,1
        bcf     PORTB,2
        bsf     PORTB,3
        return
        nop
        end
```

Рис. 7.6. Програма опитування матричної клавіатури

8. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ. АНАЛОГО-ЦИФРОВИЙ ПЕРЕТВОРЮВАЧ

Модуль АЦП має 5 каналів у 28 вивідних та 8 каналів у 40/44 вивідних мікросхем.

Вхідний аналоговий сигнал через комутатор заряджає внутрішній конденсатор АЦП CHOLD. Модуль АЦП перетворює напругу, що утримується на конденсаторі у відповідний 10-розрядний цифровий код методом послідовного наближення. Джерело верхньої та нижньої опорної напруги може бути програмно обране з виводів V_{DD}, V_{SS}, RA2 або RA3.

Допускається робота модуля АЦП в режимі SLEEP мікроконтролера, при цьому як джерело тактових імпульсів для АЦП має бути обраний RC генератор.

8.1. Регістри АЦП

Для управління АЦП в мікроконтролері використовуються 4 регістри:

- регістр результату ADRESH (старший байт);
- регістр результату ADRESL (молодший байт);
- регістр управління ADCON0;
- регістр управління ADCON1.

Регістри ADCON0 (табл.8.1) та ADCON1 (табл.8.2) використовуються для налаштування роботи модуля АЦП. Регістр ADCON0 забезпечує вибір тактового генератора АЦП, номера канала, вмикання АЦП в роботу і запускає процес перетворення. За допомогою регістра ADCON1 встановлюється які входи порту А і порту Е мікроконтролера будуть застосовуватися як аналогові, а які як дискретні.

Функціональне призначення бітів регістра ADCON0 наведене у табл. 8.1

Таблиця 8.1

Регістр ADCON0

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO-DONE	-	ADON
Біт 7-6	ADCS1:ADCS0: Вибір джерела тактового сигналу 00 - $F_{OSC}/2$ 01 - $F_{OSC}/8$ 10 - $F_{OSC}/32$ 11 - F_{RC} (внутрішній RC-генератор модуля АЦП)						

Продовження табл. 8.1

Біт 5-3	CHS2:CHS0: Вибір аналогового каналу 000 - канал 0, (RA0/AN0) 001 - канал 1, (RA1/AN) 010 - канал 2, (RA2/AN2) 011 - канал 3, (RA3/AN3) 100 - канал 4, (RA5/AN4) 101 - канал 5, (RE0/AN5) 110 - канал 6, (RE1/AN6) 111 - канал 7, (RE2/AN7)
Біт 2	GO-DONE: Біт статусу модуля АЦП Якщо біт ADON встановлений в 1, то значення біта GO-DONE 1 - модуль АЦП виконує перетворення (встановлення біта викликає початок перетворення); 0 - стан очікування (апаратно скидається по завершенню перетворення).
Біт 1	Не застосовується: читається як «0»
Біт 0	ADON: біт вмикання модуля АЦП 1 - модуль АЦП ввімкнений 0 - модуль АЦП вимкнений і не споживає струм

У регістрах ADRESH:ADRESL зберігається 10-розрядний результат АЦП перетворення. Після завершення перетворення результат перетворення записується в регістри ADRESH:ADRESL, скидається прапор GO/-DONE (ADCON0<2>), встановлюється прапор переривання ADIF. Структурна схема модуля АЦП наведена на рис. 8.1

Таблиця 8.2

Регістр ADCON1

Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

Біт 7	ADFM: Формат збереження 10-розрядного результату 1 - праве вирівнювання, 6 старших біт ADRESH читаються як «0» 0 - ліве вирівнювання, 6 молодших біт ADRESL читаються як «0»
Біти 3-0	PCFG3:PCFG0: Біти управління налаштування каналів АЦП

PCFG3: PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	$V_{REF}+$	$V_{REF}-$	Кан/ V_{REF}
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0
0001	A	A	A	A	$V_{REF}+$	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A		A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	$V_{REF}+$	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D		D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	$V_{REF}+$	D	A	A	RA3	V_{SS}	2/1
011x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0
1000	A	A	A	A	$V_{REF}+$	$V_{REF}-$	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	A	A	$V_{REF}+$	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	$V_{REF}+$	$V_{REF}-$	A	A	RA3	RA2	4/2
1100	D	D	D	A	$V_{REF}+$	$V_{REF}-$	A	A	RA3	RA2	3/2
1101	D	D	D	D	$V_{REF}+$	$V_{REF}-$	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	$V_{REF}+$	$V_{REF}-$	D	A	RA3	RA2	1/2

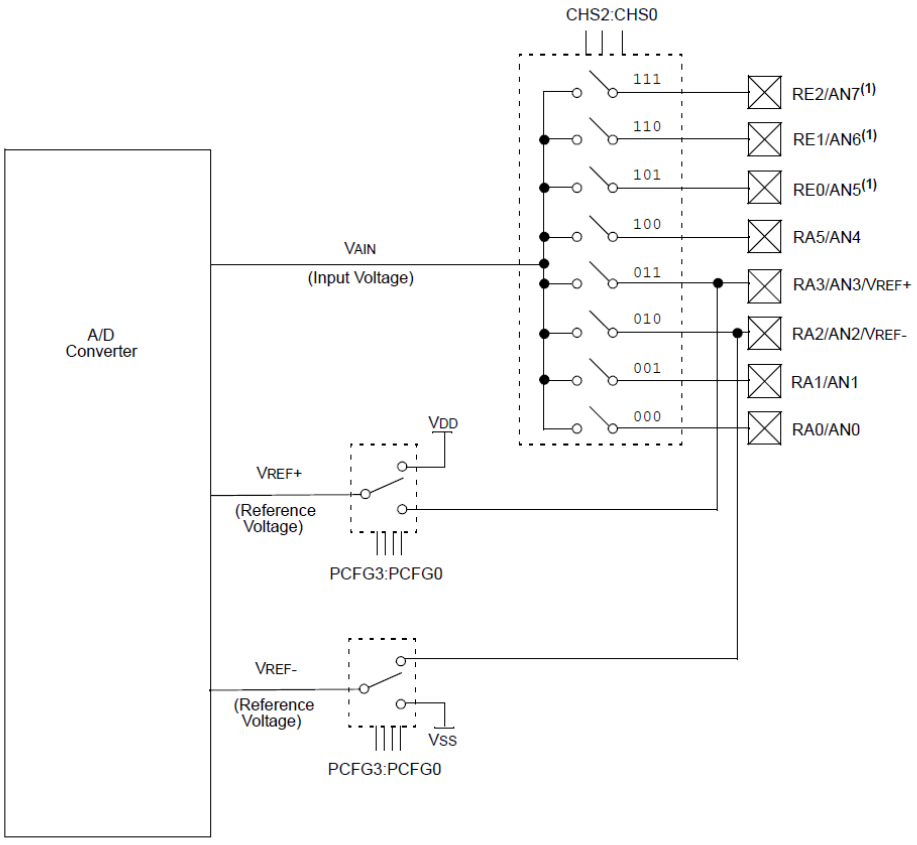


Рис. 8.1. Структурна схема АЦП [4]

8.2. Порядок роботи з АЦП

На початку роботи з АЦП треба обрати тип тактового генератора, робочий аналоговий канал, увімкнути АЦП. Відповідні біти аналогових каналів TRIS треба налаштовувати на вхід. Перед початком процеса перетворення потрібно витримати тимчасову паузу.

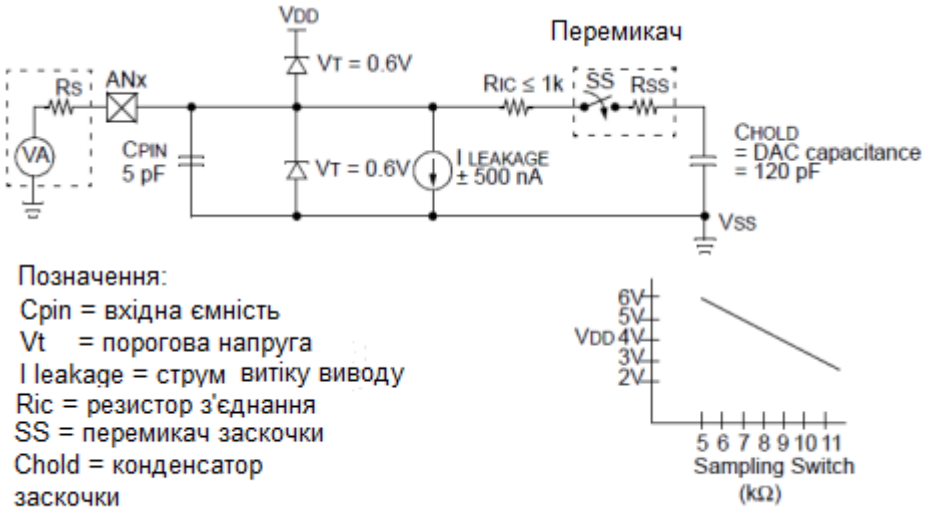


Рис. 8.2. Схема вхідного кола АЦП [4]

Рекомендована послідовність дій для роботи з АЦП:

1) налаштувати модуль АЦП, для чого:

- налаштувати виводи як аналогові входи, входи V_{REF} чи цифрові канали введення/виведення бітами PCFG3: PCFG0 регістра ADCON1;
- обрати вхідний канал АЦП бітами CHS2:CHS0 регістру ADCON0;
- обрати джерело тактових імпульсів для АЦП бітами ADCS1:ADCS0 регістру ADCON0;
- увімкнути модуль АЦП – біт ADON регістру ADCON0);
- налаштувати переривання від модуля АЦП (за необхідності):
    - скинути біт ADIF на «0»;
    - встановити біт ADIE в «1»;
    - встановити біт PEIE в «1»;
    - встановити біт GIE в «1»;

2) витримати паузу, необхідну для заряджання конденсатора C_{HOLD};

3) розпочати аналого-цифрове перетворення, для чого:

- встановити біт GO/-DONE в «1» у регістрі ADCON0;

- чекати на закінчення перетворення, доки біт GO/-DONE не буде скинутий на «0»

або

- очікувати прапора переривання (біт ADIF) після закінчення перетворення;

4) зчитати результат перетворення з регістрів ADRESH: ADRESL, скинути біт ADIF у «0», якщо це необхідно;

Для наступного перетворення необхідно виконати кроки, починаючи з 1 або 2.

Час перетворення одного біта визначається як час T_{AD}, тобто час одного тактового імпульсу тактового генератора АЦП. Мінімальний час очікування перед наступним перетворенням має становити не менше 2 T_{AD} .

8.2.1. Вибір джерела тактових імпульсів для АЦП

Час отримання одного біта результату визначається параметром T_{AD}. Для 10-розрядного результату потрібно щонайменше 12 T_{AD}. Параметри тактового сигналу для АЦП визначаються програмно, T_{AD} може приймати такі значення:

- $2T_{OSC}$;
- $8T_{OSC}$;
- $32T_{OSC}$;
- внутрішній RC генератор модуля АЦП (2-6 мкс).

Для отримання коректного результату перетворення необхідно вибрати джерело тактового сигналу АЦП, що забезпечує час TAD щонайменше 1,6 мкс.

Максимальні значення F_{OSC}, які задовільняють вимогам до T_{AD} (для мікроконтролерів з нормальним діапазоном напруги F живлення) наведені у табл. 8.3.

Таблиця 8.3

Вибір типу тактового генератора АЦП

Вибір T_{AD}		F_{OSC}
Режим	ADCS1:ADCS0	Максимум
$2T_{OSC}$	00	1,25 МГц
$8T_{OSC}$	01	5 МГц
$32T_{OSC}$	10	20 МГц
RC	11	*)

*) 1. Типове значення часу T_{AD} RC- генератора АЦП дорівнює 4 мкс, може змінюватись від 2 мкс до 6 мкс.

2. Якщо тактова частота мікроконтролера більше 1 МГц, рекомендується застосовувати RC-генератор АЦП тільки для роботи в SLEEP режимі.

3. Для мікроконтролерів з розширеним діапазоном напруг живлення (LF) на ці параметри треба дивитись в розділі електричних характеристик.

8.2.2. Налаштування аналогових входів

Регістри ADCON1, TRISA та TRISE відповідають за налаштування виводів АЦП. Якщо виводи мікросхеми налаштовуються як аналогові входи, то повинні бути встановлені в «1» відповідні біти в регістрі TRIS. Якщо відповідний біт скинутий у «0», вивод мікросхеми налаштований як цифровий вихід зі значеннями вихідної напруги V_{OH} і V_{OL} .

Модуль АЦП функціонує незалежно від стану бітів CHS2:CHS0 і бітів TRIS.

8.2.3. Аналого-цифрове перетворення

Скидання біта GO/-DONE в «0» під час перетворення призведе до його припинення. При цьому регістри результату (ADRESH:ADRESL) не змінять свого вмісту. Після завершення перетворення необхідно забезпечити тимчасову затримку $2T_{AD}$. Витримавши необхідну паузу, можна розпочати нове перетворення установкою біта GO/-DONE в «1».

На рис. 8.3. наведене послідовність отримання результату після встановлення біта GO/-DONE в «1».

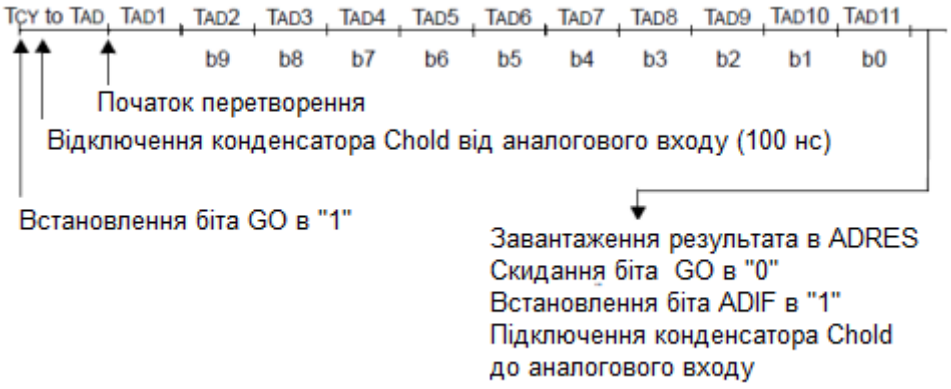


Рис. 8.3. Схема процесу перетворення в АЦП [4]

8.2.4. Вирівнювання результату перетворення

10-розрядний результат перетворення зберігається в спареному 16-розрядному регістрі ADRESH: ADRESL. Запис результату перетворення може виконуватися з правим або лівим вирівнюванням, залежно від біта ADFM. Не задіяні біти регістра ADRESH: ADRESL читаються як «0». Якщо модуль АЦП вимкнено, то 8-розрядні регістри ADRESH і ADRESL можуть використовуватися як регістри загального призначення.

Рис. 8.4. Схема вирівнювання результату перетворення [4]

Таблиця 8.4

Регістри і біти, які пов'язані з роботою АЦП

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh 8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	Старший байт результату перетворення								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	Молодший байт результату перетворення								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/-DONE	-	ADON	0000 00 - 0	0000 00 - 0
9Fh	ADCON2	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	0 - - - 0000	0 - - - 0000
85h	TRISA			Регістр напрямку даних PORTA						--11 1111	--11 1111
05h	PORTA			Регістр заскочки PORTA						--0x 0000	-- 0u 0000
89h	TRISE	IBF	OBF	IBOV	PSPM	-	Регістр напр. PORTE			0000 -111	0000 -111
09h	PORTE	-	-	-	-	-	RE2	RE1	RE0	---- -xxx	---- -uuu

Позначення: - – не застосовується, читається як 0; u – не змінюється; x – не відомо; q– залежить від умов.

Приклад програми АЦП наведений на рис. 8.5.

```
Мітка     Мнемокод і операнди          Коментар
          list      p=16f877a          ;вибір типу мікроконтролера
          #include  <p16f877a.inc>     ;підключення бібліотеки програм
tempah    equ       h'020'
tempal    equ       h'021'
tempb     equ       h'022'
          org       0x00               ;вектор скідання
          goto      begin              ;перехід до основної програми на мітку begin
          org       0x20               ;адреса розміщення програми
begin     clrw      tempah             ;очищення акумулятора
          clrf      tempal
          clrf      tempb
;Налаштування АЦП
          clrf      PORTA
          bsf       STATUS,RP0         ;перехід до банку 1
          movlw     b'00111111'        ;налаштування виводів порта А, як виходів
          movwf     TRISA
          movlw     b'10000000'        ;виводи порта А аналогові, праве вирівнювання
          movwf     ADCON1
          bcf       STATUS,RP0         ;перехід до банку 0
          movlw     b'11100001'        ;RC-генератор, канал AN4, АЦП ввімкнений
          movwf     ADCON0
m1        clrf      tempb              ;затримка часу для зарядження контенсатора
m3        incf      tempb
          btfss     tempb,3
          goto      m3
          вsf       ADCON0,2           ;початок перетворення
m2        btfsc     ADCON0,2           ;очикування завершення перетворення
          goto      m2
          movfw     ADRESH             ;зчитування старшого байта результату перетворення
          movwf     tempah             ; в акумулятор і потім в tempah
          bsf       STATUS,RP0         ;зчитування молодшего байта результата
          movfw     ADRESL             ;перетворення в акумулятор і потім в tempal
          bcf       STATUS,RP0
          movwf     tempal
          goto      m1
          nop
          end
```

Рис. 8.5. Приклад програми АЦП

8.3. Програма позіційного регулятора температури

Схема реалізації двопозіційного регулятора із застосуванням мікроконтролера наведена на рис. 8.6. Об’єкт регулювання складається з нагрівального елементу, який підключений до симистора, та хромель-копелевої термопари. Вхід управління симистором підключений до виходу оптопари, яка виконує гальванічне розв’язання між дискретним виходом RB4 мікроконтролера та входом симистора. Вихід термопари через підсилювач подається на аналоговий вхід мікроконтролера RA5 (аналоговий канал AN4).

Робота позіційного регулятора полягає у вимірюванні та перетворенні аналогового сигналу на вході RA5 в АЦП, порівнянні вимірюваного сигналу з термопари із заданим значенням температури. Задане значення температури, як і результат перетворення в АЦП, складається з двох байтів, тому порівняння треба виконувати спочатку для старших байтів, а якщо вони збігаються, то – для молодших. Якщо поточне значення менше за задане, формується дискретний сигнал на виході RB4 на увімкнення нагрівального елементу. У протиленжному випадку формується сигнал на вимкнення нагрівального елементу.

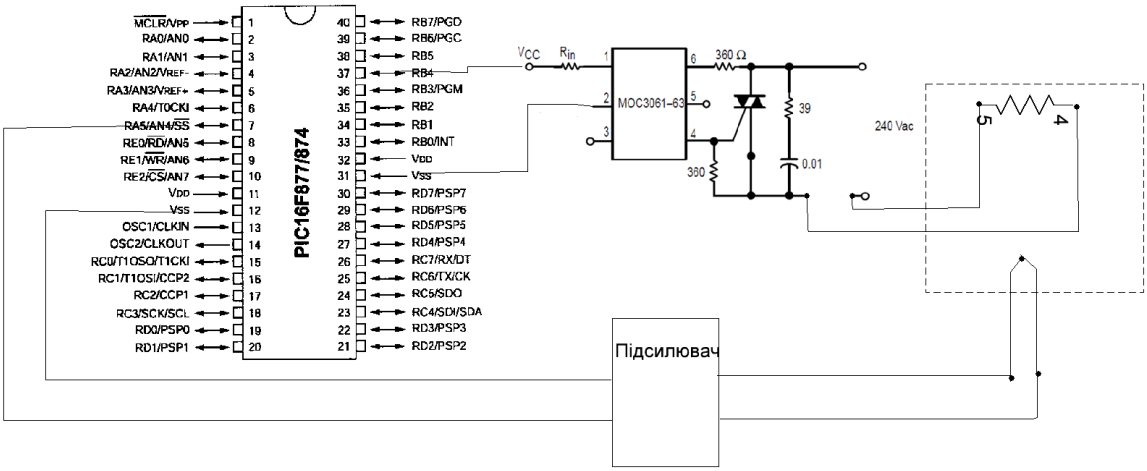


Рис. 8.6. Схема підключення позіційного регулятора температури

Приклад програми регулювання температури наведений на рис. 8.7.

Мітка	Мнемокод і операнди		Коментар
	list	p=16f877a	;вибір типу мікроконтролера
	#include	<p16f877a.inc>	;підключення бібліотеки програм
temph	equ	h’020’	;ініціалізація змінних
templ	equ	h’021’	
tempb	equ	h’022’	

```
tempzh    equ     h’023’
tempzl    equ     h’024’
          org     0x00              ;вектор скідання
          goto    begin             ;перехід до основної програми на мітку begin
          org     0x20              ;адреса розміщення програми
begin     clrw    tempah            ;очищення акумулятора
          clrf    tempal            ;очищення змінних
          clrf    tempb
          clrf    tempzh
          clrf    tempzl
;Налаштування АЦП
          clrf    PORTA
          clrf    PORTB
          movlw   b’00000010’       ;Привласнення значення заданій величині, старший
                                    ;байт
          movwf   tempzh
          movlw   b’00000000’       ;Привласнення значення заданій величині, молодший
                                    ;байт
          movwf   tempzl
          bsf     STATUS,RP0        ;перехід до банку 1
          movlw   b’00100000’       ;налаштування вивода RA5 порта A  як вхід
          movwf   TRISA
          clrf    TRISB
          movlw   b’10000000’       ;виводи порта A аналогові, праве вирівнювання
          movwf   ADCON1
          bcf     STATUS,RP0        ;перехід до банку 0
          movlw   B’11100001’       ;RC-генератор,  вивід RA5,  канал AN4, АЦП
                                    ввімкнений
          movwf   ADCON0
m7        clrf    tempb
m1        incf    tempb             ;часова затримка
          btfss   tempb,3
          goto    m1
          bsf     ADCON0,2          ;початок перетворення в АЦП
m2        btfsc   ADCON0,2          ;перевірка кінця перетворення в АЦП
          goto    m2
          movfw   ADRESH            ;зберегти результат перетворення в змінних temph,
                                    templ
          movwf   temph
          bsf     STATUS,RP0
          movfw   ADRESL
          bcf     STATUS,RP0
          movwf   templ
; порівняння сигналів поточного і заданого значення величини, яка регулюється, та
;формування сигнала на ввімкнення і вимкнення нагрівального елемента
          movfw   temph
          subwf   tempzh,0          ;порівняння старших байтів заданого значення і
                                    ;поточного значення перетворення в АЦП
          btfsc   STATUS,Z          ;якщо старші байти дорівнюють один одному,
          goto    m6                ;то перехід до порівняння молодших байтів
          btfsc   STATUS,C          ;якщо ні, то перевірка, який більше
          goto    m3                ;поточне значення менше заданого, то перехід на m3
          goto    m4                ;поточне значення більше заданого, то перехід на m4
m6        movfw   templ
          subwf   tempzl,0          ;порівняння молодших байтів
```

```
        btfsc   STATUS,C
        goto    m3              ;поточне значення менше заданого, то перехід на m3
        goto    m4              ;поточне значення більше заданого, то перехід на m4
m3      bsf     PORTB,4         ;ввімкнути нагрівальний пристрий
        goto    m5
m4      bcf     PORTB,4         ;вимкнути нагрівальний пристрий
m5      goto    m7              ;повернення на початок перетворення
        nop
        end
```

Рис. 8.7. Приклад програми регулювання температури

Контрольні запитання та завдання

1. Для чого в мікроконтролерах застосовують аналого-цифрове перетворення?
2. Якою є розрядність АЦП мікроконтролера?
3. Якою є максимальна розрядність двійкового коду АЦ-перетворення?
4. Яким чином можна визначити точність вимірювання вхідного сигналу АЦП?
5. Які регістри застосовуються в АЦП?
6. В яких регістрах зберігається результат АЦ-перетвотрення?
7. Для чого застосовують вирівнювання результату АЦ-перетворення.
8. Які параметри АЦП налаштовуються бітами регістра ADCON0.
9. Чому виникає потреба у затримці часу перед початком АЦ-перетворення?
10. Які параметри АЦП налаштовуються бітами регістра ADCON1?

9. ЗОВНІШНІ ІНТЕРФЕЙСИ МІКРО-ЕОМ

9.1 Послідовні інтерфейси та протоколи передачі

Для опису сукупності засобів схемотехніки та функцій, що забезпечують безпосередню взаємодію складових елементів систем обробки даних (СОД), мереж, систем передачі даних (СПД), підсистем периферійного устаткування, використовуються поняття «інтерфейс», «стик», «протокол» [7].

Під стандартним інтерфейсом розуміється сукупність уніфікованих апаратних, програмних та конструктивних засобів, необхідних для реалізації взаємодії різних функціональних елементів в автоматичних системах збирання та обробки інформації за умов, що визначені стандартом і спрямовані на забезпечення інформаційної, електричної та конструктивної сумісності зазначених елементів.

Стиком називається місце з'єднання пристроїв передачі сигналів та даних, що входять до системи передачі даних, опис функцій та засобів сполучення елементів засобів зв'язку та СПД.

Під протоколом розуміється строго задана процедура або сукупність правил, що регламентує спосіб виконання певного класу функцій.

Основне призначення інтерфейсів, стиків та протоколів – уніфікація внутрішньосистемних та міжсистемних, внутрішньомережних та міжмережних зв'язків з метою ефективної реалізації прогресивних методів проектування функціональних елементів обчислювальних систем та мереж.

Основними функціями інтерфейсів та стиків є забезпечення інформаційної, електричної та конструктивної сумісності.

Інформаційна сумісність – узгодженість взаємодії функціональних елементів відповідно до сукупності логічних умов.

Логічні умови визначають:

- структуру і склад уніфікованого набору шин;
- набір процедур з реалізації взаємодії та послідовність їх виконання для різних режимів функціонування;
- спосіб кодування та формати команд, даних, адресної інформації та інформації стану;
- часові співвідношення між сигналами, що управляють, обмеження на їх форму та взаємодію.

Логічні умови інформаційної сумісності визначають в цілому функціональну та структурну організацію інтерфейсу. Умови інформаційної сумісності впливають на обсяг та складність устаткування схемотехніки і програмного забезпеченні, а також на основні техніко-економічні показники інтерфейсу.

Електрична сумісність – узгодженість статичних та динамічних параметрів електричних сигналів в системі шин з урахуванням обмежень на просторове розміщення обладнання інтерфейсу та технічну реалізацію приймально-передавальних елементів.

Умови електричної сумісності визначають:

- тип приймально-передавальних елементів;
- співвідношення між логічними та електричними станами сигналів і межі їх зміни;
- коефіцієнти здатності навантаження приймально-передавальних елементів та значення допустимого ємнісного і резистивного навантаження в пристрої;
- схему підключення лінії до роз'ємів;
- вимоги до джерел та кіл електричного живлення;
- вимоги до завадостійкості.

Більшість умов електричної сумісності та типи приймально-передавальних елементів зазвичай регламентуються стандартом.

Умови електричної сумісності впливають на швидкість обміну даними, гранично допустиме число пристроїв, що підключаються, їх конфігурацію та відстань між ними, а також завадозахищеність.

Конструктивна сумісність – узгодженість конструктивних елементів інтерфейсу, призначених для забезпечення механічного контакту електричних з'єднань та механічної заміни схемних елементів, блоків та пристроїв.

Умови конструктивної сумісності визначають:

- типи сполучних елементів (роз'єм, штекер) та розподіл ліній зв'язку усередині сполучного елементу;
- конструкції плат, каркасів, стойок;
- конструкції кабельного з'єднання.

Умови конструктивної сумісності в рекомендаціях стандартних інтерфейсів не завжди встановлені повністю, а в деяких можуть бути відсутніми або визначати декілька варіантів використання.

9.2. Кола та режими передачі даних

Каналом називається середовище для поширення сигналів. Найбільш простий канал являє собою два відрізки проводу, які можуть забезпечити передачу даних на коротку відстань. Складніші канали можуть бути реалізовані з використанням коаксіального кабелю для обчислювальних мереж, наприклад, Ethernet або кабелю «кручена пара» (екранованого або неекранованого). Кабель кручена пара наразі є основним засобом для реалізації обчислювальних мереж.

Електричні кола, що реалізовують обчислювальні мережі, поділяються на несиметричні та симетричні.

Несиметричне коло – електричне коло, один з провідників якого має потенціал «земля». «Землею» називається загальна точка відліку в схемі, відносно якої вимірюють усі потенціали та напругу.

Симетричне коло – електричне коло, жоден з провідників якого не використовується як «земля». Сигнали у симетричних колах вимірюються як різницю напруги на 2-х провідниках.

Режими передачі даних.

Розрізняють три режими передачі даних:

- дуплексний;
- напівдуплексний;
- симплексний.

Дуплексний режим – це такий режим передачі даних, за яким обидва боки каналу можуть приймати та передавати дані одночасно.

Напівдуплексний режим – це такий режим передачі даних, за яким з кожного боку каналу можна або передавати або приймати дані, але не одночасно.

Симплексний режим – це такий режим передачі даних, за яким з одного боку каналу можна тільки передавати, а з іншого – тільки приймати інформацію.

9.3. Інтерфейс RS-232

9.3.1. Загальні поняття

RS-232 (*Recommended Standard 232*) – стандарт, що описує інтерфейс для послідовної двоспрямованої передачі даних між терміналом (DTE, *Data*

Terminal Equipment) та кінцевим пристроєм (DCE, *Data Circuit - Terminating Equipment*).

Інтерфейс RS-232C був застосований в перших персональних комп'ютерах фірми IBM і до сьогоднішнього дня входить до структури будь-якого персонального комп'ютера в апаратному або програмному вигляді [8, 9].

Інтерфейс RS-232 реалізований повністю апаратно на персональних комп'ютерах (PC) у вигляді мікросхем та роз'ємів. У PC його називають COM-портом (*Communication port*).

Апаратна реалізація означає те, що він працює завжди, незалежно від того, яка операційна система (ОС) встановлена на PC (він працює й без ОС). Програми можуть взаємодіяти з COM-портами усіма доступними засобами: прямим кодом мікропроцесора, апаратними перериваннями, функціями BIOS, засобами ОС, компонентами мов високого рівня.

Сом-порт, реалізований за стандартом RS-232, є універсальним. Він забезпечує роботу PC з периферійними пристроями (чим зараз зайнятий USB), взаємодію з локальною мережею через модем (Ethernet), обмін даними між PC та промисловим устаткуванням (ModBus та ін.) [9].

9.3.2. Формат даних інтерфейсу RS-232

У системах, що містять інтерфейс RS-232, дані передаються асинхронно, тобто у вигляді послідовності пакетів даних. Асинхронна послідовна передача даних є дешевим і простим способом передання даних між ЕОМ та повільними пристроями (наприклад, терміналами) кодом ASCII. Зазвичай швидкість передачі становить 9600 бод. Наразі вона сягає 115200 бод. Швидкість у бодах - це кількість часових інтервалів на секунду, що асоціюються з послідовною передачею. Сигнал в кожному інтервалі має значення «1» або «0», отже швидкість у бодах відповідає швидкості передачі даних у бітах на секунду [3].

Передавання інформації в інтерфейсі RS-232 здійснюється порціями (символами). Символ складається з таких чотирьох частин:

1) стартовий біт;

2) від п'яти до восьми біт власне даних;

3) необов'язковий біт парного або непарного паритету;

4) один, 1,5 або два стопових біти.

На рис. 9.1 наведена часова діаграма передачі символу в 11-бітовому форматі зі швидкістю 110 бод. Наприкінці кожного символу сигнал має значення «1» для завдання стопового біта. Він залишається у стані «1» аж до початку наступного символу, який розпочинається стартовим бітом зі значенням «0».

Стани логічних сигналів «1» та «0» називаються також маркером (*marking* – лінія відмічена, якщо рівень сигналу високий) та пропуском (*spacing* – лінія порожня, якщо рівень сигналу низький). Перехід від «1» до «0» на початку стартового біта надає можливість приймачу розпізнати появу вхідного символу. Отже, між послідовними символами допускаються проміжки будь-якої довжини.

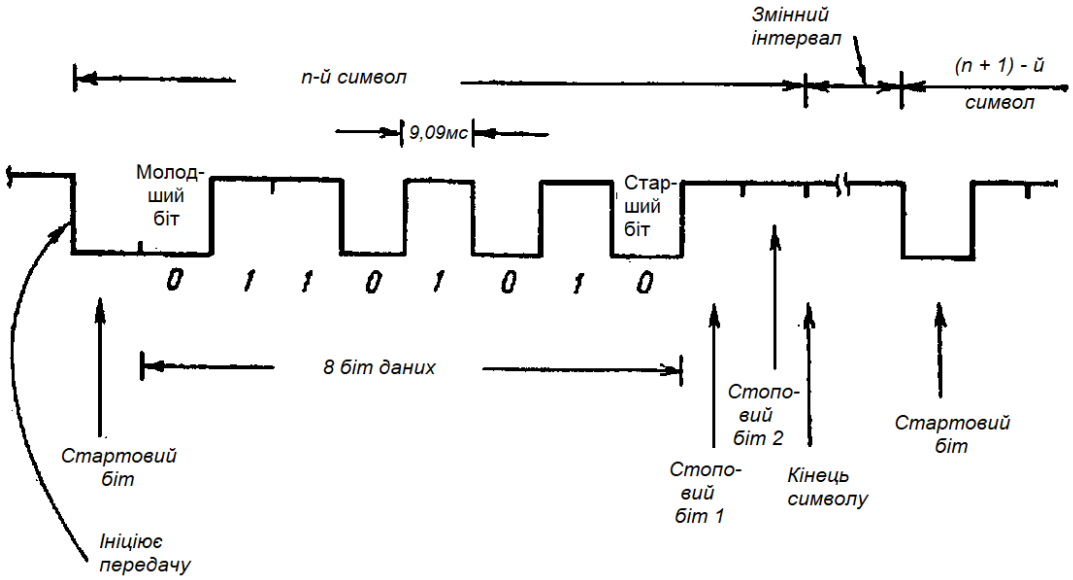


Рис. 9.1. Типовий 11-бітовий формат даних для асинхронної послідовної передачі [3]

Перевагою такої передачі є відсутність потреби у лінії спільної синхронізації. Для зв'язку обчислювального пристрою із системою достатньо двох сигнальних ліній. Замість спільної синхронізації наявні два незалежні генератори синхронізації: один – у пристрої-передавчі, інший – у пристрої-приймачі. Частота їх роботи набагато вище за швидкість передачі у бодах (зазвичай у 16 разів). Використовується стартовий біт, генератор синхронізації в приймачі може синхронізуватися за початком кожного символу, що компенсує можливу відмінність частот генераторів.

Дані передаються системною шиною паралельно. Для підключення до ЕОМ асинхронного послідовного обладнання введення інтерфейс повинен виконувати такі основні функції:

- розпізнавання стартового та стопового бітів;

- послідовно-паралельне перетворення бітів даних символу, що надходить;

- виявлення помилок.

Для асинхронного послідовного виведення інтерфейс повинен перетворювати дані з паралельної форми в послідовну та вводити біт паритету, стартовий та стоповий біти. Це функції, які раніше реалізовувалися на мікросхемах з малою мірою інтеграції, тепер виконуються великою інтегральною схемою (ВІС) універсального асинхронного приймача/передавача (УАПП, УСАПП, USART). Вона містить логічні елементи для прийому та передачі асинхронних послідовних даних.

В УАПП передбачене програмне управління форматом даних. На рис. 9.2 наведена блок-схема УАПП.

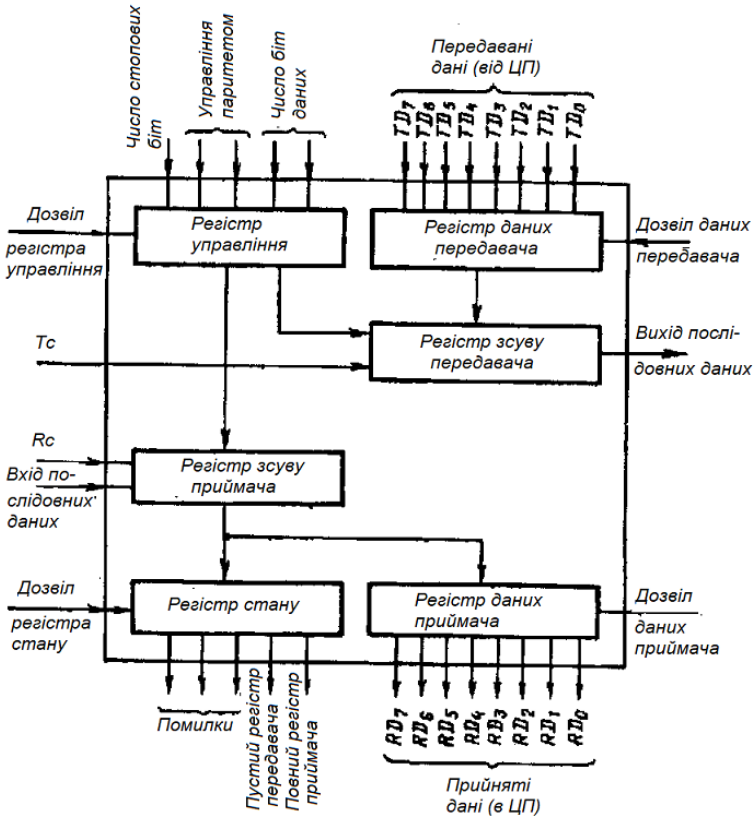


Рис. 9.2. Блок-схема універсального асинхронного приймача [3]

9.3.3. Операція введення даних

В операції введення УАПП спочатку чекає переходу від «1» до «0» на сигнальній лінії послідовного входу. Коли він виявляється, вхідна лінія опитується наприкінці восьмого імпульсу синхронізації приймача (Rc). На

рис. 9.3 наведена схема прийому даних під час асинхронної передачі. Прийом даних полягає у зчитуванні стану бітів двійкового числа, яке передається.

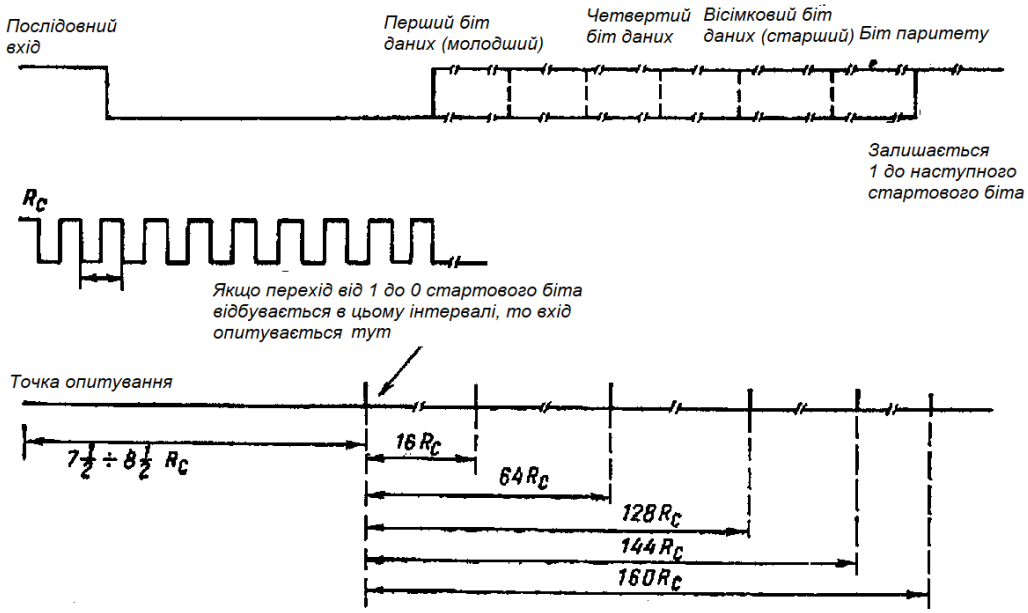


Рис. 9.3. Прийом даних під час асинхронної передачі [3]

Як випливає з рис. 9.3, точний момент опитування стану стартового біта варіюється від 7,5 до 8,5 періодів синхронізації з моменту появи стартового біта. Після розпізнавання правильного стартового біта вхідна лінія опитується через 16 імпульсів і стани бітів двійкового числа, яке передається, спрямовуються до регістру зсуву (першим надходить молодший біт даних). Оскільки частота Rc у 16 разів більше за швидкість у бодах, момент опитування є близьким до середини біта вхідного символу.

Потрібно мати окремий регістр даних приймача, оскільки після прийому символу регістр зсуву треба звільнити для прийому наступного символу. Отриманий символ передається з регістра зсуву в регістр даних приймача до надходження наступного символу, інакше в регістрі стану встановлюється біт перевантаження. Допустимий час на введення залежить від швидкості передачі та кількості бітів у символі.

Біт паритету та стоповий біт опитуються як і біти даних, але ці біти не передаються до регістру зсуву. Якщо прийнятий біт паритету не відповідає біту паритету, що сформований в УАПП за результатом прийому бітів даних, в регістрі стану фіксується помилка паритету.

Стоповий біт використовується для виявлення помилки кадру. Вона виникає, якщо частота синхронізації приймача не в 16 разів більше

швидкості передачі. В результаті при прийомі бітів даних накопичується зсув точки опитування від середини бітового інтервалу. Якщо відмінність частот в є невеликою, то символ приймається правильно і Rc синхронізуватиметься стартовим бітом наступного символу. Якщо ж відмінність частоти Rc перевищує припустимий поріг, точка опитування виходить за межі бітового інтервалу, що призводить до помилки у сприйнятті чергового біта. Цю помилку можна виявити за стоповим бітом, якщо у стоповому біті замість «1» фіксується «0».

9.3.4. Операція виведення даних

В операції виведення символ з процесора передається до регістру даних передавача, а потім до регістра зсуву передавача. Під час передавання даних символи передаються з регістра зсуву передавача послідовно (першим прийшов – першим вийшов) за тактовими сигналами синхронізації генератора (T_C). Символи можуть мати розмір від 4 до 8 біт. Після передачі попереднього символу формується стартовий біт, а потім через 16 імпульсів синхронізації передавача (T_C) передаються біти даних.

Біти даних передаються наступним чином: спочатку молодший біт, потім послідовно інші біти і наприкінці – старший біт. Після бітів даних передається сформований в УАПП біт паритету, що призначений для виявлення помилки передачі даних, та стоповий біт. Зазначимо, що при однаковій швидкості введення та виведення Rc та Tc можна підключити до одного генератора синхронізації.

Послідовний вхід та вихід УАПП представляють ТТЛ сигнали, які можна передавати тільки на коротку відстань. Тому потрібна схема перетворення їх у більш придатний для передачі тип сигналів. Застосовуються два стандарти на передачу даних на великі відстані: EIA RS-232C та струмова петля в 20 мА.

Іетерфейс RS-232C УАПП застосовує рівні сигналів -12В....+12В . Зоні нечутливості, тобто відсутності сигналів, відповідає напруга -3В...+3В. При цьому дані, які приймаються та передаються, інвертовані (рис. 9.4).

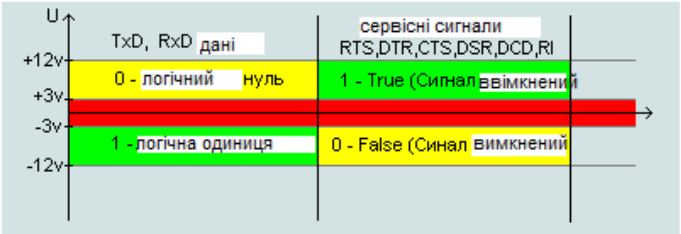


Рис. 9.4. Рівні сигналів УАПП згідно стандарту RS-232C [8]

За стандартом RS-232C сигнал з рівнем більше +3В вважається логічним «0», а з рівнем менше -3В – логічною «1». Стандарт встановлює, що формувач передавача повинен видавати сигнал від -3В до -12В для логічної «1» та від +3В до +12В – для логічного «0».

Вихідні стани:

- порт не ініціалізований – на всіх лініях напруга знаходиться в діапазоні -3В...+3В;
- режим очікування – на всіх лініях напруга знаходиться в діапазоні -3В...-12В.

Діаграма передачі символів згідно інтерфейсу RS-232C наведена на рис. 9.5. Передаються символи «0», яким відповідає ASCII- код – 30h.

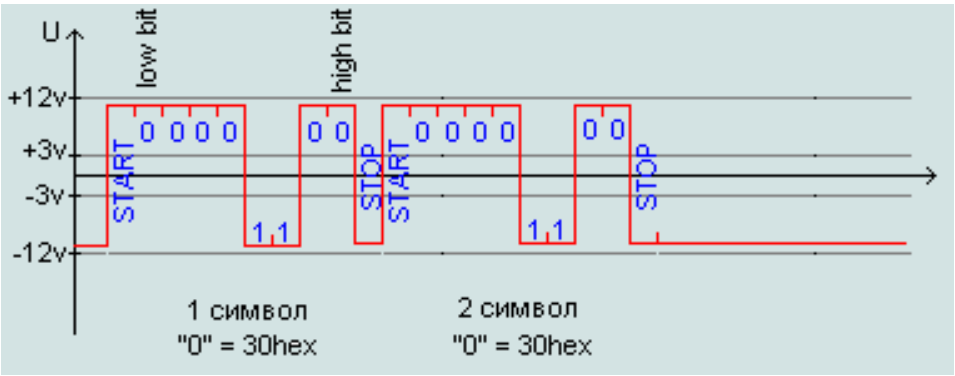


Рис. 9.5. Передача символів «00» без паритету, з одним стоповим бітом [8]

Другий стандарт спирається на струмові сигнали: наявність струму вважається логічною «1», а відсутність – логічним «0». Він розрахований на електромеханічні телетайпи. Для електромагнітів достатній струм 20 мА. У більшості інтерфейсів з цим стандартом термінал є пасивним пристроєм, а інтерфейс – активним, що формує струм для передавача і приймача.

На рис. 9.6 наведені схеми для обох стандартів.

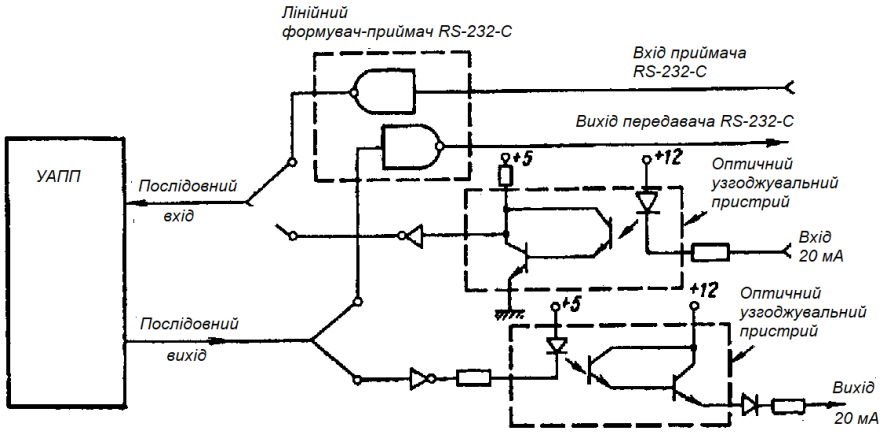


Рис. 9.6 Інтерфейс УАПП та терміналу [3]

Перетворення між RS-232C та ТТЛ вимагає однієї пари формувачів, наприклад, MC1488 та MC1489 (фірми Motorola). Лінійний формувач перетворює ТТЛ-вихід УАПП у сигнал RS-232C, а приймач перетворює послідовний вхід RS-232C від терміналу у ТТЛ-сигнал. Перетворення струмових та ТТЛ-сигналів можна виконати за допомогою оптичних узгоджуючих пристроїв. Вхідний сигнал включає світловипромінюючий діод, який сполучається із фототранзистором, що генерує необхідні рівні напруги.

9.3.5. Реалізація інтерфейсу

Реалізація цього інтерфейсу виконана на мікросхемі UART 8250 INTEL [9]. Операційна система підтримує два порти комунікації, тому використовується дві мікросхеми. Базові адреси знаходяться в елементах пам'яті 0040: 0000 для COM1 та 0040: 0002 для COM2. Базова адреса – це 2-х байтова адреса порту, який є молодшим з групи адрес портів, що мають доступ до UART.

Базові адреси:

COM1 - 3F8H або 3FxH;

COM2 - 2F8H або 2FxH.

Простір введення/виведення. Для операцій з програмно-доступними регістрами простору введення/виведення в PC/AT відводиться тільки 10 молодших біт адреси. Кожному регістру пристрою відводиться своя адреса. До пристроїв, що мають свою адресу, відносяться наступні пристрої: ПДП, переривання, таймер, клавіатура, дисководи, LPT, COM1, COM2, відеоконтроллер тощо.

USART 8250 має 10 програмованих однобайтових регістрів, за допомогою яких управляється та контролюється порт комунікації. Доступ до регістрів здійснюється через адреси 7-ми портів 3F8H - 3FEH (2F8H -2FEH). Регістр з адресою 3FBH є регістром управління лінії і його біт 7 визначає роботу регістрів. Регістри приймача-передавача та їх адреси наведенім у табл. 9.1.

З 10 регістрів для простого послідовного зв'язку необхідно використати 6 регістрів:

- регістр зберігання передавача – байт даних, які будуть надіслані;
- регістр даних приймача – останній отриманий байт даних;
- регістр управління лінією;
- регістр статусу лінії – ці два регістри ініціалізують та управляють лінією зв'язку, використовуючи швидкість обміну, що міститься у двох регістрах дільника швидкості обміну;
- регістр дільника швидкості обміну (молодший);
- регістр дільника швидкості обміну (старший).

Таблиця 9.1

Адреси регістрів приймача USART 8250

Адреса регістра	Призначення регістра
3F8H(OUT, біт 7 =0 в 3FBH)	Регістр зберігання передавача
3F8H(IN, біт 7 =0 в 3FBH)	Регістр даних приймача
3F8H(OUT, біт 7 =1 в 3FBH)	Дільник швидкості обміну(молодший)
3F9H(IN, біт 7 =1 в 3FBH)	Дільник швидкості обміну(старший)
3F9H(OUT, біт 7 =0 в 3FBH)	Регістр дозволу переривань
3FAH(IN)	Регістр ідентифікації переривання
3FBH(OUT)	Регістр управління лінії
3FCH(OUT)	Регістр управління модемом
3FDH(IN)	Регістр статусу лінії
3FEH(IN)	Регістр статусу модему

Для роботи з портом необхідно виконати наступні дії:

1) ініціалізація послідовного порту **(**чи відкриття порту);

2) встановлення поточного комунікаційного порту;

3) визначення статусу комунікаційного порту.

Ініціалізація послідовного порту (або відкриття порту)

При цьому встановлюються усі його параметри, які включають: довжину слова, кількість стоп-бітів, установку парності та швидкість обміну.

Довжина слова – кількість біт, яке утворює основну одиницю даних. Це може бути - 8 біт, іноді достатньо 7 біт (ASCII-коди), для чисел - 4 біти.

Для цього необхідно використати 4 регістри мікросхеми 8250. Це регістри дільника швидкості обміну, регістр контролю лінії та регістр дозволу переривання.

Ініціалізація швидкості обміну. Дільник швидкості обміну – це число, на яке необхідно поділити частоту системного годинника (1843200Гц), щоб отримати бажану швидкість обміну. Наприклад, для швидкості обміну 1200 бод дільник швидкості обміну має дорівнювати 96, оскільки 1843200/(96×16) дорівнює 1200. Чим більше дільник, тим менше швидкість обміну. Швидкість обміну 300 бод і менше вимагає двобайтового числа для дільника. Старший байт посилається до 3F9H (або 2F9H), а молодший – до 3F8H (або 2F8H). В обох випадках біт 7 регістра управління лінією має бути встановлений у «1» перед засиланням значень, інакше по цих двох адресах значення будуть адресовані до інших регістрів (табл. 9.2).

Таблиця 9.2

Значення дільника швидкості регістрів дільника швидкості

Швидкість обміну	Значення дільника швидкості обміну в регістрах	
	Регістр 3F9H	Регістр 3F8H
110	04H	17H
300	01H	80H
600	00H	C0H
1200	00H	60H
1800	00H	40H
2400	00H	30H
3600	00H	20H
4800	00H	18H
9600	00H	0CH

Спочатку завжди встановлюються регістри швидкості обміну, оскільки вони єдині, які вимагають установки біта 7 до регістру контролю лінії. Після цього потрібно змінити вміст регістра контролю лінії, скидаючи біт 7, щоб усі інші доступи до регістрів були правильними.

Оскільки регістр контролю лінії є регістром тільки для запису, то немає способу повернути біт 7 назад до «1» без одночасної установки усіх інших бітів цього регістра.

Ініціалізація регістра контролю лінії. Можливі значення бітів регістра контролю лінії, адреса порту якого дорівнює 3FBH (або 2FBH), наведені у табл. 9.3.

Зазвичай біти 5-7 скинуті до «0». Інші описують значення, що визначені протоколом.

Регістр дозволу переривання. Навіть якщо переривання не використовуються, все одно необхідно зробити запис до регістру дозволу переривання, щоб бути упевненим, що переривання заборонені. Запишіть до цього регістру «0». Регістр ідентифікації переривання можна ігнорувати.

Таблиця 9.3

Значення бітів регістра контролю лінії

Біти	Призначення
1-0	довжина символу 00 – 5 біт, 01 - 6 біт, 10 - 7 біт, 11 -8 біт.
2	кількість стоп-бітів. 0 – 1, 1 – 1,5, якщо довжина символів рівна п'яти, інакше - 2.
3	парність. 1 – генерується біт парності 0 – ні.
4	тип парності. 0 – непарна 1 – парна.
5	фіксація парності. Примушує біт парності завжди бути «0» або «1». 0 – скасована. 1 – завжди 1, якщо біт 3 рівний 1 і біт 4 рівний 0. чи 1 завжди 0, якщо біт 3 рівний 1 і біт 4 рівний 1. чи 1 немає парності, якщо біт 3 рівний 0.
6	установка перерви. Викликає виведення рядка нулів в якості сигналу віддаленої станції. 0 – заборонено 1 – перерва.
7	змінює адреси портів інших регістрів

Встановлення поточного комунікаційного порту.

Існує два способи, за допомогою яких програма може визначити, який з комунікаційних портів повинен використовуватися. Перший з них полягає у

вказівці номера каналу в операторові програми. Другий спосіб полягає в написанні програми для обміну через порт COM1, але зміна комунікаційного адаптера, доступ до якого йде через COM1.

Область даних BIOS містить місце для чотирьох двобайтових змінних, в яких знаходяться базові адреси комунікаційних каналів (MS-DOS підтримує тільки перші два з них). Базова адреса порту – це молодша адреса з групи адрес портів, через які можна отримати доступ до цього комунікаційного каналу. Базова адреса для COM1 зберігається в осередку 0040: 0000, а для COM2 – в осередку 0040: 0002.

Для зміни комунікаційних портів потрібно просто змінити ці два значення. Повторна зміна значень призведе до первинного призначення портів.

Визначення статусу комунікаційного порту.

Регістр статусу лінії мікросхеми 8250 визначає протокол зв'язку. Цей регістр має адресу порту на 5 більше, ніж базова адреса цього каналу. Зазвичай він постійно видимий в процесі комунікаційного обміну. При передачі даних регістр повідомляє, що попередній символ вже посланий, дозволяючи програмі записати новий символ зверху нього. Під час приймання даних регістр інформує програму про надходження наступного символу для того, щоб програма могла прочитати його перш, ніж він буде знищений наступним символом, що прибув. Значення бітів цього регістра визначені у табл. 9.4.

Таблиця 9.4

Значення бітів регістра статусу лінії

Біти	Призначення
0	1 – байт даних отриманий
1	1 – отримані дані перезаписані (попередній символ не був вчасно лічений);
2	1 – помилка парності (ймовірно, із-за шуму в лінії);
3	1 – помилка оточення (передача не синхронізована);
4	1 – виявлена перерва (отриманий довгий рядок одиниць, що відображає, що інша станція запрошує кінець передачі);
5	1 – регістр зберігання передавача порожній (у цей регістр повинні поміщатися дані, що передані);
6	1 – регістр зрушення передавача порожній (цей регістр отримує дані з регістра зберігання і перетворить їх до послідовного вигляду);
7	1 – тайм-аут (пристрій не пов'язаний з машиною).

Для визначення статусу необхідно визначити базову адресу використовуваного порту, потім додати до нього 5 і застосувати команду введення для отримання даних з порту, які будуть байтом статусу, а потім виконати побітові операції, щоб інтерпретувати значення потрібного біта.

Передавання даних.

Передавання даних є простішим, ніж приймання, оскільки програма контролює склад даних та швидкість, з якою вони посилаються. Проте процедура передавання може бути досить складною, якщо вони обробляють дані у міру того, як вони надсилаються.

Під час передавання, коли байт даних розміщується до регістру зберігання передавача, він автоматично виводиться до послідовного каналу через регістр зсуву передавача, який перетворює дані у послідовний код. Немає необхідності в імпульсі біта стробу, як це робиться у випадку паралельного передавання даних.

Біт 5 регістра статусу лінії показує, чи вільний регістр зберігання передавача для прийому даних. Регістр постійно перевіряється доти, доки біт 5 не стане дорівнювати «1». Після цього в регістр зберігання передавача посилається черговий байт. В процесі передачі біт 5 дорівнює «0», і тільки коли він стане дорівнювати «1», до регістру зберігання передавача може бути надісланий наступний символ.

Цей процес повторюється до тих пір, поки в цьому є необхідність. У програмі використовуються команди введення та виведення даних через порт. Для асемблера – це команди IN(Port), OUT(Port), для мови СИ - це функції inp(Port), outp(Port, значення).

Отримання даних.

Як тільки комунікаційний порт ініціалізований та встановлений зв'язок з віддаленою станцією – комунікаційна програма готова до прийняття даних. Залежно від складності протоколу обміну дані, що приймаються, можуть вимагати простої або складної процедури обробки. Може бути отриманий будь-який код з набору кодів, що управляють.

9.4. Універсальний синхронно-асинхронний приймач-передавач (USART)

USART – це модуль послідовного введення/виведення, який може працювати в повному дуплексному асинхронному режимі для зв'язку з терміналами, персональними комп'ютерами, або в синхронному режимі – для зв'язку з мікросхемами ЦАП, АЦП, послідовними EEPROM тощо [4].

USART може працювати у трьох режимах:

- асинхронному, повному дуплексному;
- ведучому синхронному, напівдуплексному;
- веденому синхронному, напівдуплексному.

Біти SPEN (RCSTA<7>) та TRISC<7,6> мають бути встановлені в «1» для застосування виводів RC6/TX/CK RC7/RX/DT як порти універсального синхронно-асинхронного приймача-передавача.

Модуль USART підтримує режим детектування 9-розрядної адреси для роботи в мережевому режимі. Для роботи USART застосовуються регістри спеціального призначення TXSTA і RCSTA. Це регістри управління та статусу передавача (табл. 9.5) і приймача (табл. 9.6).

Таблиця 9.5

Біти регістра управління та статусу передавача TXSTA

Номер біта	Функція
Біт 7	CSRC: Вибір джерела тактового сигналу Синхронний режим 1 – ведучий, внутрішній тактовий сигнал від BRG 0 – відомий, зовнішній тактовий сигнал із входу CR Асинхронний режим Не має значення
Біт 6	TX9: Дозвіл 9-розрядної передачі 1 – 9-розрядна передача 0 – 8 розрядна передача
Біт 5	TXEN: Дозвіл передачі 1 – дозволена 0 – заборонена Примітка: В синхронному режимі біти SREN/CREN відміняють дію біта TXEN
Біт 4	SYNC: режим роботи USART 1 – синхронний 0 - асинхронний
Біт 3	Не застосовується, читається як 0
Біт 2	BRGH: вибір високошвидкісного режиму

	Синхронний режим Не має значення Асинхронний режим 1 - високошвидкісний режим 0 – низькошвидкісний режим
Біт 1	TRMT: прапор очищення зсувного регістра передавача TSR 1 – TSR порожній 0 – TSR повний
Біт 0	TX9D: 9-й біт даних, що передаються (може застосовуватися для програмної перевірки парності)

Таблиця 9.6

Біти регістра управління та статусу приймача RCSTA

Номер біта	Функція
Біт 7	SPEN: Дозвіл роботи послідовного порту 1 – модуль USART ввімкнутий (виводи RC7/RX/DT, RC6/TX/CK підключені до USART) 0 – модуль USART вимкнутий
Біт 6	KX9: Дозвіл 9-розрядної прийому 1 – 9-розрядна прийом 0 – 8 розрядна прийом
Біт 5	SREN: Дозвіл одиночного прийому Синхронний режим 1 – дозволений одиночний прийом 0 – заборонений одиночний прийом Скидається в 0 по закінченню прийому. Примітка: В режимі відомого не має значення. Асинхронний режим Не має значення
Біт 4	CREN: Дозвіл прийому Синхронний режим 1 – прийом дозволений (при встановленні біта CREN автоматично скидається біт SREN) 0 – прийом заборонений Асинхронний режим 1 – прийом дозволений 0 – прийом заборонений
Біт 3	ADDEN: Дозвіл детектування адреси Асинхронний 9-розрядний прийом (RX9=1) 1 – детектування адреси дозволено. Якщо біт RSR<8> = 1, то генерується переривання і завантажується прийомний буфер 0 – детектування адреси заборонено. Приймаються всі байти, дев’ятий біт може застосовуватися для перевірки парності.

Продовження табл. 9.6

Біт 2	FERR: Помилка кадру, скидається при читанні регістра RCREG 1 – відбулась помилка кадру 0 – помилки кадру не було
Біт 1	OERR: Помилка переповнення внутрішнього буфера, встановлюється в 0 при скиданні біта CREN 1 – відбулась помилка переповнення 0 – помилки переповнення не було
Біт 0	RX9D: 9-й біт прийнятих даних (може застосовуватися для програмної перевірки парності)

9.4.1. Генератор частоти обміну USART BRG

Генератор BRG застосовується для роботи USART у синхронному ведучому та асинхронному режимах. BRG є окремим 8-розрядним генератором швидкості обміну в бодах, період якого визначається значенням у регістрі SPBRG. В асинхронному режимі біт BRGH (TXSTA<2>) теж впливає на швидкість обміну (у синхронному режимі біт BRGH ігнорується). У табл. 9.7 наведені формули для обчислення швидкості обміну в бодах при різних режимах роботи модуля USART (відносно внутрішнього тактового сигналу мікроконтролера).

Враховуючи потрібну швидкість та F_{osc} , для запису до регістру SPBRG (табл. 9.8) обирається найближче ціле значення, обчислене за наведеними у табл. 9.7. формулами. Потім розраховується помилка швидкості обміну.

У деяких випадках може бути вигідним застосування високошвидкісного режиму роботи USART (BRGH=1), тому що рівняння $F_{osc}/(16(X+1))$ дозволяє зменшити похибку швидкості.

Запис нового значення до регістру SPBRG скидає таймер BRG, забезпечуючи негайний перехід на нову швидкість.

Таблиця 9.7

Формули для рахування швидкості обміну даними

SYNC	BRGH = 0	BRGH = 1
0	(Асинхронний) Швидкість обміну - $F_{osc}=(64(X+1))$	(Асинхронний) Швидкість обміну - $F_{osc}=(16(X+1))$
1	(Синхронний) Швидкість обміну - $F_{osc}(4(X+1))$	(Синхронний) Швидкість обміну - $F_{osc}=(4(X+1))$

X = значення регістра SPBRG (від 0 до 255)

Таблиця 9.8

Регістри і біти, які пов'язані з роботою генератора BRG

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
98h	TXTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	Регістр генератора швидкості USART								0000 0000	0000 0000

9.4.2. Вибірка даних

Сигнал з входу RC7/RX/DT опитується колом мажоритарного детектора три рази за такт передавання, щоб визначити високого чи низького рівня сигнал присутній на вході. Якщо обраний низькошвидкісний режим (BRGH=0), то вибірка здійснюється по сьомому, восьмому та дев'ятому зрізам тактового сигналу ×16. Якщо BRGH=1 (обраний високошвидкісний режим), вибірка здійснюється на другому такті сигналу ×4 трьома запитами.

9.4.3. Асинхронний режим USART

В цьому режимі USART стандартний формат NRZ: один стартовий біт, вісім або дев'ять бітів даних та один стоповий біт.

Найбільш часто зустрічається 8-розрядний формат передачі даних. Інтегрований 8-розрядний генератор BRG дозволяє отримати стандартні швидкості передачі даних. Генератор швидкості обміну може працювати в одному з двох режимів: високошвидкісному (×16 BRGH=1 TXTA<2>), або низькошвидкісному (×64 BRGH=0 TXTA<2>).

Приймач і передавач послідовного порту працюють незалежно один від одного, але застосовують той самий формат даних і однакову швидкість обміну.

Біт парності апаратно не підтримується, але може бути реалізований програмно із застосуванням 9-розрядного формату даних. Усі дані передаються молодшим бітом вперед. В SLEEP режимі мікроконтролера модуль USART (асинхронний режим) вимкнений. Вибір асинхронного режиму USART виконується скиданням біта SYNC в «0» (TXSTA <4>).

Модуль USART в асинхронному режимі складається з наступних елементів:

- генератор швидкості обміну;
- коло опитування;
- асинхронний передавач;
- асинхронний приймач.

9.4.4. Асинхронний передавач USART

Структурна схема асинхронного передавача USART наведена на рис. 9.7. Головним в передавачі є зсувний регістр TSR, який отримує дані з буфера передавача TXREG. Дані до регістру TXREG завантажуються програмно. Після передачі стопового біта попереднього байта (в останньому машинному такті циклу BRG) TSR завантажується новим значенням з TXREG (якщо воно присутнє), після чого встановлюється прапор переривання TXIF (PIR<4>).

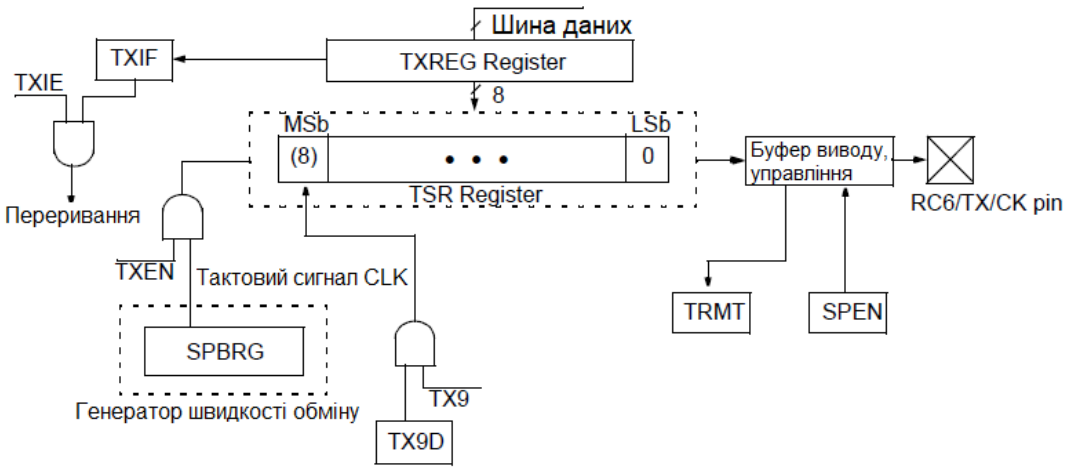


Рис 9.7. Структурна схема асинхронного передавача USART [4]

Переривання може бути дозволене або заборонене бітом TXIE (PIE<4>). Прапор TXIF встановлюється незалежно від стану біта TXIE і не може бути скинутий в «0» програмно. Очищення прапора TXIF відбувається тільки після завантаження нових даних до регістру TXREG. За аналогією біт TRMT (TXSTA<1>) відображає стан регістра TSR. Біт TMRT доступний тільки для читання і не може спричинити генерацію переривання.

Зауважимо, що регістр TSR не відображається на пам’ять і недоступний для читання. Прапор TXIF встановлюється в «1» тільки тоді, коли біт TXEN=1 і скидається автоматично в «0» після завантаження нових даних до регістру TXREG (табл. 9.9).

Таблиця 9.9

Регістри та біти, що пов'язані із роботою передавача USART в асинхронному режимі

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMF2IF	TMR1F	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMF2IE	TMR1E	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
19h	TXREG	Регістр даних передавача USART								0000 0000	0000 0000
98h	TXTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Регістр генератора швидкості USART								0000 0000	0000 0000

Для дозволу передавання необхідно встановити біт TXEN (TXSTA<5>) у стан «1». Передавання даних не почнеться доки в TXREG не будуть завантажені нові дані; не прийде черговий тактовий імпульс від генератора BRG (рис. 9.8,а). Можна спочатку завантажити дані до TXREG, а потім встановити біт TXEN. Зазвичай, після дозволу передавання регістр TSR порожній. Таким чином, дані, які записуються до TXREG одразу передаються до TSR, а TXREG залишається порожнім. Це дозволяє реалізувати передачу даних разом (рис. 9.8,б). Скидання біта TXEN в «0» призведе до негайного припинення передавання, скидання передавача та переведення виводу RC6/TX/CK у третій стан.

Для дозволу 9-розрядного передавання необхідно встановити біт TX9 (TXSTA<6>) в «1». Дев'ятий біт даних записується до біту TX9 (TXSTA<0>). Дев'ятий біт даних має бути призначений до запису до регістру TXREG, тому що дані, що записані до регістру TXREG можуть бути одразу завантажені до зсувного регістру TSR, якщо він порожній.

Рекомендована послідовність дій для передавання даних в асинхронному режимі:

1) встановити потрібну швидкість передавання за допомогою регістра SPBRG та біта BRGH;

2) обрати асинхронний режим скиданням біта SYNC в «0» та встановленням біта SPEN в «1»;

3) дозволити, якщо необхідно, переривання встановленням біта TXIE в «1»;

4) встановити біт TX9 в «1», якщо передавання 9-розрядне;

5) дозволити передавання встановленням біта TXEN в «1», автоматично встановлюється прапор TXIE.

6) записати 9-й біт даних до TX9D, якщо передавання 9-розрядне;

7) записати дані до регістру TXREG;

8) встановити біти GIE і PEIE у регістрі INTCON в «1», якщо застосовуються переривання.

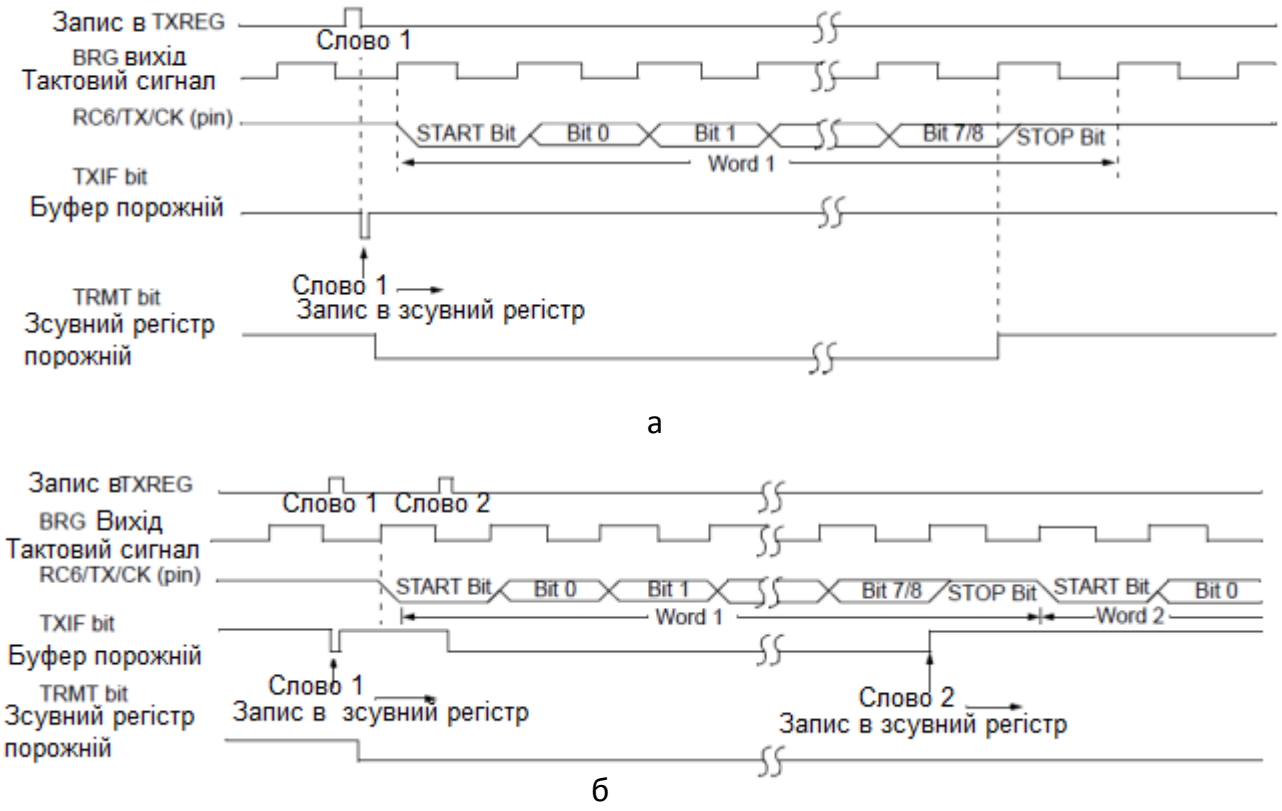


Рис. 9.8. Часова діаграма передавання даних USART [4]

9.4.5. Асинхронний приймач USART

Структурна схема асинхронного приймача USART наведена на рис. 9.9. Часові діаграми приймання даних USART наведені на рис. 9.10. Дані надсилаються на вхід RC7/RX/DT до блоку відновлення даних, який являє собою зсувний регістр, що працює на частоті у 16 разів більшій за швидкість передавання або F_{osc}.

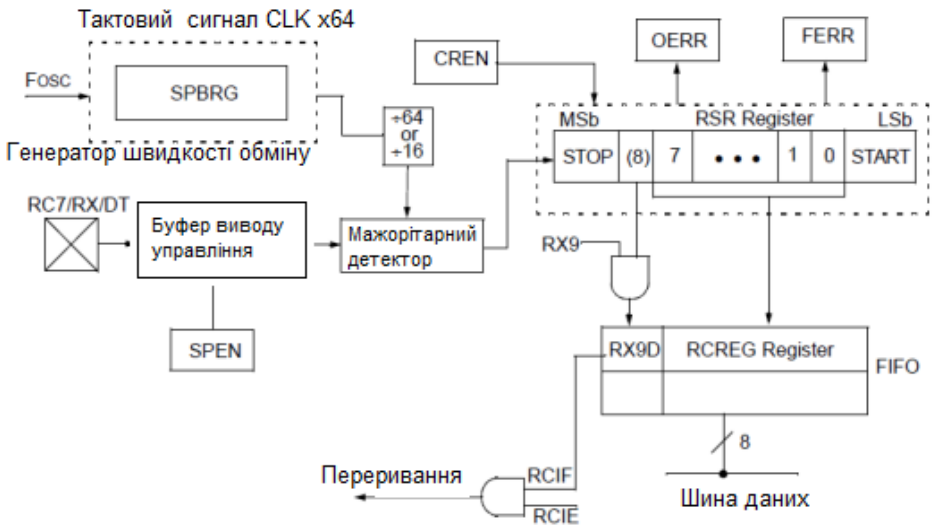


Рис. 9.9. Структурна схема асинхронного приймача USART [4]

Вмикання приймача здійснюється встановленням біта CREN регістра RCSTA в «1».

Головним в приймачі є зсувний регістр RSR. Після отримання стопового біту дані переписуються до регістру RCREG, якщо він порожній. Після запису до регістру RCREG встановлюється прапор переривання RCIF (PIR1<5>).

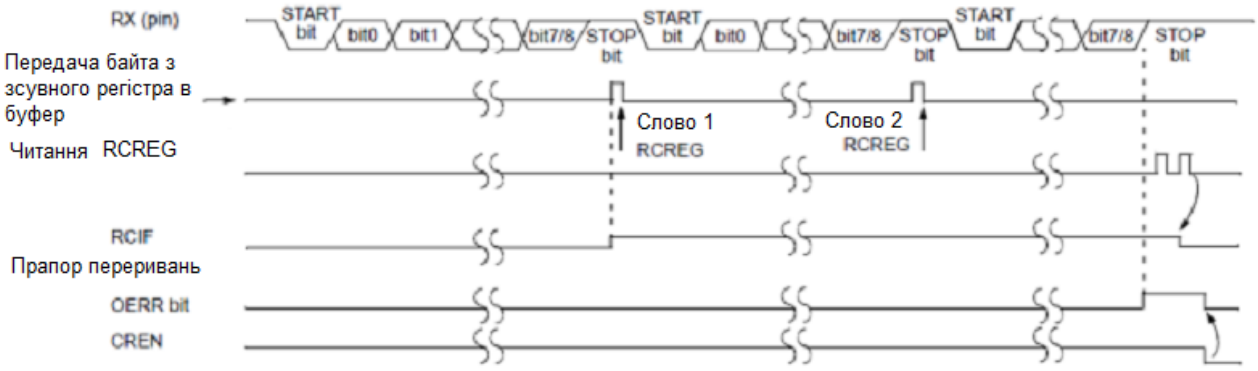


Рис. 9.10. Часові діаграми приймання даних USART [4]

Переривання можна дозволити або заборонити встановленням або скиданням біта RCIF (PIE<5>). Прапор RCIF, доступний тільки для читання, скидається апаратно під час читання з регістру RCREG (табл. 9.10).

Таблиця 9.10

Регістри та біти, що пов'язані із роботою приймача USART в асинхронному режимі

Адреса	Ім'я	Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0	Скидання POR,BOR	Інші скидання
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMF2IF	TMR1F	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMF2IE	TMR1E	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
1Ah	RCREG	Регістр даних приймача USART								0000 0000	0000 0000
98h	TXTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Регістр генератора швидкості USART								0000 0000	0000 0000

Регістр RCREG має подвійну буферизацію, тобто являє собою дворівневий буфер FIFO. Тому можна прийняти 2 байта даних до FIFO RCREG і третій до регістру RSR. Якщо FIFO заповнений і знайдений стоповий біт третього байта, встановлюється біт переповнення приймача OERR (RCSTA<1>). Байт, що прийнятий до RSR, буде загублений. Для діставання двох байт з FIFO необхідно двічі прочитати регістр RCREG. Біт OERR треба програмно очистити скиданням біта CREN, тобто забороною прийому. У будь-якому випадку, якщо біт OERR встановлений, логіка приймача вимкнена.

Біт помилки кадру FERR (RCSTA<2>) встановлюється в «1», якщо не знайдений стоповий біт. FERR та дев'ятий біт прийнятих даних буферизується так саме, як дані, що прийняті. Рекомендується спочатку прочитати регістр RCSTA, а потім RCREG, щоб не загубити інформацію RX9D і FERR.

Рекомендована послідовність дій при прийманні даних в асинхронному режимі:

1) встановити потрібну швидкість передачі за допомогою регістра SPBRG та біта BRGH;

2) обрати асинхронний режим скиданням біта SYNC в «0» та встановленням біта SPEN в «1».

3) дозволити переривання встановленням біта RCIE в «1», якщо в цьому є потреба;

4) встановити біт RX9 в «1», якщо передавання 9-розрядне;

5) дозволити прийом встановленням біта CREN в «1»;

6) чекати на встановлення біта RCIF, або переривання, якщо воно дозволене бітом RCIE;

7) зчитати 9-й біт даних (якщо дозволений 9-розрядний прийом) з регістру RCSTA та перевірити наявність помилки;

8) зчитати 8 біт даних з регістру RCREG;

9) скинути біт CREN в «0» при появі помилки переповнення;

10) якщо застосовуються переривання, біти GIE та PEIE в регістрі INTCON мають бути встановлені в «1».

9.5. Інтерфейс шина I2C

Для збільшення ефективності, спрощення схемотехнічних рішень PHILIPS розробила двоспрямовану двопровідну шину для так званого міжмікросхемного (*inter-I*C) управління, яка одержала назву – InterIC, або IIC (I2C) шина [10,11].

Наразі тільки PHILIPS виготовляє понад 150 найменувань I2C-сумісних пристроїв, що функціонально призначені для роботи в різному електронному обладнанні. Серед них IC-пам'ять відеопроцесорів та модулів обробки аудіо- та відеосигналів, АЦП та ЦАП, драйвери РК-індикаторів, процеси із вбудованим апаратним контролером I2C шини тощо.

I2C-шина є однією з модифікацій послідовних протоколів обміну даними. У стандартному режимі забезпечується передавання послідовних 8-бітових даних зі швидкістю до 100 кбіт/с та до 400 кбіт/с у "швидкому" режимі. Для здійснення процесу обміну інформацією I2C-шиною використовується лише два сигнали: лінія даних SDA тьа лінія синхронізації SCL. Для забезпечення реалізації двоспрямованості шини без застосування складних арбітрів шини вихідні каскади пристроїв, що підключені до шини, мають відкритий стик або відкритий колектор для забезпечення функції монтажного «AND».

Проста двопровідна послідовна шина I2C мінімізує кількість з'єднань між інтегральними схемами, які можуть мати меншу кількість контактів і потребувати меншої кількості друкованих доріжок на платі. В результаті друковані плати стають простішими та технологічнішими для виготовлення. Інтегрований I2C-протокол усуває необхідність дешифраторів адреси та іншої зовнішньої логіки узгодження.

Мінімальна допустима кількість мікросхем, що приєднуються до однієї шини, обмежується максимальною ємністю шини 400 пФ.

Вбудований у мікросхеми апаратний алгоритм пригнічення перешкод забезпечує цілісність даних в умовах перешкод значної величини.

Усі I2C-сумісні пристрої мають інтерфейс, який дозволяє їм зв'язуватися один з одним шиною навіть у тому випадку, якщо їхня напруга живлення істотно відрізняється. На рис. 9.11 наведене підключення кількох інтегральних мікросхем з різними напругами живлення до однієї шини обміну.

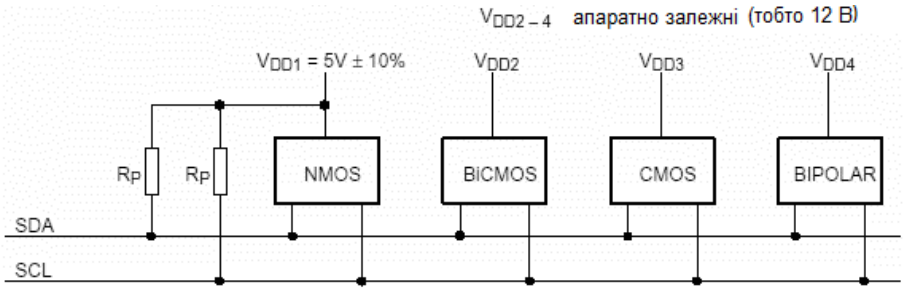


Рис. 9.11. Підключення кількох інтегральних мікросхем з різними напругами живлення до шини I2C

Кожний пристрій розпізнається за унікальною адресою і може працювати як передавач або приймач залежно від призначення.

До того ж пристрої можуть бути класифіковані як ведучі та ведені при передаванні даних. При цьому ведучий пристрій ініціює передавання даних та виробляє сигнали синхронізації, а ведений адресується ведучим..

Виходячи зі специфікації роботи шини, у кожний окремий момент у шині може бути тільки один ведучий, а саме той пристрій, який забезпечує формування сигналу шини SCL. Ведучий може виступати як у ролі ведучого-передавача, так і ведучого-приймача. Проте шина дозволяє мати кілька ведучих пристроїв, накладаючи певні особливості на їхню поведінку щодо формування сигналів управління та контролю стану шини. Можливість підключення до шини більше одного ведучого пристрою спричиняє небезпеку того, що кілька ведучих пристроїв можуть одночасно спробувати почати пересилання даних. Для усунення «зіткнень», що можуть виникнути в даному випадку, розроблено процедуру арбітражу – поведінки ведучого пристрою при виявленні факту «захоплення» шини іншим ведучим пристроєм.

Процедура синхронізації двох пристроїв полягає в тому, що всі пристрої I2C підключаються за правилом монтажного AND. У вихідному стані обидва сигнали SDA та SCL перебувають у високому стані.

9.5.1. Стани СТАРТ та СТОП

Процедура обміну починається з того, що ведучий пристрій при «високому» рівні на лінії SCL формує стан СТАРТ: генерує перехід сигналу лінії SDA з «високого» рівня до «низького» (рис.9.12). Цей перехід сприймається всіма пристроями, підключеними до шини, як ознака початку процедури обміну.

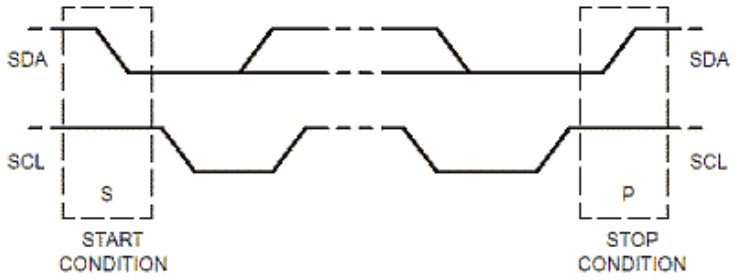


Рис. 9.12. Початок процедури обміну на шині I2C

Кожний ведучий пристрій при пересиланні даних шиною генерує власний сигнал синхронізації. Генерація синхросигналу є «обов'язком» ведучого пристрою.

Процедура обміну завершується тим, що ведучий пристрій формує стан СТОП: перехід лінії SDA від «низького» рівня до «високого» при «високому» рівню лінії SCL.

Стани СТАРТ та СТОП завжди формуються ведучим пристроєм. Зайнята після фіксації стану СТАРТ шина вважається такою, що вивільнилась, через деякий час після фіксації стану СТОП.

Рис. 9.13. Передавання даних на шиніI2C

При передачі посилок по шині I2C кожен ведучий пристрій генерує синхросигнал на лінії SCL.

Після формування стану СТАРТ, ведучий пристрій скидає стан лінії SCL на «низький» рівень і виставляє на лінію SDA старший біт першого байта повідомлення. Кількість байт у повідомленні є необмеженою (рис.9.13).

Специфікація шини I2C дозволяє зміни на лінії SDA тільки за умови «низького» рівня сигналу на лінії SCL.

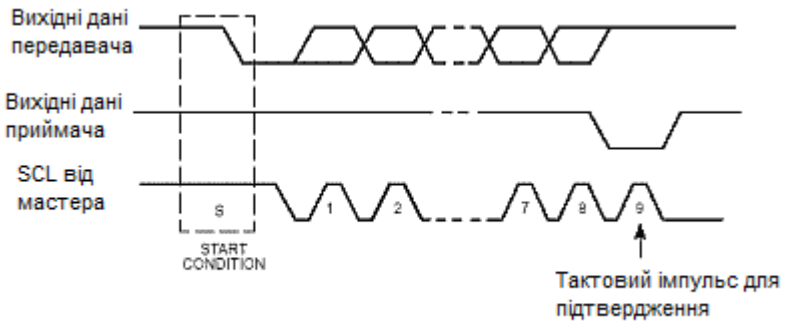


Рис. 9.14. Приймання даних на шині I2C

Дані дійсні та повинні залишатися стабільними лише під час «високого» рівня синхроімпульсу.

Для підтвердження прийому байта від ведучого-передавача веденим-приймачем у специфікації протоколу обміну на шині I2C вводиться спеціальний біт підтвердження, що виставляється на шину SDA після приймання 8 біт даних (рис. 9.14).

9.5.2. Підтвердження передачі даних

Таким чином, передача 8 біт даних від передавача до приймача завершуються додатковим циклом (формуванням 9 тактового імпульсу лінії SCL), у якому приймач виставляє «низький» рівень сигналу на лінії SDA, як ознаку успішного приймання байта.

Підтвердження передачі даних є обов'язковим. Відповідний імпульс синхронізації генерується ведучим пристроєм. Передавач встановлює лінію SDA у «високий» рівень під час синхроімпульсу підтвердження. Приймач повинен утримувати лінію SDA протягом «високого» рівня синхроімпульсу підтвердження у стабільному «низькому» рівні.

У тому випадку, якщо ведений приймач не може підтвердити свою адресу (наприклад, якщо він виконує в даний момент будь-які функції реального часу), лінія даних повинна бути залишена у «високому» стані. Після цього, якщо в пересиланні бере участь ведучий-приймач, він повинен повідомити про закінчення передавання веденому-передавачу шляхом непідтвердження останнього байта. Ведений-передавач повинен звільнити лінію даних для того, щоб дозволити ведучому видати сигнал СТОП або повторити сигнал СТАРТ.

Синхронізація виконується за допомогою підключення до лінії SCL за правилом монтажного AND.

Це означає, що ведучий пристрій не має монопольного права управління переходом лінії SCL з «низького» рівня до «високого». У тому випадку, якщо веденому пристрою необхідний додатковий час на обробку прийнятого біта, він може утримувати лінію SCL на «низькому» рівні до моменту готовності до приймання наступного біта. Таким чином, лінія SCL перебуватиме на «низькому» рівні протягом найдовшого «низького» періоду синхросигналу.

Пристрої з більш коротким «низьким» періодом будуть переходити до стану очікування на час, доки не закінчиться довгий період. Якщо у всіх задіяних пристроїв закінчиться «низький» період синхросигналу, лінія SCL перейде до «високого» рівеня сигналу. Усі пристрої почнуть проходити «високий» період своїх синхросигналів. Перший пристрій, у якого закінчиться цей період, знову встановить лінію SCL у «низький» стан.

Таким чином, «низький» період синхронізації SCL визначається найдовшим періодом синхронізації з усіх задіяних пристроїв, а «високий» період визначається найкоротшим періодом синхронізації пристроїв.

Механізм синхронізації може бути використаний приймачами як засіб керування пересиланням даних на байтовому та бітовому рівнях.

На рівні байта, якщо пристрій може прийняти байти даних з великою швидкістю, але вимагає певного часу для збереження прийнятого байта або підготовки до прийому наступного, він може утримувати лінію SCL на «низькому» рівні після приймання та підтвердження байта, переводячи, таким чином, передавач у стан очікування.

На рівні бітів такий пристрій, як мікроконтролер, без вбудованих апаратних кіл I2C або з обмеженими колами може уповільнити частоту синхроімпульсів шляхом продовження їх «низького» періоду. Таким чином,

швидкість передачі будь-якого ведучого пристрою адаптується до швидкості повільного пристрою.

9.5.3. Адресація у шині I2C

Кожний пристрій, підключений до шини, може бути програмно адресований унікальною адресою.

Для вибору приймача повідомлення ведучий пристрій використовує унікальну адресну компоненту у форматі посилки. При використанні однотипних пристроїв, інтегральні схеми часто мають додатковий селектор адреси, який може бути реалізований як додаткові цифрові входи селектора адреси, так і у вигляді аналогового входу. При цьому адреси таких однотипних пристроїв виявляються рознесені в просторі адресних пристроїв, підключених до шини.

У звичайному режимі використовується 7-бітова адресація.

Процедура адресації на шині I2C полягає у тому, що перший байт після сигналу СТАРТ визначає, який ведений пристрій адресується ведучим щодо циклу обміну. Виняток становить адреса «Загального виклику», яка адресує усі пристрої на шині. Якщо використовується ця адреса, усі пристрої в теорії повинні надіслати сигнал підтвердження. Проте, на практиці випадки, коли пристрої можуть обробляти «загальний виклик», зустрічаються рідко.

Перші сім бітів першого байта утворюють адресу веденого пристрою. Восьмий, молодший байт, визначає напрямок пересилання даних: «0» означає, що ведучий пристрій буде записувати інформацію до вибраного веденого; «1» означає, що ведучий пристрій буде зчитувати інформацію з веденого.

Після того, як адреса надіслана, кожний пристрій в системі порівнює перші сім біт після сигналу СТАРТ із власною адресою. У випадку збігу адрес пристрій вважає себе обраним як ведений-приймач або як ведений-передавач залежно від біта напряму.

Адреса веденого пристрою може складатися з фіксованої та програмованої частини.

Часто трапляється, що в системі буде кілька однотипних пристроїв (наприклад, інтегральних мікросхем пам’яті, або драйверів LED-індикаторів). За допомогою програмованої частини адреси стає можливим підключити до шини максимально можливу кількість таких пристроїв. Кількість програмованих біт на адресу залежить від кількості вільних виводів

мікросхеми. Іноді використовується один вивід з аналоговою установкою програмованого діапазону адреси, як це, наприклад, реалізовано в інтегральної мікросхемі SAA1064. При цьому залежно від потенціалу на цьому адресному виводі інтегральної мікросхеми, можливий зсув адресного простору драйвера так, щоб однотипні мікросхеми ІМС не конфліктували між собою на спільній шині.

Усі мікросхеми, що підтримують роботу у стандарті шини I2C, мають набір фіксованих адрес, перелік яких зазначений виробником в описі контролерів.

Комбінація біт 11110XX адреси зарезервована для 10-бітної адресації.

Загалом процес обміну на шині від моменту формування стану СТАРТ до стану СТОП ілюструється рис. 9.15).

Як випливає зі специфікації шини, допускаються як прості формати обміну, так і комбіновані, коли у проміжку від стану СТАРТ до стану СТОП ведучий та ведений пристрої можуть виступати як приймачем, так і передавачем даних.

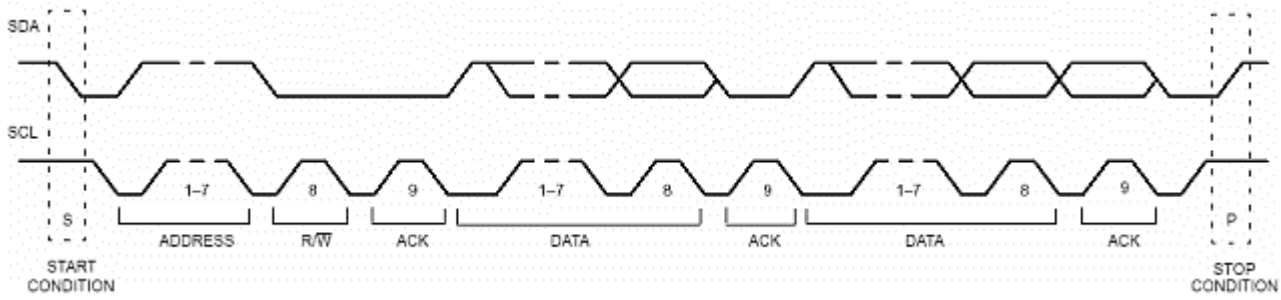


Рис. 9.15. Процес обміну на шині I2C

Комбіновані формати можуть бути використані, наприклад, для керування послідовною пам'яттю. Під час першого байта даних можна передавати адресу до пам'яті, яка записується у внутрішній регістр-заскочку. Після повторення сигналу СТАРТ та адреси веденого пристрою видаються дані з пам'яті. Усі рішення про авто-інкремент або декремент адреси, до якої відбувся попередній доступ, приймаються конструктором конкретного пристрою. Тому, у будь-якому випадку, найкращий спосіб уникнути неконтрольованої ситуації на шині полягає у тім, щоб перед використанням нової або раніше не використовуваної ІМС ретельно вивчити її опис

(*datasheet*), отримавши його з сайту виробника. До того ж виробники часто розміщують докладні інструкції щодо застосування.

У будь-якому випадку, за специфікацією шини усі пристрої, що розробляються, повинні скидати логіку шини при отриманні сигналу СТАРТ або повторний СТАРТ і готуватися до приймання адреси.

Проте основні проблеми з використанням I2C-шини виникають саме через те, що розробники, які «починають» працювати з I2C-шиною, не враховують того факту, що ведучий пристрій (часто мікропроцесор) не має монопольного права на жодну з ліній шини.

9.5.4. Розширення I2C

Стандартна шина I2C зі швидкістю передачі даних 100 кбіт/с та 7-бітною адресою існує вже протягом більше 10 років у незмінному вигляді. Стандартна шина I2C прийнята повсюдно як стандарт для сотень типів мікросхем, які випускають фірма PHILIPS та інші постачальники. Наразі специфікація шини I2C розширена у двох напрямках: збільшення швидкодії та розширення адресного простору для розширення номенклатури новостворених пристроїв.

Введення специфікації «швидкого» режиму дозволяє у чотири рази збільшити швидкість передавання даних до 400 кбіт/с. Необхідність у цьому «розширенні» стандарту була потрібна через необхідність пересилання великих обсягів інформації, і, як наслідок, необхідність збільшення пропускної спроможності каналу.

Уведення специфікації «10-бітної адресації» дозволяє використовувати 1024 додаткових адрес, оскільки більшість зі 112 адрес, допустимих при 7-бітній адресації, вже були використані більш ніж один раз. Для запобігання проблемам із розміщенням адрес нових пристроїв, бажано мати більшу кількість адресних комбінацій. Завдяки використанні нової 10-бітової адресації здійснене майже десятикратне збільшення кількості доступних адрес.

Усі нові пристрої з I2C-інтерфейсом працюють у швидкому режимі. Переважно, вони повинні вміти приймати та/або передавати дані на швидкості 400 кбіт/с. Як мінімум вони повинні бути здатні входити у синхронізацію у швидкому режимі, щоб знизити швидкість передавання (шляхом подовження «низького» періоду SCL) до допустимої величини.

Швидкі пристрої зазвичай сумісні «знизу-вгору», що означає їхню здатність працювати зі стандартними пристроями на повільній шині. Очевидно, що стандартні пристрої не здатні працювати на швидкій шині, тому що вони не можуть синхронізуватися на високій швидкості і їх стан стає непередбачуваним.

Відомі швидкі пристрої можуть мати як 7-бітну, так і 10-бітну адресу. Однак 7-бітна адреса є кращою, адже її апаратна реалізація є простішою, а довжина посилки – меншою. Пристрої із 7-бітною та 10-бітною адресами можуть одночасно використовуватись на одній шині незалежно від швидкості передачі. Як існуючі, так і майбутні ведучі пристрої зможуть генерувати як 7-бітні, так і 10-бітні адреси.

У швидкому режимі протокол, формат, логічні рівні та максимальне ємнісне навантаження ліній шини залишаються незмінними. Алгоритм синхронізації ліній SDA та SCL не змінено. Однак, від «швидких» пристроїв не вимагається сумісності з пристроями CBUS, оскільки вони не можуть працювати на високих швидкостях.

Вхідні кола швидких пристроїв повинні мати вбудоване придушення викидів та тригер Шмідта на обох лініях. Вихідний буфер швидких пристроїв повинен мати каскад із керованим часом заднього фронту ліній SDA та SCL. Як правило, при зникненні напруги живлення швидких пристроїв виводи підключення до ліній I2C-шини повинні переходити у третій стан.

Зазнали змін і схемотехнічні рішення вихідних каскадів для забезпечення часу зростання переднього фронту (перехід від «низького» стану до «високого»). Якщо для навантажень шини до 200 пФ цю роль виконують підтягуючі резистори, то для навантажень від 200 пФ до 400 пФ цю функцію виконує джерело струму або схема на резисторах, що перемикаються, яка забезпечує «форсоване» перемикання ліній I2C-шини.

10-бітна адресація також не змінює формат шини. Для цього використовується зарезервована адресна комбінація 1111XXX перших семи біт першого байта. 10-бітна адресація не впливає на існуючу 7-бітну адресацію. Пристрої з 7-бітною та 10-бітною адресацією можуть бути підключені до однієї шини. Хоча є 8 можливих комбінацій послідовності 1111XXX, з них використовують лише чотири – 11110XX. Комбінації типу 11111XX зарезервовані для подальших покращень шини. Призначення бітів перших двох байтів 10-бітна адреса формується з перших двох байтів. Перші сім бітів першого байта є комбінацією виду 11110XX, де два молодші біти

(XX) є двома старшими (9 та 8) бітами 10-бітної адреси; восьмий біт першого байта – біт напряму. «0» у цьому биті означає, що ведучий пристрій збирається записувати інформацію до веденого, а «1» – що ведучий пристрій зчитуватиме інформацію з веденого. Якщо біт напрямку дорівнює «0», то другий байт містить 8 біт, що залишилися, 10-бітної адреси. Якщо біт напрямку дорівнює «1», то наступний байт містить дані, передані від ведучого пристрою.

Насамкінець слід зазначити, що стандарт I2C-шини досить просто реалізує арбітраж зіткнень – вирішує проблему одночасної ініціалізації шини кількома ведучими пристроями без втрати даних.

Контрольні запитання та завдання

1. Які типи інтерфейсів застосовуються в обчислювальній техніці?
2. Надайте визначення інтерфейсу.
3. В чому полягає інформаційна сумісність?
4. В чому полягає електрична сумісність?
5. В чому полягає конструктивна сумісність?
6. Які режими передавання даних застосовуються в послідовних інтерфейсах?
7. Опишіть формат передавання даних в інтерфейсі RS-232.
8. Яким чином здійснюється кадрова синхронізація?
9. Опишіть порядок застосування біту паритету.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Мікропроцесорна техніка : підручник / Ю. А. Якіменко та ін. ; за ред. Т. О. Терещенка. 2-ге вид. переробл. та допов. Київ : ІВЦ «Видавництво «Політехніка» ; «Кондор», 2004. 440 с.
2. Dirksen A. J. Microcomputers: What They are and How to Put Them to Productive Use! New York : Tab Books, 1982. 231 p.
3. Gibson G. A., Liu Y.-C. Microcomputers for Engineers and Scientists. 2nd ed. Hoboken : Prentice-Hall, 1987. 482 p.
4. DS30292C. PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers. Official edition. Chandler : Microchip Technology Inc., 2001. 218 p
5. Predko M. Programming and Customizing the PIC Microcontroller. 3rd Ed. McGraw Hill Professional : New York, 2007.
6. Tavernier Ch. Les microcontroleurs PIC. Applications. 2nd ed. Malakoff : DUNOD, 2000. 264 p.
7. Демиденко М. І., Руденко О. А. Навчальний посібник з дисципліни «Комп'ютерна схемотехніка та архітектура комп'ютерів» для студентів спеціальності 122 «Комп'ютерні науки». Полтава : Нац. ун-т ім. Юрія Кондратюка, 2023. 203 с.
8. The RS232 Standard. *CAMI Research*. URL: http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html (date of access: 13.11.2024).
9. Ковальчук М. Л., Ушенко Ю. О., Угрин Д. І. Архітектура комп'ютерів : навч. посіб. Чернівці : Чернівецький національний університет ім. Ю. Федьковича, 2022. 188 с.
10. Опис шини I2C. *ТОВ «ІТТ Лтд»*. URL: https://itt-ltd.com/reference/ref_i2c.html (дата звернення: 13.11.2024).
11. The I2C-Bus and How to Use It (Including Specifications). *I2C – What's That? – I2C Bus*. URL: https://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf (date of access: 15.11.2024).

Навчальне видання

Зінченко Михайло Дмитрович, Маначин Іван Олександрович,
Бурчак Андрій Анатолійович

МІКРОПРОЦЕСОРНА ТЕХНІКА

Навчальний посібник

Електронне видання

Відповідальний редактор М. О. Рибальченко
Комп'ютерна верстка О. Ю. Потап
Дизайн обкладинки О. Ю. Потап

Експертний висновок склав канд. техн. наук, доц. О. П. Єгоров

Зареєстровано НМВ УДУНТ (№ 6 від 10.12.2024)

Формат 60x84 1/16. Ум. друк. арк. 12,15. Обл.-вид. арк. 12,26.
Зам. № 11

Видавець: Український державний університет науки і технологій
вул. Лазаряна, 2, ауд. 2216, ауд. 263 (наукова бібліотека)
м. Дніпро, 49010.
Свідоцтво суб'єкта видавничої справи ДК № 7709 від 14.12.2022